Global Constraint Catalog
2nd Edition (revision a)

Nicolas Beldiceanu
École des Mines de Nantes
LINA & INRIA, 4 rue Alfred Kastler
BP-20722, FR-44307 Nantes Cedex 3, France

Mats Carlsson
SICS, Box 1263, SE-16 429 Kista, Sweden

Jean-Xavier Rampon
LINA, UMR 6241, 2 rue de la Houssinière
B.P. 92208, FR-44322 Nantes Cedex 3, France

SICS Technical Report T2012:03
ISSN: 1100-3154

Abstract: This report presents a catalogue of global constraints where each constraint is explicitly described in terms of graph properties and/or automata and/or first order logical formulae with arithmetic. When available, it also presents some typical usage as well as some pointers to existing filtering algorithms.

Keywords: constraint programming, global constraint, catalogue, graph, automaton, first order formula, meta-data, ontology, symmetry.

February 14, 2012

1Corresponding author, Email: Nicolas.Beldiceanu@mines-nantes.fr
## Contents

**Preface**  
i

**1 Getting started**  
1

**2 Describing Global Constraints**  
3

- 2.1 Describing the arguments of a global constraint  
- 2.1.1 Basic data types  
- 2.1.2 Compound data types  
- 2.1.3 Restrictions  
- 2.1.4 Declaring a global constraint  
- 2.1.5 Describing symmetries between arguments  
- 2.2 Describing global constraints in terms of graph properties  
- 2.2.1 Basic ideas and illustrative example  
- 2.2.2 Ingredients used for describing global constraints  
- 2.2.3 Graph constraint  
- 2.3 Describing global constraints in terms of automata  
- 2.3.1 Selecting an appropriate description  
- 2.3.2 Defining an automaton  
- 2.4 Reformulating global constraints as a conjunction  
- 2.5 Semantic links between global constraints  
- 2.5.1 Assignment dimension added  
- 2.5.2 Assignment dimension removed  
- 2.5.3 Attached to cost variant  
- 2.5.4 Common keyword  
- 2.5.5 Comparison swapped  
- 2.5.6 Cost variant  
- 2.5.7 Generalisation  
- 2.5.8 Hard version  
- 2.5.9 Implied by  
- 2.5.10 Implies  
- 2.5.11 Implies (if swap arguments)  
- 2.5.12 Implies (items to collection)  
- 2.5.13 Negation  
- 2.5.14 Part of system of constraints
3 Description of the Catalogue

3.1 Which global constraints are included? .............................................. 98
3.2 Which global constraints are missing? ................................................. 100
3.3 Searching in the catalogue ................................................................. 100
3.3.1 How to see if a global constraint is in the catalogue? ...................... 100
3.3.2 How to search for all global constraints sharing the same structure ..... 101
3.3.3 Searching all places where a global constraint is referenced .......... 102
3.3.4 Searching the mapping with a constraint of a concrete system .......... 103
3.4 Figures of the catalogue .................................................................... 103
3.5 Constraints argument patterns ............................................................ 105
3.5.1 Constraints with 1 argument ............................................................ 108
3.5.2 Constraints with 2 arguments ............................................................ 112
3.5.3 Constraints with 3 arguments ............................................................ 123
3.5.4 Constraints with 4 arguments ............................................................ 131
3.5.5 Constraints with 5 arguments ............................................................ 135
3.5.6 Constraints with 6 arguments ............................................................ 136
3.5.7 Constraints with 8 arguments ............................................................ 137
3.5.8 Constraints with 10 arguments .......................................................... 137
3.6 Meta-keywords attached to the keywords .......................................... 138
3.6.1 Application area .............................................................................. 138
3.6.2 Characteristic of a constraint ............................................................ 138
3.6.3 Combinatorial object ...................................................................... 139
3.6.4 Complexity ....................................................................................... 139
3.6.5 Constraint network structure ............................................................. 139
3.6.6 Constraint type ................................................................................ 140
3.6.7 Constraint arguments ..................................................................... 140
3.6.8 Filtering ............................................................................................ 140
3.6.9 Final graph structure ...................................................................... 141
3.6.10 Geometry ......................................................................................... 141
3.6.11 Heuristics ......................................................................................... 142
3.6.12 Miscellaneous .................................................................................. 142
3.6.13 Modelling ......................................................................................... 142
3.6.14 Modelling exercises ....................................................................... 144
3.6.15 Problems ......................................................................................... 145
3.6.16 Puzzles ............................................................................................ 145
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6.17</td>
<td>Symmetry</td>
<td>146</td>
</tr>
<tr>
<td>3.7</td>
<td>Keywords attached to the global constraints</td>
<td>147</td>
</tr>
<tr>
<td>3.7.1</td>
<td>3-dimensional-matching</td>
<td>147</td>
</tr>
<tr>
<td>3.7.2</td>
<td>3-SAT</td>
<td>147</td>
</tr>
<tr>
<td>3.7.3</td>
<td>Abstract interpretation</td>
<td>147</td>
</tr>
<tr>
<td>3.7.4</td>
<td>Acyclic</td>
<td>148</td>
</tr>
<tr>
<td>3.7.5</td>
<td>Aggregate</td>
<td>148</td>
</tr>
<tr>
<td>3.7.6</td>
<td>Air traffic management</td>
<td>150</td>
</tr>
<tr>
<td>3.7.7</td>
<td>Alignment</td>
<td>150</td>
</tr>
<tr>
<td>3.7.8</td>
<td>All different</td>
<td>151</td>
</tr>
<tr>
<td>3.7.9</td>
<td>Alpha-acyclic constraint network(2)</td>
<td>151</td>
</tr>
<tr>
<td>3.7.10</td>
<td>Alpha-acyclic constraint network(3)</td>
<td>152</td>
</tr>
<tr>
<td>3.7.11</td>
<td>Apportion</td>
<td>152</td>
</tr>
<tr>
<td>3.7.12</td>
<td>Arc-consistency</td>
<td>153</td>
</tr>
<tr>
<td>3.7.13</td>
<td>Arithmetic constraint</td>
<td>155</td>
</tr>
<tr>
<td>3.7.14</td>
<td>Array constraint</td>
<td>155</td>
</tr>
<tr>
<td>3.7.15</td>
<td>Assigning and scheduling tasks that run in parallel</td>
<td>156</td>
</tr>
<tr>
<td>3.7.16</td>
<td>Assignment</td>
<td>159</td>
</tr>
<tr>
<td>3.7.17</td>
<td>Assignment dimension</td>
<td>160</td>
</tr>
<tr>
<td>3.7.18</td>
<td>Assignment to the same set of values</td>
<td>163</td>
</tr>
<tr>
<td>3.7.19</td>
<td>At least</td>
<td>168</td>
</tr>
<tr>
<td>3.7.20</td>
<td>At most</td>
<td>168</td>
</tr>
<tr>
<td>3.7.21</td>
<td>Automaton</td>
<td>168</td>
</tr>
<tr>
<td>3.7.22</td>
<td>Automaton with array of counters</td>
<td>171</td>
</tr>
<tr>
<td>3.7.23</td>
<td>Automaton with counters</td>
<td>171</td>
</tr>
<tr>
<td>3.7.24</td>
<td>Automaton without counters</td>
<td>172</td>
</tr>
<tr>
<td>3.7.25</td>
<td>Autoref</td>
<td>173</td>
</tr>
<tr>
<td>3.7.26</td>
<td>Balanced assignment</td>
<td>173</td>
</tr>
<tr>
<td>3.7.27</td>
<td>Balanced tree</td>
<td>174</td>
</tr>
<tr>
<td>3.7.28</td>
<td>Berge-acyclic constraint network</td>
<td>174</td>
</tr>
<tr>
<td>3.7.29</td>
<td>Binary constraint</td>
<td>177</td>
</tr>
<tr>
<td>3.7.30</td>
<td>Bioinformatics</td>
<td>177</td>
</tr>
<tr>
<td>3.7.31</td>
<td>Bipartite</td>
<td>178</td>
</tr>
<tr>
<td>3.7.32</td>
<td>Bipartite matching</td>
<td>178</td>
</tr>
<tr>
<td>3.7.33</td>
<td>Bipartite matching in convex bipartite graphs</td>
<td>179</td>
</tr>
<tr>
<td>3.7.34</td>
<td>Boolean channel</td>
<td>179</td>
</tr>
<tr>
<td>3.7.35</td>
<td>Boolean constraint</td>
<td>180</td>
</tr>
<tr>
<td>3.7.36</td>
<td>Border</td>
<td>180</td>
</tr>
<tr>
<td>3.7.37</td>
<td>Bound-consistency</td>
<td>180</td>
</tr>
<tr>
<td>3.7.38</td>
<td>Business rules</td>
<td>181</td>
</tr>
<tr>
<td>3.7.39</td>
<td>Centered cyclic(1) constraint network(1)</td>
<td>182</td>
</tr>
<tr>
<td>3.7.40</td>
<td>Centered cyclic(2) constraint network(1)</td>
<td>182</td>
</tr>
<tr>
<td>3.7.41</td>
<td>Centered cyclic(3) constraint network(1)</td>
<td>183</td>
</tr>
<tr>
<td>3.7.42</td>
<td>Channel routing</td>
<td>183</td>
</tr>
<tr>
<td>3.7.43</td>
<td>Channelling constraint</td>
<td>184</td>
</tr>
<tr>
<td>3.7.44</td>
<td>Circuit</td>
<td>184</td>
</tr>
</tbody>
</table>
### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.7.91</td>
<td>Duplicated variables</td>
<td>218</td>
</tr>
<tr>
<td>3.7.92</td>
<td>Dynamic programming</td>
<td>218</td>
</tr>
<tr>
<td>3.7.93</td>
<td>Empty intersection</td>
<td>218</td>
</tr>
<tr>
<td>3.7.94</td>
<td>Entailment</td>
<td>219</td>
</tr>
<tr>
<td>3.7.95</td>
<td>Equality</td>
<td>219</td>
</tr>
<tr>
<td>3.7.96</td>
<td>Equality between multisets</td>
<td>220</td>
</tr>
<tr>
<td>3.7.97</td>
<td>Equivalence</td>
<td>220</td>
</tr>
<tr>
<td>3.7.98</td>
<td>Euler knight</td>
<td>220</td>
</tr>
<tr>
<td>3.7.99</td>
<td>Excluded</td>
<td>221</td>
</tr>
<tr>
<td>3.7.100</td>
<td>Extensible</td>
<td>221</td>
</tr>
<tr>
<td>3.7.101</td>
<td>Extension</td>
<td>224</td>
</tr>
<tr>
<td>3.7.102</td>
<td>Facilities location problem</td>
<td>224</td>
</tr>
<tr>
<td>3.7.103</td>
<td>Floor planning problem</td>
<td>224</td>
</tr>
<tr>
<td>3.7.104</td>
<td>Flow</td>
<td>227</td>
</tr>
<tr>
<td>3.7.105</td>
<td>Frequency allocation problem</td>
<td>233</td>
</tr>
<tr>
<td>3.7.106</td>
<td>Functional dependency</td>
<td>233</td>
</tr>
<tr>
<td>3.7.107</td>
<td>Geometrical constraint</td>
<td>236</td>
</tr>
<tr>
<td>3.7.108</td>
<td>Golomb ruler</td>
<td>237</td>
</tr>
<tr>
<td>3.7.109</td>
<td>Graph colouring</td>
<td>237</td>
</tr>
<tr>
<td>3.7.110</td>
<td>Graph constraint</td>
<td>237</td>
</tr>
<tr>
<td>3.7.111</td>
<td>Graph partitioning constraint</td>
<td>238</td>
</tr>
<tr>
<td>3.7.112</td>
<td>Guillotine cut</td>
<td>238</td>
</tr>
<tr>
<td>3.7.113</td>
<td>Hall interval</td>
<td>238</td>
</tr>
<tr>
<td>3.7.114</td>
<td>Hamiltonian</td>
<td>239</td>
</tr>
<tr>
<td>3.7.115</td>
<td>Heuristics</td>
<td>239</td>
</tr>
<tr>
<td>3.7.116</td>
<td>Heuristics and Berge-acyclic constraint network</td>
<td>239</td>
</tr>
<tr>
<td>3.7.117</td>
<td>Heuristics and lexicographical ordering</td>
<td>241</td>
</tr>
<tr>
<td>3.7.118</td>
<td>Heuristics for two-dimensional rectangle placement problems</td>
<td>241</td>
</tr>
<tr>
<td>3.7.119</td>
<td>Hungarian method for the assignment problem</td>
<td>243</td>
</tr>
<tr>
<td>3.7.120</td>
<td>Hybrid-consistency</td>
<td>243</td>
</tr>
<tr>
<td>3.7.121</td>
<td>Hypergraph</td>
<td>244</td>
</tr>
<tr>
<td>3.7.122</td>
<td>Included</td>
<td>244</td>
</tr>
<tr>
<td>3.7.123</td>
<td>Inclusion</td>
<td>244</td>
</tr>
<tr>
<td>3.7.124</td>
<td>Incompatible pairs of values</td>
<td>245</td>
</tr>
<tr>
<td>3.7.125</td>
<td>Indistinguishable values</td>
<td>245</td>
</tr>
<tr>
<td>3.7.126</td>
<td>Interval</td>
<td>245</td>
</tr>
<tr>
<td>3.7.127</td>
<td>Joker value</td>
<td>246</td>
</tr>
<tr>
<td>3.7.128</td>
<td>Klee’s measure problem</td>
<td>246</td>
</tr>
<tr>
<td>3.7.129</td>
<td>Labelling by increasing cost</td>
<td>246</td>
</tr>
<tr>
<td>3.7.130</td>
<td>Latin square</td>
<td>249</td>
</tr>
<tr>
<td>3.7.131</td>
<td>Lexicographic order</td>
<td>249</td>
</tr>
<tr>
<td>3.7.132</td>
<td>Limited discrepancy search</td>
<td>250</td>
</tr>
<tr>
<td>3.7.133</td>
<td>Linear programming</td>
<td>250</td>
</tr>
<tr>
<td>3.7.134</td>
<td>Line-segments intersection</td>
<td>252</td>
</tr>
<tr>
<td>3.7.135</td>
<td>Logic</td>
<td>252</td>
</tr>
<tr>
<td>3.7.136</td>
<td>Logigraphe</td>
<td>252</td>
</tr>
<tr>
<td>Section Number</td>
<td>Section Description</td>
<td>Page</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.7.137</td>
<td>Magic hexagon</td>
<td>254</td>
</tr>
<tr>
<td>3.7.138</td>
<td>Magic series</td>
<td>255</td>
</tr>
<tr>
<td>3.7.139</td>
<td>Magic square</td>
<td>255</td>
</tr>
<tr>
<td>3.7.140</td>
<td>Matching</td>
<td>255</td>
</tr>
<tr>
<td>3.7.141</td>
<td>Matrix</td>
<td>256</td>
</tr>
<tr>
<td>3.7.142</td>
<td>Matrix model</td>
<td>256</td>
</tr>
<tr>
<td>3.7.143</td>
<td>Matrix symmetry</td>
<td>256</td>
</tr>
<tr>
<td>3.7.144</td>
<td>Maximum</td>
<td>257</td>
</tr>
<tr>
<td>3.7.145</td>
<td>Maximum clique</td>
<td>257</td>
</tr>
<tr>
<td>3.7.146</td>
<td>Maximum number of occurrences</td>
<td>257</td>
</tr>
<tr>
<td>3.7.147</td>
<td>maxint</td>
<td>257</td>
</tr>
<tr>
<td>3.7.148</td>
<td>Metro</td>
<td>258</td>
</tr>
<tr>
<td>3.7.149</td>
<td>Minimum</td>
<td>260</td>
</tr>
<tr>
<td>3.7.150</td>
<td>Minimum cost flow</td>
<td>261</td>
</tr>
<tr>
<td>3.7.151</td>
<td>Minimum feedback vertex set</td>
<td>262</td>
</tr>
<tr>
<td>3.7.152</td>
<td>Minimum hitting set cardinality</td>
<td>262</td>
</tr>
<tr>
<td>3.7.153</td>
<td>Minimum number of occurrences</td>
<td>262</td>
</tr>
<tr>
<td>3.7.154</td>
<td>Modulo</td>
<td>262</td>
</tr>
<tr>
<td>3.7.155</td>
<td>Multi-site employee scheduling with calendar constraints</td>
<td>263</td>
</tr>
<tr>
<td>3.7.156</td>
<td>Multiset</td>
<td>265</td>
</tr>
<tr>
<td>3.7.157</td>
<td>Multiset ordering</td>
<td>265</td>
</tr>
<tr>
<td>3.7.158</td>
<td>No cycle</td>
<td>265</td>
</tr>
<tr>
<td>3.7.159</td>
<td>No loop</td>
<td>265</td>
</tr>
<tr>
<td>3.7.160</td>
<td>n-Amazon</td>
<td>266</td>
</tr>
<tr>
<td>3.7.161</td>
<td>n-queen</td>
<td>269</td>
</tr>
<tr>
<td>3.7.162</td>
<td>Non-deterministic automaton</td>
<td>269</td>
</tr>
<tr>
<td>3.7.163</td>
<td>Non-overlapping</td>
<td>269</td>
</tr>
<tr>
<td>3.7.164</td>
<td>Number of changes</td>
<td>270</td>
</tr>
<tr>
<td>3.7.165</td>
<td>Number of distinct equivalence classes</td>
<td>270</td>
</tr>
<tr>
<td>3.7.166</td>
<td>Number of distinct values</td>
<td>270</td>
</tr>
<tr>
<td>3.7.167</td>
<td>Obscure</td>
<td>271</td>
</tr>
<tr>
<td>3.7.168</td>
<td>One succ</td>
<td>271</td>
</tr>
<tr>
<td>3.7.169</td>
<td>Open automaton constraint</td>
<td>272</td>
</tr>
<tr>
<td>3.7.170</td>
<td>Open constraint</td>
<td>273</td>
</tr>
<tr>
<td>3.7.171</td>
<td>Order constraint</td>
<td>274</td>
</tr>
<tr>
<td>3.7.172</td>
<td>Orthotope</td>
<td>275</td>
</tr>
<tr>
<td>3.7.173</td>
<td>Overlapping alldifferent</td>
<td>275</td>
</tr>
<tr>
<td>3.7.174</td>
<td>Pair</td>
<td>275</td>
</tr>
<tr>
<td>3.7.175</td>
<td>Packing almost squares</td>
<td>276</td>
</tr>
<tr>
<td>3.7.176</td>
<td>Pallet loading</td>
<td>276</td>
</tr>
<tr>
<td>3.7.177</td>
<td>Partition</td>
<td>277</td>
</tr>
<tr>
<td>3.7.178</td>
<td>Path</td>
<td>277</td>
</tr>
<tr>
<td>3.7.179</td>
<td>Partridge</td>
<td>277</td>
</tr>
<tr>
<td>3.7.180</td>
<td>Pattern sequencing</td>
<td>278</td>
</tr>
<tr>
<td>3.7.181</td>
<td>Pentomino</td>
<td>279</td>
</tr>
<tr>
<td>3.7.182</td>
<td>Periodic</td>
<td>279</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.7.183</td>
<td>Permutation</td>
<td>279</td>
</tr>
<tr>
<td>3.7.184</td>
<td>Permutation channel</td>
<td>280</td>
</tr>
<tr>
<td>3.7.185</td>
<td>Phi-tree</td>
<td>280</td>
</tr>
<tr>
<td>3.7.186</td>
<td>Phylogeny</td>
<td>282</td>
</tr>
<tr>
<td>3.7.187</td>
<td>Pick-up delivery</td>
<td>282</td>
</tr>
<tr>
<td>3.7.188</td>
<td>Planarity test</td>
<td>282</td>
</tr>
<tr>
<td>3.7.189</td>
<td>Polygon</td>
<td>282</td>
</tr>
<tr>
<td>3.7.190</td>
<td>Positioning constraint</td>
<td>282</td>
</tr>
<tr>
<td>3.7.191</td>
<td>Predefined constraint</td>
<td>283</td>
</tr>
<tr>
<td>3.7.192</td>
<td>Preferences</td>
<td>284</td>
</tr>
<tr>
<td>3.7.193</td>
<td>Producer-consumer</td>
<td>284</td>
</tr>
<tr>
<td>3.7.194</td>
<td>Product</td>
<td>285</td>
</tr>
<tr>
<td>3.7.195</td>
<td>Program verification</td>
<td>285</td>
</tr>
<tr>
<td>3.7.196</td>
<td>Proximity constraint</td>
<td>285</td>
</tr>
<tr>
<td>3.7.197</td>
<td>Pure functional dependency</td>
<td>286</td>
</tr>
<tr>
<td>3.7.198</td>
<td>Quadtree</td>
<td>287</td>
</tr>
<tr>
<td>3.7.199</td>
<td>Range</td>
<td>288</td>
</tr>
<tr>
<td>3.7.200</td>
<td>Rank</td>
<td>288</td>
</tr>
<tr>
<td>3.7.201</td>
<td>RCC8</td>
<td>288</td>
</tr>
<tr>
<td>3.7.202</td>
<td>Rectangle clique partition</td>
<td>289</td>
</tr>
<tr>
<td>3.7.203</td>
<td>Regret based heuristics</td>
<td>289</td>
</tr>
<tr>
<td>3.7.204</td>
<td>Regret based heuristics in matrix problems</td>
<td>290</td>
</tr>
<tr>
<td>3.7.205</td>
<td>Reified automaton constraint</td>
<td>290</td>
</tr>
<tr>
<td>3.7.206</td>
<td>Reified constraint</td>
<td>292</td>
</tr>
<tr>
<td>3.7.207</td>
<td>Relation</td>
<td>293</td>
</tr>
<tr>
<td>3.7.208</td>
<td>Relaxation</td>
<td>293</td>
</tr>
<tr>
<td>3.7.209</td>
<td>Relaxation dimension</td>
<td>294</td>
</tr>
<tr>
<td>3.7.210</td>
<td>Resource constraint</td>
<td>295</td>
</tr>
<tr>
<td>3.7.211</td>
<td>Run of a permutation</td>
<td>296</td>
</tr>
<tr>
<td>3.7.212</td>
<td>SAT</td>
<td>296</td>
</tr>
<tr>
<td>3.7.213</td>
<td>Scalar product</td>
<td>297</td>
</tr>
<tr>
<td>3.7.214</td>
<td>Sequence</td>
<td>297</td>
</tr>
<tr>
<td>3.7.215</td>
<td>Sequence dependent set-up</td>
<td>298</td>
</tr>
<tr>
<td>3.7.216</td>
<td>Sequencing with release times and deadlines</td>
<td>299</td>
</tr>
<tr>
<td>3.7.217</td>
<td>Set channel</td>
<td>299</td>
</tr>
<tr>
<td>3.7.218</td>
<td>Set packing</td>
<td>300</td>
</tr>
<tr>
<td>3.7.219</td>
<td>Shikaku</td>
<td>300</td>
</tr>
<tr>
<td>3.7.220</td>
<td>Scheduling constraint</td>
<td>301</td>
</tr>
<tr>
<td>3.7.221</td>
<td>Scheduling with machine choice, calendars and preemption</td>
<td>301</td>
</tr>
<tr>
<td>3.7.222</td>
<td>Shared table</td>
<td>305</td>
</tr>
<tr>
<td>3.7.223</td>
<td>Schur number</td>
<td>306</td>
</tr>
<tr>
<td>3.7.224</td>
<td>SLAM problem</td>
<td>306</td>
</tr>
<tr>
<td>3.7.225</td>
<td>Sliding cyclic(1) constraint network(1)</td>
<td>306</td>
</tr>
<tr>
<td>3.7.226</td>
<td>Sliding cyclic(1) constraint network(2)</td>
<td>307</td>
</tr>
<tr>
<td>3.7.227</td>
<td>Sliding cyclic(1) constraint network(3)</td>
<td>307</td>
</tr>
<tr>
<td>3.7.228</td>
<td>Sliding cyclic(2) constraint network(2)</td>
<td>308</td>
</tr>
</tbody>
</table>
3.7.229 Sliding sequence constraint ........................................ 308
3.7.230 Smallest square for packing consecutive dominoes ............ 309
3.7.231 Smallest rectangle area ............................................ 310
3.7.232 Smallest square for packing rectangles with distinct sizes ..... 312
3.7.233 Soft constraint ....................................................... 314
3.7.234 Sort ................................................................. 314
3.7.235 Sort based reformulation ............................................ 314
3.7.236 Sparse functional dependency ................................... 315
3.7.237 Sparse table ......................................................... 315
3.7.238 Sport timetabling .................................................... 315
3.7.239 Squared squares .................................................... 315
3.7.240 Statistics ............................................................. 320
3.7.241 Strip packing ......................................................... 320
3.7.242 Strong articulation point ......................................... 321
3.7.243 Strong bridge ......................................................... 321
3.7.244 Strongly connected component ................................ 322
3.7.245 Subset sum .......................................................... 323
3.7.246 Sudoku ............................................................... 323
3.7.247 Sum ................................................................. 324
3.7.248 Sweep ............................................................... 324
3.7.249 Symmetric ........................................................... 328
3.7.250 Symmetry ............................................................ 328
3.7.251 System of constraints ............................................. 329
3.7.252 Table ................................................................. 330
3.7.253 Temporal constraint ............................................... 330
3.7.254 Ternary constraint ................................................ 330
3.7.255 Timetabling constraint ............................................ 331
3.7.256 Time window ....................................................... 331
3.7.257 Touch ............................................................... 331
3.7.258 Tree ................................................................. 332
3.7.259 Tuple ............................................................... 332
3.7.260 Two-dimensional orthogonal packing ............................ 332
3.7.261 Unary constraint .................................................... 333
3.7.262 Undirected graph .................................................. 333
3.7.263 Value constraint .................................................... 333
3.7.264 Value partitioning constraint ................................... 334
3.7.265 Value precedence .................................................. 335
3.7.266 Variable-based violation measure ................................ 335
3.7.267 Variable indexing .................................................. 335
3.7.268 Variable subscript ................................................ 335
3.7.269 Vector ............................................................... 336
3.7.270 Vpartition ............................................................ 336
3.7.271 Weighted assignment .............................................. 337
3.7.272 Workload covering ............................................... 337
3.7.273 Zebra puzzle ......................................................... 337
3.7.274 Zero-duration task ................................................ 342
# Further Topics

4.1 Differences from the 2000 report .............................................. 344
4.2 Differences from the 2005 report .............................................. 346
4.3 Graph invariants ................................................................. 347
  4.3.1 Graph classes ............................................................... 347
  4.3.2 Format of an invariant ..................................................... 348
  4.3.3 Using the database of invariants ....................................... 349
  4.3.4 The database of graph invariants ..................................... 350
4.4 The electronic version of the catalogue .................................... 399
  4.4.1 Prolog facts describing a constraint .................................. 399
  4.4.2 XML schema associated with a global constraint ................... 404

# Global Constraint Catalogue

5.1 abs_value ................................................................................. 420
5.2 all_differ_from_at_least_k_pos .................................................. 422
5.3 all_equal ................................................................................. 426
5.4 all_incomparable ..................................................................... 428
5.5 all_min_dist ............................................................................ 430
5.6 alldifferent .............................................................................. 434
5.7 alldifferent_between_sets ....................................................... 442
5.8 alldifferent_consecutive_values .............................................. 444
5.9 alldifferent_cst ....................................................................... 446
5.10 alldifferent_except_0 ............................................................ 450
5.11 alldifferent_interval .............................................................. 454
5.12 alldifferent_modulo ............................................................... 458
5.13 alldifferent_on_intersection .................................................. 462
5.14 alldifferent_partition ............................................................ 466
5.15 alldifferent_same_value ......................................................... 470
5.16 allperm ................................................................................ 474
5.17 among .................................................................................. 478
5.18 among_diff_0 ....................................................................... 486
5.19 among_interval ..................................................................... 490
5.20 among_low_up ..................................................................... 494
5.21 among_modulo ..................................................................... 498
5.22 among_seq .......................................................................... 502
5.23 among_var .......................................................................... 506
5.24 and ...................................................................................... 510
5.25 arith ..................................................................................... 514
5.26 arith_or ................................................................................. 518
5.27 arith_sliding ......................................................................... 522
5.28 assign_and_counts .................................................................. 526
5.29 assign_and_nvalues ............................................................... 530
5.30 atleast .................................................................................. 534
5.31 atmost .................................................................................. 538
5.32 atmost_nvector ..................................................................... 542
5.33
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.34</td>
<td>atmost1</td>
<td>550</td>
</tr>
<tr>
<td>5.35</td>
<td>atmost_value</td>
<td>552</td>
</tr>
<tr>
<td>5.36</td>
<td>atmost_vector</td>
<td>556</td>
</tr>
<tr>
<td>5.37</td>
<td>balance</td>
<td>560</td>
</tr>
<tr>
<td>5.38</td>
<td>balance_cycle</td>
<td>566</td>
</tr>
<tr>
<td>5.39</td>
<td>balance_interval</td>
<td>570</td>
</tr>
<tr>
<td>5.40</td>
<td>balance_modulo</td>
<td>574</td>
</tr>
<tr>
<td>5.41</td>
<td>balance_partition</td>
<td>578</td>
</tr>
<tr>
<td>5.42</td>
<td>balance_path</td>
<td>582</td>
</tr>
<tr>
<td>5.43</td>
<td>balance_tree</td>
<td>586</td>
</tr>
<tr>
<td>5.44</td>
<td>between_min_max</td>
<td>590</td>
</tr>
<tr>
<td>5.45</td>
<td>bin_packing</td>
<td>594</td>
</tr>
<tr>
<td>5.46</td>
<td>bin_packing_capa</td>
<td>600</td>
</tr>
<tr>
<td>5.47</td>
<td>binary_tree</td>
<td>602</td>
</tr>
<tr>
<td>5.48</td>
<td>bipartite</td>
<td>606</td>
</tr>
<tr>
<td>5.49</td>
<td>calendar</td>
<td>610</td>
</tr>
<tr>
<td>5.50</td>
<td>cardinality_atleast</td>
<td>620</td>
</tr>
<tr>
<td>5.51</td>
<td>cardinality_atmost</td>
<td>624</td>
</tr>
<tr>
<td>5.52</td>
<td>cardinality_atmost_partition</td>
<td>628</td>
</tr>
<tr>
<td>5.53</td>
<td>change</td>
<td>632</td>
</tr>
<tr>
<td>5.54</td>
<td>change_continuity</td>
<td>638</td>
</tr>
<tr>
<td>5.55</td>
<td>change_pair</td>
<td>650</td>
</tr>
<tr>
<td>5.56</td>
<td>change_partition</td>
<td>656</td>
</tr>
<tr>
<td>5.57</td>
<td>change_vectors</td>
<td>660</td>
</tr>
<tr>
<td>5.58</td>
<td>circuit</td>
<td>662</td>
</tr>
<tr>
<td>5.59</td>
<td>circuit_cluster</td>
<td>666</td>
</tr>
<tr>
<td>5.60</td>
<td>circular_change</td>
<td>672</td>
</tr>
<tr>
<td>5.61</td>
<td>clause_and</td>
<td>676</td>
</tr>
<tr>
<td>5.62</td>
<td>clause_or</td>
<td>680</td>
</tr>
<tr>
<td>5.63</td>
<td>clique</td>
<td>684</td>
</tr>
<tr>
<td>5.64</td>
<td>colored_matrix</td>
<td>688</td>
</tr>
<tr>
<td>5.65</td>
<td>coloured_cumulative</td>
<td>692</td>
</tr>
<tr>
<td>5.66</td>
<td>coloured_cumulatives</td>
<td>698</td>
</tr>
<tr>
<td>5.67</td>
<td>common</td>
<td>704</td>
</tr>
<tr>
<td>5.68</td>
<td>common_interval</td>
<td>708</td>
</tr>
<tr>
<td>5.69</td>
<td>common_modulo</td>
<td>712</td>
</tr>
<tr>
<td>5.70</td>
<td>common_partition</td>
<td>716</td>
</tr>
<tr>
<td>5.71</td>
<td>compare_and_count</td>
<td>720</td>
</tr>
<tr>
<td>5.72</td>
<td>cond_lex_cost</td>
<td>722</td>
</tr>
<tr>
<td>5.73</td>
<td>cond_lex_greater</td>
<td>726</td>
</tr>
<tr>
<td>5.74</td>
<td>cond_lex_greatereq</td>
<td>730</td>
</tr>
<tr>
<td>5.75</td>
<td>cond_lex_less</td>
<td>734</td>
</tr>
<tr>
<td>5.76</td>
<td>cond_lex_leseq</td>
<td>738</td>
</tr>
<tr>
<td>5.77</td>
<td>connect_points</td>
<td>742</td>
</tr>
<tr>
<td>5.78</td>
<td>connected</td>
<td>746</td>
</tr>
<tr>
<td>5.79</td>
<td>consecutive_groups_of_ones</td>
<td>748</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>5.80</td>
<td>consecutive_values</td>
<td></td>
</tr>
<tr>
<td>5.81</td>
<td>contains_sboxes</td>
<td></td>
</tr>
<tr>
<td>5.82</td>
<td>correspondence</td>
<td></td>
</tr>
<tr>
<td>5.83</td>
<td>count</td>
<td></td>
</tr>
<tr>
<td>5.84</td>
<td>counts</td>
<td></td>
</tr>
<tr>
<td>5.85</td>
<td>coveredby_sboxes</td>
<td></td>
</tr>
<tr>
<td>5.86</td>
<td>covers_sboxes</td>
<td></td>
</tr>
<tr>
<td>5.87</td>
<td>crossing</td>
<td></td>
</tr>
<tr>
<td>5.88</td>
<td>cumulative</td>
<td></td>
</tr>
<tr>
<td>5.89</td>
<td>cumulative_convex</td>
<td></td>
</tr>
<tr>
<td>5.90</td>
<td>cumulative_product</td>
<td></td>
</tr>
<tr>
<td>5.91</td>
<td>cumulative_two_d</td>
<td></td>
</tr>
<tr>
<td>5.92</td>
<td>cumulative_with_level_of_priority</td>
<td></td>
</tr>
<tr>
<td>5.93</td>
<td>cumulatives</td>
<td></td>
</tr>
<tr>
<td>5.94</td>
<td>cutset</td>
<td></td>
</tr>
<tr>
<td>5.95</td>
<td>cycle</td>
<td></td>
</tr>
<tr>
<td>5.96</td>
<td>cycle_card_on_path</td>
<td></td>
</tr>
<tr>
<td>5.97</td>
<td>cycle_or_accessibility</td>
<td></td>
</tr>
<tr>
<td>5.98</td>
<td>cycle_resource</td>
<td></td>
</tr>
<tr>
<td>5.99</td>
<td>cyclic_change</td>
<td></td>
</tr>
<tr>
<td>5.100</td>
<td>cyclic_change_joker</td>
<td></td>
</tr>
<tr>
<td>5.101</td>
<td>dag</td>
<td></td>
</tr>
<tr>
<td>5.102</td>
<td>decreasing</td>
<td></td>
</tr>
<tr>
<td>5.103</td>
<td>deepest_valley</td>
<td></td>
</tr>
<tr>
<td>5.104</td>
<td>derangement</td>
<td></td>
</tr>
<tr>
<td>5.105</td>
<td>differ_from_at_least_k_pos</td>
<td></td>
</tr>
<tr>
<td>5.106</td>
<td>diffn</td>
<td></td>
</tr>
<tr>
<td>5.107</td>
<td>diffn_column</td>
<td></td>
</tr>
<tr>
<td>5.108</td>
<td>diffn_include</td>
<td></td>
</tr>
<tr>
<td>5.109</td>
<td>discrepancy</td>
<td></td>
</tr>
<tr>
<td>5.110</td>
<td>disj</td>
<td></td>
</tr>
<tr>
<td>5.111</td>
<td>disjint</td>
<td></td>
</tr>
<tr>
<td>5.112</td>
<td>disjoint_sboxes</td>
<td></td>
</tr>
<tr>
<td>5.113</td>
<td>disjoint_tasks</td>
<td></td>
</tr>
<tr>
<td>5.114</td>
<td>disjunctive</td>
<td></td>
</tr>
<tr>
<td>5.115</td>
<td>disjunctive_or_same_end</td>
<td></td>
</tr>
<tr>
<td>5.116</td>
<td>disjunctive_or_same_start</td>
<td></td>
</tr>
<tr>
<td>5.117</td>
<td>distance</td>
<td></td>
</tr>
<tr>
<td>5.118</td>
<td>distance_between</td>
<td></td>
</tr>
<tr>
<td>5.119</td>
<td>distance_change</td>
<td></td>
</tr>
<tr>
<td>5.120</td>
<td>divisible</td>
<td></td>
</tr>
<tr>
<td>5.121</td>
<td>divisible_or</td>
<td></td>
</tr>
<tr>
<td>5.122</td>
<td>dom_reachability</td>
<td></td>
</tr>
<tr>
<td>5.123</td>
<td>domain</td>
<td></td>
</tr>
<tr>
<td>5.124</td>
<td>domain_constraint</td>
<td></td>
</tr>
<tr>
<td>5.125</td>
<td>elem</td>
<td></td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.126</td>
<td>elem_from_to</td>
<td>954</td>
</tr>
<tr>
<td>5.127</td>
<td>element</td>
<td>958</td>
</tr>
<tr>
<td>5.128</td>
<td>element_greatereq</td>
<td>962</td>
</tr>
<tr>
<td>5.129</td>
<td>element_lesseq</td>
<td>966</td>
</tr>
<tr>
<td>5.130</td>
<td>element_matrix</td>
<td>970</td>
</tr>
<tr>
<td>5.131</td>
<td>element_product</td>
<td>974</td>
</tr>
<tr>
<td>5.132</td>
<td>element_sparse</td>
<td>978</td>
</tr>
<tr>
<td>5.133</td>
<td>elementnn</td>
<td>982</td>
</tr>
<tr>
<td>5.134</td>
<td>elements</td>
<td>986</td>
</tr>
<tr>
<td>5.135</td>
<td>elements_alldifferent</td>
<td>990</td>
</tr>
<tr>
<td>5.136</td>
<td>elements_sparse</td>
<td>996</td>
</tr>
<tr>
<td>5.137</td>
<td>eq</td>
<td>1000</td>
</tr>
<tr>
<td>5.138</td>
<td>eq_cst</td>
<td>1002</td>
</tr>
<tr>
<td>5.139</td>
<td>eq_set</td>
<td>1004</td>
</tr>
<tr>
<td>5.140</td>
<td>equal_sboxes</td>
<td>1006</td>
</tr>
<tr>
<td>5.141</td>
<td>equivalent</td>
<td>1010</td>
</tr>
<tr>
<td>5.142</td>
<td>exactly</td>
<td>1012</td>
</tr>
<tr>
<td>5.143</td>
<td>gcd</td>
<td>1016</td>
</tr>
<tr>
<td>5.144</td>
<td>geost</td>
<td>1018</td>
</tr>
<tr>
<td>5.145</td>
<td>geost_time</td>
<td>1024</td>
</tr>
<tr>
<td>5.146</td>
<td>geq</td>
<td>1030</td>
</tr>
<tr>
<td>5.147</td>
<td>geq_cst</td>
<td>1032</td>
</tr>
<tr>
<td>5.148</td>
<td>global_cardinality</td>
<td>1034</td>
</tr>
<tr>
<td>5.149</td>
<td>global_cardinality_low_up</td>
<td>1040</td>
</tr>
<tr>
<td>5.150</td>
<td>global_cardinality_low_up_no_loop</td>
<td>1044</td>
</tr>
<tr>
<td>5.151</td>
<td>global_cardinality_no_loop</td>
<td>1048</td>
</tr>
<tr>
<td>5.152</td>
<td>global_cardinality_with_costs</td>
<td>1052</td>
</tr>
<tr>
<td>5.153</td>
<td>global_contiguity</td>
<td>1058</td>
</tr>
<tr>
<td>5.154</td>
<td>golomb</td>
<td>1062</td>
</tr>
<tr>
<td>5.155</td>
<td>graph_crossing</td>
<td>1066</td>
</tr>
<tr>
<td>5.156</td>
<td>graph_isomorphism</td>
<td>1072</td>
</tr>
<tr>
<td>5.157</td>
<td>group</td>
<td>1076</td>
</tr>
<tr>
<td>5.158</td>
<td>group_skip_isolated_item</td>
<td>1088</td>
</tr>
<tr>
<td>5.159</td>
<td>gt</td>
<td>1098</td>
</tr>
<tr>
<td>5.160</td>
<td>highest_peak</td>
<td>1100</td>
</tr>
<tr>
<td>5.161</td>
<td>imply</td>
<td>1104</td>
</tr>
<tr>
<td>5.162</td>
<td>in</td>
<td>1106</td>
</tr>
<tr>
<td>5.163</td>
<td>in_interval</td>
<td>1110</td>
</tr>
<tr>
<td>5.164</td>
<td>in_interval_reified</td>
<td>1114</td>
</tr>
<tr>
<td>5.165</td>
<td>in_intervals</td>
<td>1118</td>
</tr>
<tr>
<td>5.166</td>
<td>in_relation</td>
<td>1120</td>
</tr>
<tr>
<td>5.167</td>
<td>in_same_partition</td>
<td>1124</td>
</tr>
<tr>
<td>5.168</td>
<td>in_set</td>
<td>1128</td>
</tr>
<tr>
<td>5.169</td>
<td>incomparable</td>
<td>1130</td>
</tr>
<tr>
<td>5.170</td>
<td>increasing</td>
<td>1132</td>
</tr>
<tr>
<td>5.171</td>
<td>increasing_global_cardinality</td>
<td>1136</td>
</tr>
</tbody>
</table>
5.172 increasing_nvalue ............................ 1142
5.173 increasing_nvalue_chain ........................ 1148
5.174 increasing_sum ............................. 1154
5.175 indexed_sum ............................... 1156
5.176 inflexion ................................. 1160
5.177 inside_sboxes .............................. 1164
5.178 int_value_precede ........................... 1168
5.179 int_value_precede_chain ...................... 1172
5.180 interval_and_count ........................ 1178
5.181 interval_and_sum ........................... 1184
5.182 inverse .................................. 1188
5.183 inverse_offset ............................. 1194
5.184 inverse_set ............................... 1198
5.185 inverse_within_range ....................... 1202
5.186 ith_pos_different_from_0 ..................... 1206
5.187 k_alldifferent ............................. 1208
5.188 k_cut ..................................... 1216
5.189 k_disjoint ................................ 1218
5.190 k_same .................................. 1222
5.191 k_same_interval ............................ 1226
5.192 k_same_modulo ............................. 1230
5.193 k_same_partition ........................... 1234
5.194 k_used_by ................................. 1238
5.195 k_used_by_interval ......................... 1242
5.196 k_used_by_modulo .......................... 1246
5.197 k_used_by_partition ....................... 1250
5.198 length_first_sequence ...................... 1254
5.199 length_last_sequence ...................... 1258
5.200 leq ...................................... 1262
5.201 leq_cst .................................. 1264
5.202 lex2 ...................................... 1266
5.203 lex_alldifferent ............................ 1268
5.204 lex_between ............................... 1272
5.205 lex_chain_less ............................. 1276
5.206 lex_chain_leq ............................... 1280
5.207 lex_different .............................. 1284
5.208 lex_equal ................................. 1288
5.209 lex_greater ............................... 1292
5.210 lex_greatereq .............................. 1298
5.211 lex_less .................................. 1304
5.212 lex_leq  .................................. 1310
5.213 lex_leq_allperm ............................ 1316
5.214 link_set_to_booleans ........................ 1318
5.215 longest_change ............................ 1322
5.216 lt ......................................... 1326
5.217 map ........................................ 1328
<table>
<thead>
<tr>
<th>Section</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.218</td>
<td>max_index</td>
<td>1332</td>
</tr>
<tr>
<td>5.219</td>
<td>max_n</td>
<td>1334</td>
</tr>
<tr>
<td>5.220</td>
<td>max_value</td>
<td>1338</td>
</tr>
<tr>
<td>5.221</td>
<td>max_size_set_of_consecutive_var</td>
<td>1344</td>
</tr>
<tr>
<td>5.222</td>
<td>maximum</td>
<td>1348</td>
</tr>
<tr>
<td>5.223</td>
<td>maximum_modulo</td>
<td>1352</td>
</tr>
<tr>
<td>5.224</td>
<td>meet_sboxes</td>
<td>1354</td>
</tr>
<tr>
<td>5.225</td>
<td>min_index</td>
<td>1360</td>
</tr>
<tr>
<td>5.226</td>
<td>min_n</td>
<td>1364</td>
</tr>
<tr>
<td>5.227</td>
<td>min_value</td>
<td>1368</td>
</tr>
<tr>
<td>5.228</td>
<td>min_size_set_of_consecutive_var</td>
<td>1374</td>
</tr>
<tr>
<td>5.229</td>
<td>minimum</td>
<td>1378</td>
</tr>
<tr>
<td>5.230</td>
<td>minimum_except_0</td>
<td>1382</td>
</tr>
<tr>
<td>5.231</td>
<td>minimum_greater_than</td>
<td>1386</td>
</tr>
<tr>
<td>5.232</td>
<td>minimum_modulo</td>
<td>1392</td>
</tr>
<tr>
<td>5.233</td>
<td>minimum_weight_alldifferent</td>
<td>1394</td>
</tr>
<tr>
<td>5.234</td>
<td>multi_global_contiguity</td>
<td>1398</td>
</tr>
<tr>
<td>5.235</td>
<td>multi_inter_distance</td>
<td>1400</td>
</tr>
<tr>
<td>5.236</td>
<td>nand</td>
<td>1402</td>
</tr>
<tr>
<td>5.237</td>
<td>nclass</td>
<td>1406</td>
</tr>
<tr>
<td>5.238</td>
<td>neq</td>
<td>1410</td>
</tr>
<tr>
<td>5.239</td>
<td>neq_cst</td>
<td>1412</td>
</tr>
<tr>
<td>5.240</td>
<td>nequivalence</td>
<td>1414</td>
</tr>
<tr>
<td>5.241</td>
<td>next_element</td>
<td>1418</td>
</tr>
<tr>
<td>5.242</td>
<td>next_greater_element</td>
<td>1424</td>
</tr>
<tr>
<td>5.243</td>
<td>ninterval</td>
<td>1428</td>
</tr>
<tr>
<td>5.244</td>
<td>no_peak</td>
<td>1432</td>
</tr>
<tr>
<td>5.245</td>
<td>no_valley</td>
<td>1436</td>
</tr>
<tr>
<td>5.246</td>
<td>non_overlap_sboxes</td>
<td>1440</td>
</tr>
<tr>
<td>5.247</td>
<td>nor</td>
<td>1446</td>
</tr>
<tr>
<td>5.248</td>
<td>not_all_equal</td>
<td>1450</td>
</tr>
<tr>
<td>5.249</td>
<td>not_in</td>
<td>1454</td>
</tr>
<tr>
<td>5.250</td>
<td>npair</td>
<td>1458</td>
</tr>
<tr>
<td>5.251</td>
<td>nset_of_consecutive_values</td>
<td>1462</td>
</tr>
<tr>
<td>5.252</td>
<td>nvalue</td>
<td>1466</td>
</tr>
<tr>
<td>5.253</td>
<td>nvalue_on_intersection</td>
<td>1472</td>
</tr>
<tr>
<td>5.254</td>
<td>nvalues</td>
<td>1476</td>
</tr>
<tr>
<td>5.255</td>
<td>nvalues_except_0</td>
<td>1480</td>
</tr>
<tr>
<td>5.256</td>
<td>nvector</td>
<td>1484</td>
</tr>
<tr>
<td>5.257</td>
<td>nvectors</td>
<td>1490</td>
</tr>
<tr>
<td>5.258</td>
<td>nvisible_from_end</td>
<td>1494</td>
</tr>
<tr>
<td>5.259</td>
<td>nvisible_from_start</td>
<td>1496</td>
</tr>
<tr>
<td>5.260</td>
<td>open_alldifferent</td>
<td>1498</td>
</tr>
<tr>
<td>5.261</td>
<td>open_among</td>
<td>1502</td>
</tr>
<tr>
<td>5.262</td>
<td>open_atleast</td>
<td>1506</td>
</tr>
<tr>
<td>5.263</td>
<td>open_atmost</td>
<td>1508</td>
</tr>
</tbody>
</table>
5.264 open_global_cardinality .............................................. 1510
5.265 open_global_cardinality_low_up ....................................... 1514
5.266 open_maximum .......................................................... 1518
5.267 open_minimum ............................................................ 1520
5.268 opposite_sign ........................................................... 1522
5.269 or ........................................................................... 1524
5.270 orchard ....................................................................... 1528
5.271 ordered_atleast_nvector .................................................. 1532
5.272 ordered_atmost_nvector .................................................. 1536
5.273 ordered_global_cardinality ............................................... 1540
5.274 ordered_nvector ............................................................ 1544
5.275 orth_link_ori_siz_end ....................................................... 1548
5.276 orth_on_the_ground ......................................................... 1552
5.277 orth_on_top_of_orth ........................................................ 1554
5.278 orths_are_connected ......................................................... 1558
5.279 overlap_sboxes .............................................................. 1562
5.280 path .......................................................................... 1566
5.281 path_from_to ................................................................. 1570
5.282 pattern ...................................................................... 1574
5.283 peak .......................................................................... 1578
5.284 period ....................................................................... 1582
5.285 period_except_0 ............................................................. 1584
5.286 period_vectors ............................................................... 1586
5.287 permutation ................................................................. 1588
5.288 place_in_pyramid ............................................................ 1590
5.289 polyomino ................................................................. 1594
5.290 power ...................................................................... 1598
5.291 precedence ................................................................. 1600
5.292 product_ctr ................................................................. 1602
5.293 proper_forest ............................................................... 1604
5.294 range_ctr .................................................................. 1608
5.295 relaxed_sliding_sum ....................................................... 1612
5.296 remainder .................................................................. 1616
5.297 roots ........................................................................ 1618
5.298 same .......................................................................... 1622
5.299 same_and_global_cardinality ......................................... 1630
5.300 same_and_global_cardinality_low_up ................................. 1634
5.301 same_intersection ........................................................ 1640
5.302 same_interval .............................................................. 1644
5.303 same_modulo .............................................................. 1648
5.304 same_partition ............................................................ 1652
5.305 same_sign ................................................................. 1656
5.306 scalar_product .......................................................... 1658
5.307 sequence_folding .......................................................... 1660
5.308 set_value_precede ......................................................... 1666
5.309 shift ................................................................. 1668
5.310 sign_of .................................................. 1672
5.311 size_max_seq_alldifferent ............................. 1674
5.312 size_max_starting_seq_alldifferent .................. 1678
5.313 sliding_card_skip0 ....................................... 1682
5.314 sliding_distribution ................................. 1686
5.315 sliding_sum ............................................ 1690
5.316 sliding_time_window .................................... 1694
5.317 sliding_time_window_from_start ...................... 1698
5.318 sliding_time_window_sum ............................... 1702
5.319 smooth .................................................. 1708
5.320 soft_all_equal_max_var ................................. 1714
5.321 soft_all_equal_min_ctr ................................. 1716
5.322 soft_all_equal_min_var ................................. 1720
5.323 soft_alldifferent_ctr ................................. 1726
5.324 soft_alldifferent_var .................................. 1730
5.325 soft_cumulative ....................................... 1734
5.326 soft_same_interval_var ................................. 1738
5.327 soft_same_modulo_var .................................. 1742
5.328 soft_same_partition_var .............................. 1746
5.329 soft_same_var .......................................... 1750
5.330 soft_used_by_interval_var ............................ 1754
5.331 soft_used_by_modulo_var .............................. 1758
5.332 soft_used_by_partition_var ............................ 1762
5.333 soft_used_by_var ....................................... 1766
5.334 some_equal ............................................. 1770
5.335 sort ..................................................... 1772
5.336 sort_permutation ...................................... 1778
5.337 stable_compatibility .................................... 1784
5.338 stage_element ......................................... 1792
5.339 stretch_circuit ........................................ 1798
5.340 stretch_path ............................................ 1802
5.341 stretch_path_partition ................................... 1810
5.342 strict_lex2 ............................................. 1814
5.343 strictly_decreasing ..................................... 1816
5.344 strictly_increasing ..................................... 1820
5.345 strongly_connected ..................................... 1824
5.346 subgraph_isomorphism .................................. 1826
5.347 sum ..................................................... 1830
5.348 sum_ctr ................................................ 1834
5.349 sum_cubes_ctr ....................................... 1838
5.350 sum_free ............................................... 1840
5.351 sum_of_increments .................................... 1842
5.352 sum_of_weights_of_distinct_values ................... 1844
5.353 sum_set ................................................. 1848
5.354 sum_squares_ctr ...................................... 1850
5.355 symmetric .............................................. 1852
<p>| B.17 | among | 2013 |
| B.18 | among_diff_0 | 2016 |
| B.19 | among_interval | 2019 |
| B.20 | among_low_up | 2022 |
| B.21 | among_modulo | 2026 |
| B.22 | among_seq | 2029 |
| B.23 | among_var | 2032 |
| B.24 | and | 2035 |
| B.25 | arith | 2037 |
| B.26 | arith_or | 2040 |
| B.27 | arith_sliding | 2045 |
| B.28 | assign_and_counts | 2050 |
| B.29 | assign_and_nvalues | 2053 |
| B.30 | atleast | 2056 |
| B.31 | atleast_nvalue | 2059 |
| B.32 | atleast_nvector | 2061 |
| B.33 | atmost | 2063 |
| B.34 | atmost1 | 2065 |
| B.35 | atmost_nvalue | 2066 |
| B.36 | atmost_nvector | 2068 |
| B.37 | balance | 2070 |
| B.38 | balance_cycle | 2072 |
| B.39 | balance_interval | 2074 |
| B.40 | balance_modulo | 2076 |
| B.41 | balance_partition | 2078 |
| B.42 | balance_path | 2080 |
| B.43 | balance_tree | 2083 |
| B.44 | between_min_max | 2086 |
| B.45 | bin_packing | 2089 |
| B.46 | bin_packing_capa | 2091 |
| B.47 | binary_tree | 2093 |
| B.48 | bipartite | 2095 |
| B.49 | calendar | 2096 |
| B.50 | cardinality_atleast | 2100 |
| B.51 | cardinality_atmost | 2103 |
| B.52 | cardinality_atmost_partition | 2106 |
| B.53 | change | 2108 |
| B.54 | change_continuity | 2111 |
| B.55 | change_pair | 2119 |
| B.56 | change_partition | 2128 |
| B.57 | change_vectors | 2130 |
| B.58 | circuit | 2133 |
| B.59 | circuit_cluster | 2135 |
| B.60 | circular_change | 2137 |
| B.61 | clause_and | 2140 |
| B.62 | clause_or | 2142 |</p>
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.63</td>
<td>clique</td>
<td>2144</td>
</tr>
<tr>
<td>B.64</td>
<td>colored_matrix</td>
<td>2146</td>
</tr>
<tr>
<td>B.65</td>
<td>coloured_cumulative</td>
<td>2149</td>
</tr>
<tr>
<td>B.66</td>
<td>coloured_cumulatives</td>
<td>2154</td>
</tr>
<tr>
<td>B.67</td>
<td>common</td>
<td>2160</td>
</tr>
<tr>
<td>B.68</td>
<td>common_interval</td>
<td>2162</td>
</tr>
<tr>
<td>B.69</td>
<td>common_modulo</td>
<td>2165</td>
</tr>
<tr>
<td>B.70</td>
<td>common_partition</td>
<td>2167</td>
</tr>
<tr>
<td>B.71</td>
<td>compare_and_count</td>
<td>2170</td>
</tr>
<tr>
<td>B.72</td>
<td>cond_lex_cost</td>
<td>2173</td>
</tr>
<tr>
<td>B.73</td>
<td>cond_lex_greater</td>
<td>2175</td>
</tr>
<tr>
<td>B.74</td>
<td>cond_lex_greatereq</td>
<td>2177</td>
</tr>
<tr>
<td>B.75</td>
<td>cond_lex_less</td>
<td>2179</td>
</tr>
<tr>
<td>B.76</td>
<td>cond_lex_lesetq</td>
<td>2181</td>
</tr>
<tr>
<td>B.77</td>
<td>connect_points</td>
<td>2183</td>
</tr>
<tr>
<td>B.78</td>
<td>connected</td>
<td>2186</td>
</tr>
<tr>
<td>B.79</td>
<td>consecutive_groups_of_ones</td>
<td>2187</td>
</tr>
<tr>
<td>B.80</td>
<td>consecutive_values</td>
<td>2190</td>
</tr>
<tr>
<td>B.81</td>
<td>contains_sboxes</td>
<td>2192</td>
</tr>
<tr>
<td>B.82</td>
<td>correspondence</td>
<td>2195</td>
</tr>
<tr>
<td>B.83</td>
<td>count</td>
<td>2197</td>
</tr>
<tr>
<td>B.84</td>
<td>counts</td>
<td>2200</td>
</tr>
<tr>
<td>B.85</td>
<td>coveredby_sboxes</td>
<td>2203</td>
</tr>
<tr>
<td>B.86</td>
<td>covers_sboxes</td>
<td>2207</td>
</tr>
<tr>
<td>B.87</td>
<td>crossing</td>
<td>2211</td>
</tr>
<tr>
<td>B.88</td>
<td>cumulative</td>
<td>2213</td>
</tr>
<tr>
<td>B.89</td>
<td>cumulative_convex</td>
<td>2216</td>
</tr>
<tr>
<td>B.90</td>
<td>cumulative_product</td>
<td>2218</td>
</tr>
<tr>
<td>B.91</td>
<td>cumulative_two_d</td>
<td>2222</td>
</tr>
<tr>
<td>B.92</td>
<td>cumulative_with_level_of_priority</td>
<td>2225</td>
</tr>
<tr>
<td>B.93</td>
<td>cumulatives</td>
<td>2228</td>
</tr>
<tr>
<td>B.94</td>
<td>cutset</td>
<td>2232</td>
</tr>
<tr>
<td>B.95</td>
<td>cycle</td>
<td>2234</td>
</tr>
<tr>
<td>B.96</td>
<td>cycle_card_on_path</td>
<td>2236</td>
</tr>
<tr>
<td>B.97</td>
<td>cycle_or_accessibility</td>
<td>2238</td>
</tr>
<tr>
<td>B.98</td>
<td>cycle_resource</td>
<td>2240</td>
</tr>
<tr>
<td>B.99</td>
<td>cyclic_change</td>
<td>2243</td>
</tr>
<tr>
<td>B.100</td>
<td>cyclic_change_joker</td>
<td>2247</td>
</tr>
<tr>
<td>B.101</td>
<td>dag</td>
<td>2252</td>
</tr>
<tr>
<td>B.102</td>
<td>decreasing</td>
<td>2254</td>
</tr>
<tr>
<td>B.103</td>
<td>deepest_valley</td>
<td>2256</td>
</tr>
<tr>
<td>B.104</td>
<td>derangement</td>
<td>2258</td>
</tr>
<tr>
<td>B.105</td>
<td>differ_from_at_least_k_pos</td>
<td>2260</td>
</tr>
<tr>
<td>B.106</td>
<td>diffn</td>
<td>2263</td>
</tr>
<tr>
<td>B.107</td>
<td>diffn_column</td>
<td>2267</td>
</tr>
<tr>
<td>B.108</td>
<td>diffn_include</td>
<td>2269</td>
</tr>
<tr>
<td>B.155</td>
<td>graph_crossing</td>
<td>2389</td>
</tr>
<tr>
<td>B.156</td>
<td>graph_isomorphism</td>
<td>2391</td>
</tr>
<tr>
<td>B.157</td>
<td>group</td>
<td>2393</td>
</tr>
<tr>
<td>B.158</td>
<td>group_skip_isolated_item</td>
<td>2399</td>
</tr>
<tr>
<td>B.159</td>
<td>gt</td>
<td>2405</td>
</tr>
<tr>
<td>B.160</td>
<td>highest_peak</td>
<td>2406</td>
</tr>
<tr>
<td>B.161</td>
<td>imply</td>
<td>2408</td>
</tr>
<tr>
<td>B.162</td>
<td>in</td>
<td>2410</td>
</tr>
<tr>
<td>B.163</td>
<td>in_interval</td>
<td>2412</td>
</tr>
<tr>
<td>B.164</td>
<td>in_interval_reified</td>
<td>2414</td>
</tr>
<tr>
<td>B.165</td>
<td>in_intervals</td>
<td>2415</td>
</tr>
<tr>
<td>B.166</td>
<td>in_relation</td>
<td>2417</td>
</tr>
<tr>
<td>B.167</td>
<td>in_same_partition</td>
<td>2419</td>
</tr>
<tr>
<td>B.168</td>
<td>in_set</td>
<td>2422</td>
</tr>
<tr>
<td>B.169</td>
<td>incomparable</td>
<td>2423</td>
</tr>
<tr>
<td>B.170</td>
<td>increasing</td>
<td>2425</td>
</tr>
<tr>
<td>B.171</td>
<td>increasing_global_cardinality</td>
<td>2428</td>
</tr>
<tr>
<td>B.172</td>
<td>increasing_nvalue</td>
<td>2438</td>
</tr>
<tr>
<td>B.173</td>
<td>increasing_nvalue_chain</td>
<td>2446</td>
</tr>
<tr>
<td>B.174</td>
<td>increasing_sum</td>
<td>2448</td>
</tr>
<tr>
<td>B.175</td>
<td>indexed_sum</td>
<td>2449</td>
</tr>
<tr>
<td>B.176</td>
<td>inflexion</td>
<td>2452</td>
</tr>
<tr>
<td>B.177</td>
<td>inside_sboxes</td>
<td>2454</td>
</tr>
<tr>
<td>B.178</td>
<td>int_value_precede</td>
<td>2457</td>
</tr>
<tr>
<td>B.179</td>
<td>int_value_precede_chain</td>
<td>2460</td>
</tr>
<tr>
<td>B.180</td>
<td>interval_and_count</td>
<td>2467</td>
</tr>
<tr>
<td>B.181</td>
<td>interval_and_sum</td>
<td>2471</td>
</tr>
<tr>
<td>B.182</td>
<td>inverse</td>
<td>2474</td>
</tr>
<tr>
<td>B.183</td>
<td>inverse_offset</td>
<td>2476</td>
</tr>
<tr>
<td>B.184</td>
<td>inverse_set</td>
<td>2478</td>
</tr>
<tr>
<td>B.185</td>
<td>inverse_within_range</td>
<td>2480</td>
</tr>
<tr>
<td>B.186</td>
<td>ith_pos_different_from_0</td>
<td>2481</td>
</tr>
<tr>
<td>B.187</td>
<td>k_alldifferent</td>
<td>2483</td>
</tr>
<tr>
<td>B.188</td>
<td>k_cut</td>
<td>2485</td>
</tr>
<tr>
<td>B.189</td>
<td>k_disjoint</td>
<td>2487</td>
</tr>
<tr>
<td>B.190</td>
<td>k_same</td>
<td>2489</td>
</tr>
<tr>
<td>B.191</td>
<td>k_same_interval</td>
<td>2491</td>
</tr>
<tr>
<td>B.192</td>
<td>k_same_modulo</td>
<td>2493</td>
</tr>
<tr>
<td>B.193</td>
<td>k_same_partition</td>
<td>2495</td>
</tr>
<tr>
<td>B.194</td>
<td>k_used_by</td>
<td>2497</td>
</tr>
<tr>
<td>B.195</td>
<td>k_used_by_interval</td>
<td>2499</td>
</tr>
<tr>
<td>B.196</td>
<td>k_used_by_modulo</td>
<td>2501</td>
</tr>
<tr>
<td>B.197</td>
<td>k_used_by_partition</td>
<td>2503</td>
</tr>
<tr>
<td>B.198</td>
<td>length_first_sequence</td>
<td>2505</td>
</tr>
<tr>
<td>B.199</td>
<td>length_last_sequence</td>
<td>2508</td>
</tr>
<tr>
<td>B.200</td>
<td>leq</td>
<td>2511</td>
</tr>
</tbody>
</table>
B.201 leq_cst . .......................... 2512
B.202 lex2 . .......................... 2513
B.203 lex_alldifferent . .......................... 2514
B.204 lex_between . .......................... 2516
B.205 lex_chain_less . .......................... 2521
B.206 lex_chain_leseq . .......................... 2523
B.207 lex_different . .......................... 2525
B.208 lex_equal . .......................... 2528
B.209 lex_greater . .......................... 2531
B.210 lex_greatereq . .......................... 2534
B.211 lex_less . .......................... 2537
B.212 lex_less_eq . .......................... 2540
B.213 lex_less_eq_allperm . .......................... 2543
B.214 link_set_to_booleans . .......................... 2544
B.215 longest_change . .......................... 2546
B.216 lt . .......................... 2549
B.217 map . .......................... 2550
B.218 max_index . .......................... 2552
B.219 max_n . .......................... 2554
B.220 max_nvalue . .......................... 2556
B.221 max_size_set_of_consecutive_var . .......................... 2558
B.222 maximum . .......................... 2560
B.223 maximum_modulo . .......................... 2562
B.224 meet_sboxes . .......................... 2564
B.225 min_index . .......................... 2568
B.226 min_n . .......................... 2570
B.227 min_nvalue . .......................... 2572
B.228 min_size_set_of_consecutive_var . .......................... 2574
B.229 minimum . .......................... 2576
B.230 minimum_except_0 . .......................... 2578
B.231 minimum_greater_than . .......................... 2581
B.232 minimum_modulo . .......................... 2584
B.233 minimum_weight_alldifferent . .......................... 2586
B.234 multi_global_contiguity . .......................... 2588
B.235 multi_inter_distance . .......................... 2590
B.236 nand . .......................... 2592
B.237 nclass . .......................... 2594
B.238 neq . .......................... 2596
B.239 neq_cst . .......................... 2597
B.240 nequivalence . .......................... 2598
B.241 next_element . .......................... 2600
B.242 next_greater_element . .......................... 2604
B.243 ninterval . .......................... 2606
B.244 no_peak . .......................... 2608
B.245 no_valley . .......................... 2610
B.246 non_overlap_sboxes . .......................... 2612
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.247 nor</td>
<td></td>
</tr>
<tr>
<td>B.248 not_all_equal</td>
<td></td>
</tr>
<tr>
<td>B.249 not_in</td>
<td></td>
</tr>
<tr>
<td>B.250 npair</td>
<td></td>
</tr>
<tr>
<td>B.251 nset_of_consecutive_values</td>
<td></td>
</tr>
<tr>
<td>B.252 nvalue</td>
<td></td>
</tr>
<tr>
<td>B.253 nvalue_on_intersection</td>
<td></td>
</tr>
<tr>
<td>B.254 nvalues</td>
<td></td>
</tr>
<tr>
<td>B.255 nvalues_except_0</td>
<td></td>
</tr>
<tr>
<td>B.256 nvector</td>
<td></td>
</tr>
<tr>
<td>B.257 nvectors</td>
<td></td>
</tr>
<tr>
<td>B.258 nvisible_from_end</td>
<td></td>
</tr>
<tr>
<td>B.259 nvisible_from_start</td>
<td></td>
</tr>
<tr>
<td>B.260 open_alldifferent</td>
<td></td>
</tr>
<tr>
<td>B.261 open_among</td>
<td></td>
</tr>
<tr>
<td>B.262 open_atleast</td>
<td></td>
</tr>
<tr>
<td>B.263 open_atmost</td>
<td></td>
</tr>
<tr>
<td>B.264 open_global_cardinality</td>
<td></td>
</tr>
<tr>
<td>B.265 open_global_cardinality_low_up</td>
<td></td>
</tr>
<tr>
<td>B.266 open_maximum</td>
<td></td>
</tr>
<tr>
<td>B.267 open_minimum</td>
<td></td>
</tr>
<tr>
<td>B.268 opposite_sign</td>
<td></td>
</tr>
<tr>
<td>B.269 or</td>
<td></td>
</tr>
<tr>
<td>B.270 orchard</td>
<td></td>
</tr>
<tr>
<td>B.271 ordered_atleast_nvector</td>
<td></td>
</tr>
<tr>
<td>B.272 ordered_atmost_nvector</td>
<td></td>
</tr>
<tr>
<td>B.273 ordered_global_cardinality</td>
<td></td>
</tr>
<tr>
<td>B.274 ordered_nvector</td>
<td></td>
</tr>
<tr>
<td>B.275 orth_link_ori_siz_end</td>
<td></td>
</tr>
<tr>
<td>B.276 orth_on_the_ground</td>
<td></td>
</tr>
<tr>
<td>B.277 orth_on_top_of_orth</td>
<td></td>
</tr>
<tr>
<td>B.278 orths_are_connected</td>
<td></td>
</tr>
<tr>
<td>B.279 overlap_sboxes</td>
<td></td>
</tr>
<tr>
<td>B.280 path</td>
<td></td>
</tr>
<tr>
<td>B.281 path_from_to</td>
<td></td>
</tr>
<tr>
<td>B.282 pattern</td>
<td></td>
</tr>
<tr>
<td>B.283 peak</td>
<td></td>
</tr>
<tr>
<td>B.284 period</td>
<td></td>
</tr>
<tr>
<td>B.285 period_except_0</td>
<td></td>
</tr>
<tr>
<td>B.286 period_vectors</td>
<td></td>
</tr>
<tr>
<td>B.287 permutation</td>
<td></td>
</tr>
<tr>
<td>B.288 place_in_pyramid</td>
<td></td>
</tr>
<tr>
<td>B.289 polyomino</td>
<td></td>
</tr>
<tr>
<td>B.290 power</td>
<td></td>
</tr>
<tr>
<td>B.291 precedence</td>
<td></td>
</tr>
<tr>
<td>B.292 product_ctr</td>
<td></td>
</tr>
</tbody>
</table>

Notes:
- The contents list is a directory of terms with page numbers indicating their location in a document.
- The page numbers range from 2615 to 2708.
<p>| B.293  | proper_forest                      | 2710  |
| B.294  | range_ctr                          | 2712  |
| B.295  | relaxed_sliding_sum                | 2714  |
| B.296  | remainder                          | 2717  |
| B.297  | roots                              | 2718  |
| B.298  | same                               | 2719  |
| B.299  | same_and_global_cardinality        | 2721  |
| B.300  | same_and_global_cardinality_low_up | 2724  |
| B.301  | same_intersection                  | 2727  |
| B.302  | same_interval                      | 2730  |
| B.303  | same_modulo                        | 2732  |
| B.304  | same_partition                     | 2734  |
| B.305  | same_sign                          | 2737  |
| B.306  | scalar_product                     | 2738  |
| B.307  | sequence_folding                   | 2741  |
| B.308  | set_value_precede                  | 2744  |
| B.309  | shift                              | 2745  |
| B.310  | sign_of                            | 2749  |
| B.311  | size_max_seq_alldifferent          | 2750  |
| B.312  | size_max_starting_seq_alldifferent  | 2752  |
| B.313  | sliding_card_skip0                 | 2755  |
| B.314  | sliding_distribution               | 2758  |
| B.315  | sliding_sum                        | 2761  |
| B.316  | sliding_time_window                | 2764  |
| B.317  | sliding_time_window_from_start     | 2766  |
| B.318  | sliding_time_window_sum            | 2769  |
| B.319  | smooth                             | 2776  |
| B.320  | soft_all_equal_max_var             | 2779  |
| B.321  | soft_all_equal_min_ctr             | 2781  |
| B.322  | soft_all_equal_min_var             | 2783  |
| B.323  | soft_alldifferent_ctr              | 2785  |
| B.324  | soft_alldifferent_var              | 2787  |
| B.325  | soft_cumulative                    | 2789  |
| B.326  | soft_same_interval_var             | 2791  |
| B.327  | soft_same_modulo_var               | 2794  |
| B.328  | soft_same_partition_var            | 2796  |
| B.329  | soft_same_var                      | 2799  |
| B.330  | soft_used_by_interval_var          | 2801  |
| B.331  | soft_used_by_modulo_var            | 2804  |
| B.332  | soft_used_by_partition_var         | 2806  |
| B.333  | soft_used_by_var                   | 2809  |
| B.334  | some_equal                         | 2812  |
| B.335  | sort                               | 2814  |
| B.336  | sort_permutation                   | 2816  |
| B.337  | stable_compatibility               | 2818  |
| B.338  | stage_element                      | 2820  |</p>
<table>
<thead>
<tr>
<th>B.339</th>
<th>stretch_circuit</th>
<th>2823</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.340</td>
<td>stretch_path</td>
<td>2826</td>
</tr>
<tr>
<td>B.341</td>
<td>stretch_path_partition</td>
<td>2830</td>
</tr>
<tr>
<td>B.342</td>
<td>strict_lex2</td>
<td>2836</td>
</tr>
<tr>
<td>B.343</td>
<td>strictly_decreasing</td>
<td>2837</td>
</tr>
<tr>
<td>B.344</td>
<td>strictly_increasing</td>
<td>2840</td>
</tr>
<tr>
<td>B.345</td>
<td>strongly_connected</td>
<td>2842</td>
</tr>
<tr>
<td>B.346</td>
<td>subgraph_isomorphism</td>
<td>2843</td>
</tr>
<tr>
<td>B.347</td>
<td>sum</td>
<td>2845</td>
</tr>
<tr>
<td>B.348</td>
<td>sum_ctr</td>
<td>2847</td>
</tr>
<tr>
<td>B.349</td>
<td>sum_cubes_ctr</td>
<td>2849</td>
</tr>
<tr>
<td>B.350</td>
<td>sum_free</td>
<td>2851</td>
</tr>
<tr>
<td>B.351</td>
<td>sum_of_increments</td>
<td>2852</td>
</tr>
<tr>
<td>B.352</td>
<td>sum_of_weights_of_distinct_values</td>
<td>2854</td>
</tr>
<tr>
<td>B.353</td>
<td>sum_set</td>
<td>2857</td>
</tr>
<tr>
<td>B.354</td>
<td>sum_squares_ctr</td>
<td>2859</td>
</tr>
<tr>
<td>B.355</td>
<td>symmetric</td>
<td>2861</td>
</tr>
<tr>
<td>B.356</td>
<td>symmetric_alldifferent</td>
<td>2862</td>
</tr>
<tr>
<td>B.357</td>
<td>symmetric_alldifferent_except_0</td>
<td>2864</td>
</tr>
<tr>
<td>B.358</td>
<td>symmetric_cardinality</td>
<td>2866</td>
</tr>
<tr>
<td>B.359</td>
<td>symmetric_gcc</td>
<td>2868</td>
</tr>
<tr>
<td>B.360</td>
<td>temporal_path</td>
<td>2870</td>
</tr>
<tr>
<td>B.361</td>
<td>tour</td>
<td>2873</td>
</tr>
<tr>
<td>B.362</td>
<td>track</td>
<td>2875</td>
</tr>
<tr>
<td>B.363</td>
<td>tree</td>
<td>2879</td>
</tr>
<tr>
<td>B.364</td>
<td>tree_range</td>
<td>2881</td>
</tr>
<tr>
<td>B.365</td>
<td>tree_resource</td>
<td>2885</td>
</tr>
<tr>
<td>B.366</td>
<td>twin</td>
<td>2889</td>
</tr>
<tr>
<td>B.367</td>
<td>two_layer_edge_crossing</td>
<td>2891</td>
</tr>
<tr>
<td>B.368</td>
<td>two_orth_are_in_contact</td>
<td>2894</td>
</tr>
<tr>
<td>B.369</td>
<td>two_orth_column</td>
<td>2897</td>
</tr>
<tr>
<td>B.370</td>
<td>two_orth_do_not_overlap</td>
<td>2899</td>
</tr>
<tr>
<td>B.371</td>
<td>two_orth_include</td>
<td>2902</td>
</tr>
<tr>
<td>B.372</td>
<td>used_by</td>
<td>2904</td>
</tr>
<tr>
<td>B.373</td>
<td>used_by_interval</td>
<td>2906</td>
</tr>
<tr>
<td>B.374</td>
<td>used_by_modulo</td>
<td>2908</td>
</tr>
<tr>
<td>B.375</td>
<td>used_by_partition</td>
<td>2910</td>
</tr>
<tr>
<td>B.376</td>
<td>uses</td>
<td>2913</td>
</tr>
<tr>
<td>B.377</td>
<td>valley</td>
<td>2915</td>
</tr>
<tr>
<td>B.378</td>
<td>vec_eq_tuple</td>
<td>2917</td>
</tr>
<tr>
<td>B.379</td>
<td>visible</td>
<td>2919</td>
</tr>
<tr>
<td>B.380</td>
<td>weighted_partial_alldiff</td>
<td>2924</td>
</tr>
<tr>
<td>B.381</td>
<td>xor</td>
<td>2928</td>
</tr>
<tr>
<td>B.382</td>
<td>Utilities</td>
<td>2930</td>
</tr>
</tbody>
</table>
Preface

This catalogue presents a list of global constraints. Within this catalogue the term "global constraint" should be understood as an expressive and concise condition involving a non-fixed number of variables. This informal definition does not make any assumption neither about the potential use of a global constraint nor about the techniques\(^1\) associated with the development of global constraints. It contains about 381 constraints, which are explicitly described in terms of graph properties and/or automata and/or first order logic formulae and/or conjunction of other constraints.

This *Global Constraint Catalogue* is an expanded version of the list of global constraints presented in \[25\] and an updated version of \[37\]. The principle used for describing global constraints has been slightly modified in order to deal with a larger number of global constraints. Since 2003, we try to provide an automaton that recognises the solutions associated with a global constraint. Since 2009, we also try to provide a first order logic formula for defining the solutions accepted by a geometrical constraint.

Writing a dictionary is a long process, especially in a field where new words are defined every year. In this context, one difficulty is to express explicitly the meaning of global constraints in terms of meta-data. Finding an appropriate and concise description that easily captures the meaning of most global constraints seems to be a tricky task.

One may wonder how so many constraints can be used at all in practice? However many fields produce a number of articles containing partial and specific results. Within the area of global constraints, we fill that trying extracting and classifying such knowledge, as well as providing meta-data for encoding it, may be a help, both for humans and machines, to exploit systematically ongoing research results and to put these results in perspective.

**Goal of the catalogue.** This catalogue has four main goals. First, it provides an overview of most of the different global constraints that were gradually introduced in the area of constraint programming since the work of J.-L. Laurière on ALICE \[238\]. It also identifies new global constraints for which no existing published work exists. The global constraints are arranged in alphabetic order, and for all of them a description and an example are systematically provided. When available, it also presents some typical usage as well as some pointers to existing filtering algorithms.

\(^1\) As quoted by J. N. Hooker in \[197\], "identifying a field with its techniques is an intellectually as well as practically unsatisfying" and has a lot of drawbacks.
Second, the global constraints described in this catalogue are not only accessible to humans, who can read the catalogue for searching for some information. It is also available to machines, which can read and interpret it. This is why there exists an electronic version of this catalogue where one can get, for most global constraints, a complete description in terms of meta-data. In fact, most of this catalogue and its figures were automatically generated from this electronic version by a computer program. This description is based on three complementary ways to look at a global constraint. The first one defines a global constraint as searching for a graph with specific properties \[24\], the second one characterises a global constraint in terms of an automaton that only recognises the solutions associated with that global constraint \[34, 286\]², while the third one defines in the context of geometric constraints a global constraint as a restricted first order logic formula \[93\]. The key point of these descriptions is their ability to define explicitly in a concise way the meaning of most global constraints. In addition these descriptions can also be systematically turned into polynomial filtering algorithms.

Third, we hope that this unified description of apparently diverse global constraints will allow for establishing a systematic link between the properties of basic concepts used for describing global constraints and the properties of the global constraints as a whole.

Finally, we also hope that it will attract more people from the algorithmic community into the area of constraints. To a certain extent this has already started at places like CWI in Amsterdam, the Max-Planck für Informatik (Saarbrücken) or the university of Waterloo. We also hope that it will attract people from combinatorics in order to produce theories and knowledge that could nicely unify and/or put in perspective different aspects of constraints (i.e., breaking symmetries, counting the number of solutions).

**Use of the catalogue.** The catalogue is organised into five chapters:

- **Chapter 1** provide a short overview of the main entries you may first consult when you are not familiar with the catalogue.

- **Chapter 2** explains how the meaning of global constraints is described in terms of graph-properties or in terms of automata. On the one hand, if one wants to consult the catalogue for getting the informal definition of a global constraint, examples of use of that constraint or pointers to filtering algorithms, then one only needs to read the first section of Chapter 2: describing the arguments of a global constraint, page 6. On the other hand, if one wants to understand those entries describing explicitly the meaning of a constraint then all the material of Chapter 2 is required.

- **Chapter 3** describes the content of the catalogue as well as different ways for searching through the catalogue. This material is essential.

²Automata were first use in the 90ies by N. R. Vempaty [406] and J. Amilhastre [6] in the context of constraint networks. Later on in 2007, they were also used by M.-C. Coté et al. [118] in the context of linear programming.
• Chapter 4 covers additional topics, such as the differences from the 2000 report [25] on global constraints, the generation of implied constraints that are systematically linked to the graph-based description of a global constraint, and the electronic version of the catalogue. The material describing the format of the entries of a global constraint is mandatory for those who want to exploit the electronic version in order to write pre-processors for performing various tasks for a global constraint.

• Finally, Chapter 5 corresponds to the catalogue itself, which gives the global constraints in alphabetical order.

Acknowledgments. Nicolas Beldiceanu was the principal investigator and main architect of the constraint catalogue, provided the main ideas, and wrote a checker for the constraint descriptions, a figure generation program for the constraint descriptions and an evaluator for most constraints. Jean-Xavier Rampon provided the proofs for the graph invariants. Mats Carlsson contributed to the design of the meta-data format, generated some of the automata together with their negated form, provide some constraints evaluators, and wrote the program that created the \LaTeX{} version of this catalogue from the constraint descriptions.

The idea of describing explicitly the meaning of global constraints in a declarative way has been inspired by the work on meta-knowledge of Jacques Pitrat [296, 297, 298].


Furthermore, we are grateful to Irit Katriel who contributed by updating the description of some filtering algorithms related to flow and matching of the catalogue, to Luis Quesada and Stéphane Zampelli who provide inputs for the \texttt{dom\_reachability}, the \texttt{subgraph\_isomorphism} and \texttt{graph\_isomorphism} constraints, and to Radoslaw Szymanek and Guido Tack for providing the correspondence of global constraints of the catalogue with the constraints of \texttt{JaCoP} and \texttt{Gecode}. We are also es-
especially grateful to Sophie Demasse both, for creating the on-line version of the cata-
logue (http://www.emn.fr/x-info/sdemasse/gccat/) and for writing down the entry
related to the cumulative longest hole problems, to Helmut Simonis both, for designing
the XML schema (see Section 4.4.2) for the global constraints and their arguments, for
providing the corresponding generation programs and for providing data for several
rectangle placement problems, and to Pierre Flener and Justin Pearson for providing
feedback with respect to the Symmetry slot of global constraints.

The geometric constraints geost and visible as well as the constraints related
to the Region Connection Calculus where integrated within the catalogue while working
on the European Union Sixth Framework Programme Contract FP6-034691 “Net-
WMS”.

Finally, we want to acknowledge the continuing support of SICS and EMN for
providing excellent working conditions over the years. The part of this work related to
graph properties in Chapter 5 was done while the corresponding author was working at
SICS.

Readers may send their suggestion via email to the corresponding author with
catalogue as subject.

Uppsala, Sweden, August 2003 – Nantes, France, February 2012 — NB, MC, JXR
Chapter 1

Getting started

If you are using the pdf version of the catalogue use Adobe Reader if you want to be sure to see PDF annotations. If you do not see on your screen a small yellow bullet at the beginning of this paragraph, you are using a PDF viewer that does not fully support PDF annotations. Within keywords and constraints, the icons

💡 indicates a point of interest (e.g., a necessary condition, a typical use),

⚠️ denotes a typical error or a common misunderstanding.

The main entries you may consult if you want to have a first look to the catalogue are:

- To get an idea of how global constraint arguments are described look at Section 2.1.

- To search in the catalogue look at Section 3.3.

- To search a constraint from a keyword look at Section 3.7.

- To get an idea how keywords are structured look at Section 3.6.

- To know available semantic links between constraints look at Section 2.5.

- To get through the core global constraints look at the keyword core.

- To see how constraints symmetries are described look at Section 2.1.5.

- To get an idea of general filtering techniques look at the meta-keyword filtering and more specifically to the entries Berge-acyclic constraint network, constructive disjunction, flow and sweep. To get the notion of consistency achieved by a filtering algorithm look at the keywords arc-consistency and bound-consistency.

---

1Since we are using the LaTeX package pdfcomment and since most PDF viewers do not support PDF annotations.
• To get an idea of modelling techniques and of modelling exercises look at the meta-keywords modelling and modelling exercises.

• To get and idea of reformulations of global constraints look at Section 2.4.

• To get an idea of general ways to explicitly represent the meaning of global constraints look at (a) Section 2.2 for the graph property based description, (b) Section 2.3 for the automaton based description, (c) the reference [93]) for the logical based description (e.g., see the Logic slot of meet_sboxes).

• To get an idea of the meta-data used for describing a constraint look at Section 4.4.1 for the facts and Section 4.4.2 for the XML schema.

• To get the correspondence of global constraints of the catalogue with concrete constraint systems or modelling languages, such as Choco, Gecode, JaCoP, MiniZinc, or SICStus look at Appendix C.
# Chapter 2

## Describing Global Constraints

### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Describing the arguments of a global constraint</td>
<td>6</td>
</tr>
<tr>
<td>2.1.1</td>
<td>Basic data types</td>
<td>6</td>
</tr>
<tr>
<td>2.1.2</td>
<td>Compound data types</td>
<td>8</td>
</tr>
<tr>
<td>2.1.3</td>
<td>Restrictions</td>
<td>9</td>
</tr>
<tr>
<td>2.1.4</td>
<td>Declaring a global constraint</td>
<td>17</td>
</tr>
<tr>
<td>2.1.5</td>
<td>Describing symmetries between arguments</td>
<td>18</td>
</tr>
<tr>
<td>2.2</td>
<td>Describing global constraints in terms of graph properties</td>
<td>39</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Basic ideas and illustrative example</td>
<td>39</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Ingredients used for describing global constraints</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>Collection generators</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>Elementary constraints attached to the arcs</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>Simple arithmetic expressions</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>Arithmetic expressions</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>Arc constraints</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Graph generators</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>Graph properties</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>Graph terminology and notations</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>Graph parameters</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>Graph class</td>
<td>69</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Graph constraint</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>Simple graph constraint</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>Dynamic graph constraint</td>
<td>74</td>
</tr>
<tr>
<td>2.3</td>
<td>Describing global constraints in terms of automata</td>
<td>78</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Selecting an appropriate description</td>
<td>78</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Defining an automaton</td>
<td>82</td>
</tr>
<tr>
<td>2.4</td>
<td>Reformulating global constraints as a conjunction</td>
<td>83</td>
</tr>
<tr>
<td>2.5</td>
<td>Semantic links between global constraints</td>
<td>84</td>
</tr>
</tbody>
</table>
2.5.1 Assignment dimension added ........................................ 84
2.5.2 Assignment dimension removed .................................... 84
2.5.3 Attached to cost variant ............................................. 85
2.5.4 Common keyword ..................................................... 86
2.5.5 Comparison swapped ................................................ 86
2.5.6 Cost variant ............................................................ 86
2.5.7 Generalisation ......................................................... 86
2.5.8 Hard version ............................................................ 86
2.5.9 Implied by .............................................................. 87
2.5.10 Implies ................................................................. 87
2.5.11 Implies (if swap arguments) ....................................... 87
2.5.12 Implies (items to collection) ...................................... 87
2.5.13 Negation ............................................................... 88
2.5.14 Part of system of constraints ....................................... 88
2.5.15 Related ................................................................. 88
2.5.16 Related to a common problem .................................... 88
2.5.17 Root concept ......................................................... 89
2.5.18 Shift of concept ....................................................... 89
2.5.19 Soft variant ........................................................... 89
2.5.20 Specialisation ......................................................... 89
2.5.21 System of constraints ............................................... 89
2.5.22 Used in graph description ......................................... 90
2.5.23 Used in reformulation .............................................. 90
2.5.24 Uses in its reformulation .......................................... 90

We first motivate the need for an explicit description of global constraints and then present the graph-based as well as the automaton-based descriptions used throughout the catalogue. On the one hand, the graph-based representation considers a global constraint as a subgraph of an initial given graph. This subgraph has to satisfy a set of required graph properties. On the other hand, the automaton-based representation denotes a global constraint as a hypergraph constructed from a given constraint checker.\footnote{A constraint checker is a program that takes an instance of a constraint for which all variables are fixed and tests whether the constraint is satisfied or not.}

Both, the initial graph of the graph-based representation, as well as the hypergraph of the automaton-based representation have a very regular structure, which should give the opportunity for efficient filtering algorithms taking advantage of this structure.

We now present our motivations for an explicit description of the meaning of global constraints. The current trend\footnote{This can be observed in all constraint manuals where the description of the meaning is always informal.} is to first use natural language for describing the meaning of a global constraint and second to work out a specialised filtering algorithm. Since we have a huge number of potential global constraints that can be combined in a lot of ways, this is an immense task. Since we are also interested in providing other services, such as visualisation \cite{425,364,367}, explanations \cite{340}, cuts for linear programming \cite{199}, moves for local search \cite{76}, generation of clauses for SAT...
solvers [275], generation of multivalued decision diagrams that represent compact relaxations of global constraints [196], soft global constraints [294, 49, 399], learning implied global constraints [56], simplifying away fixed variables from global constraints when they have the same effect on the remaining unfixed variables in order to automatically identify equivalent subproblems during search [109], and specialised heuristics for each global constraint this is even worse. One could argue that a candidate for describing explicitly the meaning of global constraints would be second order predicate calculus. This could perhaps solve our description problem but would, at least currently, not be useful for deriving any filtering algorithm.\footnote{One could perhaps use a system like MONA [193] or some ideas from [77] for getting a constraint checker in the context of the graph-based representation.} For a similar reason Prolog was restricted to Horn clauses for which one had a reasonable solving mechanism. What we want to stress through this example is the fact that a declarative description is really useful only if it also provides some hints about how to deal with that description. Our first choice of a graph-based representation has been influenced by the following observations:

- The concept of graph has its roots in the area of mathematical recreations (see for instance L. Euler [142], H. E. Dudeney [135], E. Lucas [250] and T. P. Kirkman [220]), which was somehow the ancestor of combinatorial problems. In this perspective a graph-based description makes sense.

- In one of the first books introducing graph theory [53], C. Berge presents graph theory as a way of grouping apparently diverse problems and results. This was also the case for global constraints.

- The parameters associated with graphs are concrete and concise. Moreover a lot of results about graphs can be expressed in terms of graph invariants involving various graph parameters that are valid for specific graph classes. In essence, formulas are a kind of declarative statement that is much more compact than algorithms.

- Finally, it is well known that graph theory is an important tool [261] with respect to the development of efficient filtering algorithms [320, 322, 325, 333, 262, 215, 46, 397, 313].

Our second choice of an automaton-based representation has been motivated by the following observation. Writing a constraint checker is usually a straightforward task. The corresponding program can usually be turned into an automaton. Of course an automaton is typically used on a fixed sequence of symbols. But, in the context of filtering algorithms, we have to deal with a sequence of variables. For this purpose we have shown [34] for some automata how to decompose them into a conjunction of smaller constraints. In this context, a global constraint can be seen as a hypergraph corresponding to its decomposition.
2.1 Describing the arguments of a global constraint

Since global constraints have to receive their arguments in some form, no matter whether we use the graph-based or the automaton-based description, we start by describing the abstract data types that we use in order to specify the arguments of a global constraint. These abstract data types are not related to any specific programming language like Caml, C, C++, Java or Prolog. If one wants to focus on a specific language, then one has to map these abstract data types to the data types that are available within the considered programming language. In a second phase we describe all the restrictions that one can impose on the arguments of a global constraint. Finally, in a third phase we show how to use these ingredients in order to declare the arguments of a global constraint.

2.1.1 Basic data types

We provide the following basic data types:

- **atom** corresponds to an atom. Predefined atoms are MININT and MAXINT, which respectively correspond to the smallest and to the largest integer.

- **int** corresponds to an integer value.

- **dvar** corresponds to a domain variable. A domain variable \( V \) is a variable that will be assigned an integer value taken from an initial finite set of integer values denoted by \( \text{dom}(V) \). \( \underline{V} \) and \( \overline{V} \) respectively denote the minimum and the maximum values of \( \text{dom}(V) \).

- **fdvar** corresponds to a possibly unbounded domain variable. A possibly unbounded domain variable is a variable that will be assigned an integer value from an initial finite set of integer values denoted by \( \text{dom}(V) \) or from interval minus infinity, plus infinity. This type is required for declaring the domain of a variable. It is also required by some systems in the context of specific constraints like arithmetic or element constraints.

- **sint** corresponds to a finite set of integer values.

- **svar** corresponds to a set variable. A set variable \( V \) is a variable that will be assigned to a finite set of integer values. Its lower bound \( \text{lb}(V) \) denotes the set of integer values that for sure belong to \( V \), while its upper bound \( \text{ub}(V) \) denotes the set of integer values that may belong to \( V \). \( \text{dom}(V) = \{v_1, \ldots, v_n, v_{n+1}, \ldots, v_m\} \) is a shortcut for combining the lower and upper bounds of \( V \) in one single notation:
  
  - Bold values designate those values that only belong to \( V \).
  - Plain values indicate those values that belong to \( \overline{V} \) and not to \( \underline{V} \).

- **mint** corresponds to a multiset of integer values.

- **mvar** corresponds to a multiset variable. A multiset variable is a variable that will be assigned to a multiset of integer values.
2.1. DESCRIBING THE ARGUMENTS OF A GLOBAL CONSTRAINT

- **real** corresponds to a *real number*.
- **rvar** corresponds to a *real variable*. A *real variable* is a variable that will be assigned a *real number* taken from an initial finite set of intervals. A real number is usually represented by an interval of two floating point numbers.

Beside domain, set, multiset and float variables we have not yet introduced *graph variables* [131]. A graph variable is currently simulated by using one set variable for each vertex of the graph (see the third example of type declaration of 2.1.2).
2.1.2 Compound data types

We provide the following compound data types:

- **list**($T$) corresponds to a list of elements of type $T$, where $T$ is a basic or a compound data type.

- **collection**($A_1, A_2, \ldots, A_n$) corresponds to a collection of ordered items, where each item consists of $n > 0$ attributes $A_1, A_2, \ldots, A_n$. Each attribute is an expression of the form $a - T$, where $a$ is the name of the attribute and $T$ the type of the attribute (a basic or a compound data type). All names of the attributes of a given collection should be distinct and different from the keyword **key**, which corresponds to an implicit attribute. Its value is the position of an item within the collection. The first item of a collection is associated with position 1.

The following notations are used for instantiated arguments:

- A list of elements $e_1, e_2, \ldots, e_n$ is denoted $[e_1, e_2, \ldots, e_n]$.

- A finite set of integers $i_1, i_2, \ldots, i_n$ is denoted $\{i_1, i_2, \ldots, i_n\}$.

- A multiset of integers $i_1, i_2, \ldots, i_n$ is denoted $\{\{i_1, i_2, \ldots, i_n\}\}$.

- A collection of $n$ items, each item having $m$ attributes, is denoted by $\langle a_1 - v_{11} \ldots a_m - v_{1m}, a_1 - v_{21} \ldots a_m - v_{2m}, \ldots, a_1 - v_{n1} \ldots a_m - v_{nm} \rangle$. Each item is separated from the previous item by a comma. When the items of the collection involve one single attribute $a_1$, $\langle v_{11}, v_{21}, \ldots, v_{n1} \rangle$ can eventually be used as a shortcut for $\langle a_1 - v_{11}, a_1 - v_{21}, \ldots, a_1 - v_{n1} \rangle$.

- The $i^{th}$ item of a collection $c$ is denoted $c[i]$.

- The value of the attribute $a$ of the $i^{th}$ item of a collection $c$ is denoted $c[i].a$. Note that, within an arithmetic expression, we can use the shortcut $c[i]$ when the collection $c$ involves one single attribute.

- The number of items of a collection $c$ is denoted $|c|$.

---

4This attribute is not explicitly defined.
2.1. DESCRIBING THE ARGUMENTS OF A GLOBAL CONSTRAINT

**EXAMPLE:** Let us illustrate with four examples, the types one can create. These examples concern the creation of a collection of variables, a collection of tasks, a graph variable [131] and a collection of orthotopes.

- In the first example we define VARIABLES so that it corresponds to a collection of variables. VARIABLES is for instance used in the alldifferent constraint. The declaration VARIABLES: collection(var − dvar) defines a collection of items, each of which having one attribute var that is a domain variable.

- In the second example we define TASKS so that it corresponds to a collection of tasks, each task being defined by its origin, its duration, its end and its resource consumption. Such a collection is for instance used in the cumulative constraint. The declaration TASKS: collection(origin − dvar, duration − dvar, end − dvar, height − dvar) defines a collection of items, each of which having the four attributes origin, duration, end and height which all are domain variables.

- In the third example we define a graph as a collection of nodes NODES, each node being defined by its index (i.e., identifier) and its successors. Such a collection is for instance used in the dag constraint. The declaration NODES: collection(index − int, succ − svar) defines a collection of items, each of which having the two attributes index and succ which respectively are integers and set variables.

- In the last example we define ORTHOTOPES so that is corresponds to a collection of orthotopes. Each orthotope is described by an attribute orth. Unlike the previous examples, the type of this attribute does not correspond any more to a basic data type but rather to a collection of n items, where n is the number of dimensions of the orthotope. This collection, named ORTHOTOPE, defines for a given dimension the origin, the size and the end of the object in this dimension. This leads to the two declarations:

  - ORTHOTOPE − collection(ori − dvar, siz − dvar, end − dvar).
  - ORTHOTOPES − collection(orth − ORTHOTOPE).

ORTHOTOPES is for instance used in the diffn constraint.

---

**2.1.3 Restrictions**

When defining the arguments of a global constraint, it is often the case that one needs to express additional conditions that refine the type declarations of its arguments. For this purpose we provide restrictions that allow for specifying these additional conditions. Each restriction has a name and a set of arguments and is described by the following items:

- A small paragraph first describes the effect of the restriction,

- An example points to a constraint using the restriction,

- Finally, a ground instance, preceded by the symbol ⊲, which satisfies the restriction is given. Similarly, a ground instance, preceded by the symbol ⊳, which

\---

\[ a \] An orthotope corresponds to the generalisation of a segment, a rectangle and a box to the n-dimensional case.

\[ b \] 1 for a segment, 2 for a rectangle, 3 for a box, . . . .
violates the restriction is proposed. In this latter case, a bold font may be used for pointing to the source of the problem.

Currently the list of restrictions is:

- **in_list(Arg,ListAtoms)**
  - Arg is an argument of type atom,
  - ListAtoms is a non-empty list of distinct atoms.

This restriction forces Arg to be one of the atoms specified in the list ListAtoms.

**EXAMPLE:** An example of use of such restriction can be found in the `change(NCHANGE, VARIABLES, CTR)` constraint: `in_list(CTR,[=,≠,≤,≥,>)` forces the last argument CTR of the `change` constraint to take its value in the list of atoms `[=,≠,≤,≥,>]`.

> `change(1,(var – 4, var – 4, var – 4, var – 6), ≠)`
> `change(1,(var – 4, var – 4, var – 4, var – 6), 3)`

- **in_list(Arg,Attr,ListIntOrAtom)**
  - Arg is an argument of type collection,
  - Attr is an attribute of type `dvar` or of type `int` of the collection denoted by Arg.
  - When Attr is an attribute of type `int`, ListIntOrAtom is a non-empty list of distinct integers; Otherwise, when Attr is an attribute of type `atom`, ListIntOrAtom is a non-empty list of distinct atoms.

This restriction enforces for all items of the collection Arg, the attribute Attr to take its value within the list ListIntOrAtom.

- **in_attr(Arg1,Attr1,Arg2,Attr2)**
  - Arg1 is an argument of type collection,
  - Attr1 is an attribute of type dvar or of type `int` of the collection denoted by Arg1.
  - Arg2 is an argument of type collection,
  - Attr2 is an attribute of type `int` of the collection denoted by Arg2.

Let $S_2$ denote the set of values assigned to the Attr2 attributes of the items of the collection Arg2. This restriction enforces the following condition: for all items of the collection Arg1, the attribute Attr1 takes its value in the set $S_2$. 
2.1. DESCRIBING THE ARGUMENTS OF A GLOBAL CONSTRAINT

EXAMPLE: An example of use of such restriction can be found in the cumulatives\((TASKS, MACHINES, CTR)\) constraint: \(\text{in\_attr}(TASKS, \text{machine}, MACHINES, \text{id})\) enforces that the \text{machine} attribute of each task of the \text{TASKS} collection correspond to a machine identifier (i.e., an \text{id} attribute of the \text{MACHINES} collection).

\[
\begin{array}{l}
\text{cumulatives}(\langle \text{machine} - 1 \text{ origin} - 2 \text{ duration} - 2 \text{ end} - 4 \text{ height} - 2, \\
\text{machine} - 1 \text{ origin} - 2 \text{ duration} - 2 \text{ end} - 4 \text{ height} - 2, \\
\text{machine} - 2 \text{ origin} - 1 \text{ duration} - 4 \text{ end} - 5 \text{ height} - 5, \\
\text{machine} - 1 \text{ origin} - 4 \text{ duration} - 2 \text{ end} - 6 \text{ height} - 1), \\
(\text{id} - 1 \text{ capacity} - 9, \text{id} - 2 \text{ capacity} - 8), \leq) \\
\end{array}
\]

\[
\begin{array}{l}
\text{cumulatives}(\langle \text{machine} - 5 \text{ origin} - 2 \text{ duration} - 2 \text{ end} - 4 \text{ height} - 2, \\
\text{machine} - 1 \text{ origin} - 2 \text{ duration} - 2 \text{ end} - 4 \text{ height} - 2, \\
\text{machine} - 2 \text{ origin} - 1 \text{ duration} - 4 \text{ end} - 5 \text{ height} - 5, \\
\text{machine} - 1 \text{ origin} - 4 \text{ duration} - 2 \text{ end} - 6 \text{ height} - 1), \\
(\text{id} - 1 \text{ capacity} - 9, \text{id} - 2 \text{ capacity} - 8), \leq) \\
\end{array}
\]

- **distinct(\text{Arg, Attri})**

  - \text{Arg} is an argument of type \text{collection}.
  
  - \text{Attri} is an attribute of type \text{int} or \text{dvar}, or a list (possibly empty) of distinct attributes of type \text{int} or \text{dvar} of the collection denoted by \text{Arg}.

For all pairs of distinct items of the collection \text{Arg} this restriction enforces that there be at least one attribute specified by \text{Attri} with two distinct values. When \text{Attri} is the empty list all items of the collection \text{Arg} should be distinct.

**EXAMPLE:** An example of use of such restriction can be found in the cycle\((NCYCLE, NODES)\) constraint: \text{distinct}(NODES, \text{index}) enforces that all \text{index} attributes of the NODES collection take distinct values.

\[
\begin{array}{l}
\text{cycle}(2, (\text{index} - 1 \text{ succ} - 2, \text{index} - 2 \text{ succ} - 1, \text{index} - 3 \text{ succ} - 3)) \\
\text{cycle}(2, (\text{index} - 1 \text{ succ} - 2, \text{index} - 1 \text{ succ} - 1, \text{index} - 3 \text{ succ} - 3)) \\
\end{array}
\]

- **increasing_seq(\text{Arg, Attri})**

  - \text{Arg} is an argument of type \text{collection}.
  
  - \text{Attri} is an attribute of type \text{int} or a list of distinct attributes of type \text{int} of the collection denoted by \text{Arg}.

Let \(n\) and \(m\) respectively denote the number of items of the collection \text{Arg}, and the number of attributes of \text{Attri}. For item \(i\) of the collection \text{Arg} let \(t_i\) denote the tuple of values \(\langle v_{i,1}, v_{i,2}, \ldots, v_{i,m}\rangle\) where \(v_{i,j}\) is the value of attribute \(j\) of \text{Attri} of item \(i\) of \text{Arg}. The restriction enforces a strict lexicographical ordering on the tuples \(t_1, t_2, \ldots, t_n\).
CHAPTER 2. DESCRIBING GLOBAL CONSTRAINTS

EXAMPLE: An example of use of such restriction can be found in the `element_matrix(MAX_I, MAX_J, INDEX_I, INDEX_J, MATRIX, VALUE)` constraint: `increasing_seq(MATRIX, [i,j])` enforces that all items of the `MATRIX` collection be sorted in strictly increasing lexicographic order on the pair `(i, j).

def `element_matrix(2,2,1,2, (i = 1 j = 1 v = 4, i = 1 j = 2 v = 7, i = 2 j = 1 v = 1, i = 2 j = 2 v = 1), 7)`

def `element_matrix(2,2,1,2, (i = 1 j = 2 v = 4, i = 1 j = 1 v = 7, i = 2 j = 1 v = 1, i = 2 j = 2 v = 1), 7)`

- `non_increasing_size(Arg, Attr)`
  
  - `Arg` is an argument of type collection,
  
  - `Attr` is an attribute of the collection denoted by `Arg`. This attribute should be of type collection.

This restriction enforces for each pair of consecutive items `Arg[i], Arg[i+1]` that the number of items of the collection `Arg[i].Attr` is greater than or equal to the number of items of the collection `Arg[i+1].Attr`.

EXAMPLE: An example of use of such restriction can be found in the `k_used_by(SETS)` constraint: `non_increasing_size(SETS, set)` enforces for all consecutive pairs of items `SETS[i], SETS[i+1]` that the number of items of the collection `SETS[i].set` is not greater than or equal to the number of items of the collection `SETS[i+1].set`.

```python
def `k_used_by(set = {var = 5, var = 1, var = 1},
set = {var = 5, var = 1, var = 1},
set = {var = 5, var = 1})

def `k_used_by(set = {var = 5, var = 1, var = 1},
set = {var = 5, var = 1},
set = {var = 5, var = 1, var = 1})
```

- `required(Arg, Attrs)`
  
  - `Arg` is an argument of type collection,
  
  - `Attr` is an attribute or a list of distinct attributes of the collection denoted by `Arg`.

This restriction enforces that all attributes denoted by `Attr` be explicitly used within all items of the collection `Arg`.

EXAMPLE: An example of use of such restriction can be found in the `cumulative(TASKS, LIMIT)` constraint: `required(TASKS, height)` enforces that all items of the `TASKS` collection mention the height attribute.

```python
def `cumulative((origin = 2 duration = 2 end = 4 height = 2, origin = 2 duration = 2 end = 4 height = 2, origin = 1 duration = 4 end = 5 height = 5, origin = 2 duration = 2 end = 6 height = 1), 12)`

def `cumulative((origin = 2 duration = 2 end = 4, origin = 2 duration = 2 end = 4 height = 2, origin = 1 duration = 4 end = 5 height = 5, origin = 4 duration = 2 end = 6 height = 1), 12)`
```
The required restriction is usually systematically used for every attribute of a collection. It is not used when some attributes may be implicitly defined according to other attributes. In this context, we use the require at least restriction, which we now introduce.

- **require at least**(Atleast, Arg, Attrs)
  - Atleast is a positive integer,
  - Arg is an argument of type collection,
  - Attrs is a non-empty list of distinct attributes of the collection denoted by Arg. The length of this list should be strictly greater than Atleast.

This restriction enforces that at least Atleast attributes of the list Attrs be explicitly used within all items of the collection Arg.

**EXAMPLE:** An example of use of such restriction can be found in the `cumulative`(TASKS, LIMIT) constraint: `require at least`(2, TASKS [origin, duration, end]) enforces that all items of the TASKS collection mention at least two attributes from the list of attributes [origin, duration, end]. In this context, this stems from the equality origin + duration = end. This allows for retrieving the third attribute from the values of the two others.

```plaintext
cumulative((origin - 2 duration - 2 height - 2,
  origin - 2 end - 4 height - 2,
  duration - 4 end - 5 height - 5,
  origin - 4 duration - 2 end - 6 height - 1), 12)
cumulative((origin - 2 height - 2,
  origin - 2 duration - 2 end - 4 height - 2,
  origin - 1 duration - 4 end - 5 height - 5,
  origin - 4 duration - 2 end - 6 height - 1), 12)
```

- **same size**(Arg, Attr)
  - Arg is an argument of type collection,
  - Attr is an attribute of the collection denoted by Arg. This attribute should be of type collection.

This restriction enforces that all collections denoted by Attr have the same number of items.
CHAPTER 2. DESCRIBING GLOBAL CONSTRAINTS

- **Term\_Comparison Term\_2**

  - **Term\_1** is a term. A term is an expression that can be evaluated to one or possibly several integer values. The expressions we allow for a term are defined in the next paragraph.

  - **Comparison** is one of the following comparison operators ≤, ≥, <, >, =, ≠.

  - **Term\_2** is a term.

Let \( v_{1,1}, v_{1,2}, \ldots, v_{1,n_1} \) and \( v_{2,1}, v_{2,2}, \ldots, v_{2,n_2} \) be the values respectively associated with Term\_1 and with Term\_2. The restriction Term\_1 Comparison Term\_2 forces \( v_{1,i} \) Comparison \( v_{2,j} \) to hold for every \( i \in [1, n_1] \) and every \( j \in [1, n_2] \).

A term is one of the following expressions:

- \( e \) where \( e \) is an integer. The corresponding value is \( e \).

- \( |c| \) where \( c \) is an argument of type collection. The value of \( |c| \) is the number of items of the collection denoted by \( c \).

**EXAMPLE:** This kind of expression is for instance used in the restrictions of the `atleast` constraint: \( N \leq |\text{VARIABLES}| \) restricts \( N \) to be less than or equal to the number of items of the VARIABLES collection.

\[
\begin{align*}
\text{\texttt{atleast}}(2, \var{5}, \var{8}, \var{5}, 5) \\
\text{\texttt{atleast}}(4, \var{5}, \var{8}, \var{5}, 5)
\end{align*}
\]

- **\( \text{sum}(c.a) \)**, **\( \text{sum}(c.a) \)** denotes the sum of the values assigned to the attribute \( a \) of the collection denoted by \( c \). It is equal to 0 if the collection is empty.

- **\( \text{range}(c.a) \)**, **\( \text{range}(c.a) \)** denotes the difference between the maximum value and the minimum value plus one of the values assigned to the attribute \( a \) of the collection denoted by \( c \). It is equal to 0 if the collection is empty.

- **\( \text{minval}(c.a) \)**, **\( \text{minval}(c.a) \)** denotes the minimum over the values assigned to the attribute \( a \) of the collection denoted by \( c \). It is equal to 0 if the collection is empty.

**EXAMPLE:** An example of use of such restriction can be found in the `diffn(ORTHOTOPES)` constraint: `same_size(ORTHOTOPES, orth)` forces all the items of the ORTHOTOPES collection to be constituted from the same number of items (of type ORTHOTOPE). From a practical point of view, this forces the `diffn` constraint to take as its argument a set of points, a set of rectangles, . . . , a set of orthotopes.

\[
\begin{align*}
\text{\texttt{diffn}}((\text{orth} - \langle \text{ori} - 2 \text{ siz} - 2 \text{ end} - 4, \text{ori} - 1 \text{ siz} - 3 \text{ end} - 4 \rangle), \\
\text{orth} - \langle \text{ori} - 4 \text{ siz} - 4 \text{ end} - 8, \text{ori} - 3 \text{ siz} - 3 \text{ end} - 3 \rangle), \\
\text{orth} - \langle \text{ori} - 9 \text{ siz} - 2 \text{ end} - 11, \text{ori} - 4 \text{ siz} - 3 \text{ end} - 7 \rangle))
\end{align*}
\]
2.1. DESCRIBING THE ARGUMENTS OF A GLOBAL CONSTRAINT

– \texttt{maxval}(c.a). \texttt{maxval}(c.a) denotes the maximum over the values assigned to the attribute \(a\) of the collection denoted by \(c\). It is equal to 0 if the collection is empty.

– \texttt{nval}(c.a). \texttt{nval}(c.a) denotes the number of distinct values over the values assigned to the attribute \(a\) of the collection denoted by \(c\). It is equal to 0 if the collection is empty.

– \texttt{prod}(c.a). \texttt{prod}(c.a) denotes the product of the values assigned to the attribute \(a\) of the collection denoted by \(c\). It is equal to 1 if the collection is empty.

– \(t\), where \(t\) is an argument of type \texttt{int}. The value of \(t\) is the value of the corresponding argument.

\textbf{EXAMPLE:} This kind of expression is for instance used in the restrictions of the \texttt{atleast}(\(N\), VARIABLES, VALUE) constraint: \(N \geq 0\) forces the first argument of the \texttt{atleast} constraint to be greater than or equal to 0.

\(\triangleright \texttt{atleast}(2, \langle \text{var} - 5, \text{var} - 8, \text{var} - 5 \rangle, 5)\)

\(\blacktriangleright \texttt{atleast}(-1, \langle \text{var} - 5, \text{var} - 8, \text{var} - 5 \rangle, 5)\)

– \(v\), where \(v\) is an argument of type \texttt{dvar}. The value of \(v\) will be the value assigned to variable \(v\).\footnote{Restrictions are defined on the ground instance of a global constraint.}

\textbf{EXAMPLE:} This kind of expression is for instance used in the restrictions of the \texttt{among}(\(NVAR\), VARIABLES, VALUES) constraint: \(NVAR \geq 0\) forces the first argument of the \texttt{among} constraint to be greater than or equal to 0.

\(\triangleright \texttt{among}(2, \langle \text{var} - 5, \text{var} - 8, \text{var} - 5 \rangle, \langle \text{val} - 1, \text{val} - 5 \rangle)\)

\(\blacktriangleright \texttt{among}(-1, \langle \text{var} - 5, \text{var} - 8, \text{var} - 5 \rangle, \langle \text{val} - 1, \text{val} - 5 \rangle)\)

– \(s\), where \(s\) is an argument of type \texttt{sint} or \texttt{svar}. The values denoted by \(s\) are all the values of the corresponding set.

\textbf{EXAMPLE:} This kind of expression is for instance used in the restrictions of the \texttt{open\_alldifferent}(\(S\), VARIABLES) constraint: \(S \geq 1\) forces all elements of the set corresponding to the first argument of the \texttt{open\_alldifferent} constraint to be greater than or equal to 1.

\(\triangleright \texttt{open\_alldifferent}([1, 2, 3], \langle \text{var} - 5, \text{var} - 8, \text{var} - 3, \text{var} - 8, \text{var} - 9 \rangle)\)

\(\blacktriangleright \texttt{open\_alldifferent}([0, 1, 2, 3], \langle \text{var} - 5, \text{var} - 8, \text{var} - 3, \text{var} - 8, \text{var} - 9 \rangle)\)

– \(c.a\), where \(c\) is an argument of type \texttt{collection} and \(a\) an attribute of \(c\) of type \texttt{int} or \texttt{dvar}. The values denoted by \(c.a\) are all the values corresponding to attribute \(a\) for the different items of \(c\). When \(c.a\) designates a domain variable we consider the value assigned to that variable.
CHAPTER 2. DESCRIBING GLOBAL CONSTRAINTS

**EXAMPLE:** This kind of expression is for instance used in the restrictions of the `cumulative` constraint: TASKS.duration ≥ 0 enforces for all items of the TASKS collection that the duration attribute be greater than or equal to 0.

```plaintext
cumulative((origin − 2 duration − 2 end − 4 height − 2,
origin − 2 duration − 2 end − 4 height − 2,
origin − 1 duration − 4 end − 5 height − 5,
origin − 1 duration − 4 end − 5 height − 5), 12)
```

- `c.a`, where `c` is an argument of type collection and `a` an attribute of `c` of type `sint` or `svar`. The values denoted by `c.a` are all the values belonging to the sets corresponding to attribute `a` for the different items of `c`. When `c.a` designates a set variable we consider the values that finally belong to that set.

**EXAMPLE:** This kind of expression is for instance used in the restrictions of the `inverse_set` constraint: X.x ≥ 1 enforces for all items of the X collection that all the potential elements of the set variable associated with the x attribute be greater than or equal to 1.

```plaintext
inverse_set((index − 1 x − {2, 4},
index − 2 x − {4},
index − 3 x − {1},
index − 4 x − {4}),
(index − 1 y − {3},
index − 2 y − {1},
index − 3 y − {},
index − 4 y − {1, 2, 4},
index − 5 y − {}))
```

- `min(t_1, t_2)` or `max(t_1, t_2)`, where `t_1` and `t_2` are terms. Let \( \mathcal{V}_1 \) and \( \mathcal{V}_2 \) denote the sets of values respectively associated with the terms `t_1` and `t_2`. Let \( \min(\mathcal{V}_1) \), \( \max(\mathcal{V}_1) \) and \( \min(\mathcal{V}_2) \), \( \max(\mathcal{V}_2) \) denote the minimum and maximum values of \( \mathcal{V}_1 \) and \( \mathcal{V}_2 \). The value associated with `min(t_1, t_2)` is \( \min(\min(\mathcal{V}_1), \min(\mathcal{V}_2)) \), while the value associated with `max(t_1, t_2)` is \( \max(\max(\mathcal{V}_1), \max(\mathcal{V}_2)) \).
2.1. DESCRIBING THE ARGUMENTS OF A GLOBAL CONSTRAINT

• We can use a disjunction between two restrictions.

EXAMPLE: This kind of expression is for instance used in the Typical slot of the among_low_up constraint:
\[
\text{among\_low\_up}(\text{LOW, UP, VARIABLES, VALUES}) \quad \text{constraint: LOW} \lor \text{UP} < |\text{VARIABLES}|
\]
forces the pair LOW, UP to impose a restriction on the variables of the VARIABLES collection.

\[
\begin{align*}
\triangleright & \text{among\_low\_up}(1, 2, (9, 2, 4, 5), (0, 2, 4, 6, 9)) \\
\triangleright & \text{among\_low\_up}(0, 3, (9, 2, 4, 5), (0, 2, 4, 6, 9)) \\
\triangleright & \text{among\_low\_up}(1, 4, (9, 2, 4, 5), (0, 2, 4, 6, 9)) \\
\triangleright & \text{among\_low\_up}(0, 4, (9, 2, 4, 5), (0, 2, 4, 6, 9))
\end{align*}
\]

\footnote{\texttt{\textbackslash /} denotes an integer division, a division in which the fractional part is discarded.}

• Finally, we can also use a constraint \(C\) of the catalogue for expressing a restriction as long as that constraint is not defined according to the constraint under consideration. The constraint \(C\) should have a graph-based or an automaton-based description so that its meaning is explicitly defined.

EXAMPLE: An example of use of such restriction can be found in the sort_permutation constraint:
\[
\text{sort\_permutation}(\text{FROM, PERMUTATION, TO}) \quad \text{constraint: alldifferent(\text{PERMUTATION})}
\]
is used to express that the variables of the second argument of the sort_permutation constraint should take distinct values.

2.1.4 Declaring a global constraint

Declaring a global constraint consists of providing the following information:

\[
\begin{align*}
\text{interval}(\text{NVAL, VARIABLES, SIZE\_INTERVAL}) \quad \text{constraint: NVAL} & \geq \min(1, |\text{VARIABLES}|) \text{ forces NVAL to be greater than or equal to the minimum of 1 and the number of items of the VARIABLES collection.} \\
\triangleright & \text{interval}(2, (\text{var} - 3, \text{var} - 1, \text{var} - 9, \text{var} - 1, \text{var} - 9), 4) \\
\triangleright & \text{interval}(0, (\text{var} - 3, \text{var} - 1, \text{var} - 9, \text{var} - 1, \text{var} - 9), 4)
\end{align*}
\]

\[
t_{1} \text{ op } t_{2}
\]
where \(t_{1}\) and \(t_{2}\) are terms and \(\text{op}\) one of the operators \(+\), \(-\), \(*\) or \(/\).

\footnote{\texttt{\textbackslash /} denotes an integer division, a division in which the fractional part is discarded.}

EXAMPLE: This kind of expression is for instance used in the restrictions of the

\[
\begin{align*}
\text{relaxed\_sliding\_sum}(\text{ATLEAST, ATMOST, LOW, UP, SEQ, VARIABLES}) \quad \text{constraint: ATMOST} & \leq |\text{VARIABLES}| - \text{SEQ} + 1 \text{ forces ATMOST to be less than or equal to an arithmetic expression that corresponds to the number of sequences of SEQ consecutive variables in a sequence of |VARIABLES| variables.} \\
\triangleright & \text{relaxed\_sliding\_sum}(3, 4, 3, 7, 4, (\text{var} - 2, \text{var} - 4, \text{var} - 2, \text{var} - 0, \text{var} - 0, \text{var} - 3, \text{var} - 4)) \\
\triangleright & \text{relaxed\_sliding\_sum}(3, 9, 3, 7, 4, (\text{var} - 2, \text{var} - 4, \text{var} - 2, \text{var} - 0, \text{var} - 0, \text{var} - 3, \text{var} - 4))
\end{align*}
\]

\[
al\text{LOW} \lor \text{UP} \leq |\text{VARIABLES}|
\]
forces the pair LOW, UP to impose a restriction on the variables of the VARIABLES collection.

\[
\begin{align*}
\triangleright & \text{among\_low\_up}(1, 2, (9, 2, 4, 5), (0, 2, 4, 6, 9)) \\
\triangleright & \text{among\_low\_up}(0, 3, (9, 2, 4, 5), (0, 2, 4, 6, 9)) \\
\triangleright & \text{among\_low\_up}(1, 4, (9, 2, 4, 5), (0, 2, 4, 6, 9)) \\
\triangleright & \text{among\_low\_up}(0, 4, (9, 2, 4, 5), (0, 2, 4, 6, 9))
\end{align*}
\]

\footnote{Since when both, LOW \leq 0 and UP \geq |VARIABLES|, the corresponding among_low_up constraint always holds.}
**CHAPTER 2. DESCRIBING GLOBAL CONSTRAINTS**

- A term constraint \( A_1, A_2, \ldots, A_n \), where constraint corresponds to the name of the global constraint and \( A_1, A_2, \ldots, A_n \) to its arguments.

- A possibly empty list of type declarations, where each declaration has the form `type:type_declaration;` type is the name of the new type we define and type declaration is a basic data type, a compound data type or a type previously defined.

- An argument declaration \( A_1:T_1, A_2:T_2, \ldots, A_n:T_n \) giving for each argument \( A_1, A_2, \ldots, A_n \) of the global constraint constraint its type. Each type is a basic data type, a compound data type, or a type that was declared in the list of type declarations.

- A possibly empty list of restrictions, where each restriction is one of the restrictions described in Section 2.1.3 on page 9.

---

**EXAMPLE:** The arguments of the `all_differ_from_at_least_k_pos` constraint are described by:

<table>
<thead>
<tr>
<th>Constraint</th>
<th>all_differ_from_at_least_k_pos(K, VECTORS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type(s)</td>
<td>VECTOR − collection(var − dvar)</td>
</tr>
<tr>
<td>Argument(s)</td>
<td>K − int</td>
</tr>
<tr>
<td></td>
<td>VECTORS − collection(vec − VECTOR)</td>
</tr>
<tr>
<td>Restriction(s)</td>
<td>required(VECTOR, var)</td>
</tr>
<tr>
<td></td>
<td>K ≥ 0</td>
</tr>
<tr>
<td></td>
<td>required(VECTORS, vec)</td>
</tr>
<tr>
<td></td>
<td>same_size(VECTORS, vec)</td>
</tr>
</tbody>
</table>

The first line indicates that the `all_differ_from_at_least_k_pos` constraint has two arguments: \( K \) and VECTORS. The second line declares a new type VECTOR, which corresponds to a collection of variables. The third line indicates that the first argument \( K \) is an integer, while the fourth line tells that the second argument VECTORS corresponds to a collection of vectors of type VECTOR. Finally the four restrictions respectively enforce that:

- All the items of the VECTOR collection mention the var attribute,
- \( K \) be greater than or equal to 0,
- All the items of the VECTORS collection mention the vec attribute,
- All the vectors have the same number of components.

---

**2.1.5 Describing symmetries between arguments**

Given a satisfied ground instance of a global constraint constraint, it is often the case that the constraint is still satisfied \([113, 156]\) if we permute:

- Some of its arguments.

  E.g., consider the disequality constraint `neq(X, Y)`, which enforces \( X \) being assigned an integer value that is different from \( Y \). Given the solution `neq(3, 5)` we can swap both arguments and still get a solution (i.e., `neq(5, 3)`).
2.1. Describing the Arguments of a Global Constraint

- **Items of some collections that are passed as one of its arguments.**
  E.g., consider the `alldifferent(VARIABLES)` constraint, which imposes all variables of the collection `VARIABLES` being assigned a distinct integer value. Given the solution `alldifferent((5,1,9,3))` we can swap any pair of items and still get a solution. For instance, if we swap the first and fourth items we still get a solution (i.e., `alldifferent((3,1,9,5))`).

- **Attributes of some items of some of its collections.**
  E.g., given a collection of pairs `PAIRS`, where each pair has two attributes `x` and `y`, the `npair(N,PAIRS)` constraint enforces `N` being the number of distinct pairs in `PAIRS`. Given the solution `npair(3,(x−3 y−1,x−1 y−5,x−3 y−1,x−1 y−5,x−1 y−3))` we can interchange attributes `x` and `y` and still get a solution (i.e., `npair(3,(x−1 y−3,x−5 y−1,x−1 y−3,x−5 y−1,x−3 y−1))`).

- **A pair of values with respect to an attribute of some of its collections.**
  E.g., consider the `bin_packing` constraint, which assigns items to bins in such a way that the total weight of the items in each bin does not exceed an overall fixed capacity. Each item has a `bin` and a `weight` attributes, which respectively give the bin to which the item will be assigned, and the weight of the item. Given the solution `bin_packing(5,(bin−3 weight−4,bin−1 weight−3,bin−3 weight−1))`, we can interchange all occurrences of value 3 with all occurrences of value 1 with respect to the `bin` attribute. After this swap of values we get the new solution `bin_packing(5,(bin−1 weight−4,bin−3 weight−3,bin−1 weight−1))`. This simply consists of swapping the content of two bins. Since all bins have the same capacity we still get a solution.

We provide the following moves, where each move is described by (1) an explicit fact (i.e., a meta-data), (2) a textual explanation, and (3) several concrete examples:

- **args(PERMUTATION)** denotes the fact that we swap the arguments of a constraint with respect to a given permutation. Arguments which are exchanged must have the same type under the hypothesis that they are ground (i.e., for instance the basic data types `int` and `dvar`, which respectively denote an integer value and a domain variable can be exchanged since a ground domain variable corresponds to an integer value). The permutation `PERMUTATION` is described by using standard notation, that is by providing the different cycles of the permutation.

**EXAMPLE 1:** As a first example where we can swap two arguments, consider the `eq_cst(VAR1,VAR2,CST2)` constraint which, given two domain variables `VAR1`, `VAR2` and an integer value `CST2`, enforces the condition `VAR1 = VAR2 + CST2`. Within the electronic catalogue this is represented by the following meta-data, `args([[VAR1],[VAR2,CST2]])`, to which corresponds the following textual form:

  arguments are permutable w.r.t. permutation (VAR1) (VAR2, CST2).

Note that, even if arguments `VAR2` and `CST2` do not have the same type (i.e., `VAR2` is a domain variable, while `CST2` is an integer value), both arguments can be exchanged since we consider the ground case. For instance, since `eq_cst(8,2,6)` is satisfied, `eq_cst(8,6,2)` is also satisfied.
EXAMPLE 2: As a second example where we can swap several arguments, consider the \texttt{common([NCOMMON1, NCOMMON2, VARIABLES1, VARIABLES2])} constraint which, given two domain variables \texttt{NCOMMON1}, \texttt{NCOMMON2} and two collections of domain variables \texttt{VARIABLES1, VARIABLES2}, enforces the following two conditions:

- \texttt{NCOMMON1} is the number of variables of \texttt{VARIABLES1} assigned a value in \texttt{VARIABLES2}.
- \texttt{NCOMMON2} is the number of variables of \texttt{VARIABLES2} assigned a value in \texttt{VARIABLES1}.

Within the electronic catalogue this is represented by the following meta-data, 
\texttt{args([\{NCOMMON1, NCOMMON2\}, \{VARIABLES1, VARIABLES2\}])}, to which corresponds the following textual form:

\textit{arguments are permutable w.r.t. permutation (NCOMMON1, NCOMMON2) (VARIABLES1, VARIABLES2).}

For instance, since \texttt{common(3, 4, \{1, 9, 1, 5\}, \{2, 1, 9, 9, 6, 9\})} is satisfied, \texttt{common(4, 3, \{2, 1, 9, 9, 6, 9\}, \{1, 9, 1, 5\})} is also satisfied.

- \texttt{items(COLLECTION, PERMUTATIONS)} denotes the fact that we can permute the items of the collection \texttt{COLLECTION} with respect to a permutation belonging to a given set of permutations \texttt{PERMUTATIONS}:

  - \texttt{COLLECTION} stands for one of the following:
    1. An argument \texttt{ARG} of the global constraint that corresponds to a \texttt{collection} of items.
    2. A term \texttt{ARG.attr}, where \texttt{attr} is an attribute of a \texttt{collection} of items that is an argument \texttt{ARG} of the global constraint; in addition, the type of \texttt{attr} is itself a collection. Given a collection \texttt{ARG} of \texttt{m} items \{\texttt{ARG[1]}, \texttt{ARG[2]}, \ldots, \texttt{ARG[m]}\}, a permutation of \texttt{PERMUTATIONS}, not necessarily the same, is applied on the items of a subset of the set of collections \{\texttt{ARG[1].attr}, \texttt{ARG[2].attr}, \ldots, \texttt{ARG[m].attr}\}.

  - \texttt{PERMUTATIONS} represents a set of permutations. It can take one of the following values:
    1. \texttt{all} stands for \textit{all possible permutations}. Note that this case is a little artificial since it does not really correspond to a symmetry of the constraint, but rather to the use of a collection for representing a set of variables. But, to our best knowledge in 2010, concrete solvers do also not use sets of variables but rather collections, lists or arrays of variables.
    2. \texttt{reverse} stands for the set that only contains the permutation that maps the sequence \texttt{e_1, e_2, \ldots, e_n} to \texttt{e_n, e_{n-1}, \ldots, e_1}.
    3. \texttt{shift} stands for the set that only contains the permutation that maps the sequence \texttt{e_1, e_2, \ldots, e_n} to \texttt{e_n, e_1, \ldots, e_{n-1}}.
EXAMPLE 1: As a first example, consider the \texttt{alldifferent(VARIABLES)} constraint, which has one single argument corresponding to a collection of variables which must all be assigned distinct values. Within the electronic catalogue this is represented by the following meta-data, \texttt{items(VARIABLES, all)}, to which corresponds the following textual form:

\begin{quote}
\texttt{items of VARIABLES are permutable.}
\end{quote}

For instance, since \texttt{alldifferent((1, 4, 9))} is satisfied, all permutations of \((1, 4, 9)\) (i.e., \((1, 4, 9), (1, 9, 4), (4, 1, 9), (4, 9, 1), (9, 1, 4), (9, 4, 1)\)) correspond to valid solutions of the \texttt{alldifferent} constraint.

EXAMPLE 2: As a second example, consider the \texttt{k_same(SETS)} constraint, which has one single argument corresponding to a collection of sets, where each set is a collection of domain variables that must be assigned the same set of values (i.e., \texttt{k_same} enforces an equality between multisets). The argument \texttt{SETS} is a collection, where each item consists of one single \texttt{set} attribute. The type of a \texttt{set} attribute is a collection of domain variables. Within the electronic catalogue this is represented by the following meta-data, \texttt{items(SETS.set, all)}, to which corresponds the following textual form:

\begin{quote}
\texttt{items of SETS.set are permutable.}
\end{quote}

For instance, since \texttt{k_same((\texttt{set} \rightarrow \{1, 4, 4\}, \texttt{set} \rightarrow \{4, 4, 1\}, \texttt{set} \rightarrow \{1, 4, 4\}))} is satisfied, it is also satisfied for all permutations of the elements of its second set \(\{4, 4, 1\}\), i.e.:

\begin{itemize}
  \item \texttt{k_same((\texttt{set} \rightarrow \{1, 4, 4\}, \texttt{set} \rightarrow \{4, 4, 1\}, \texttt{set} \rightarrow \{1, 4, 4\}))},
  \item \texttt{k_same((\texttt{set} \rightarrow \{1, 4, 4\}, \texttt{set} \rightarrow \{4, 1, 4\}, \texttt{set} \rightarrow \{1, 4, 4\}))},
  \item \texttt{k_same((\texttt{set} \rightarrow \{1, 4, 4\}, \texttt{set} \rightarrow \{4, 4, 1\}, \texttt{set} \rightarrow \{1, 4, 4\}))}.
\end{itemize}

- \texttt{items_sync(COLLECTIONS, PERMUTATIONS)} denotes the fact that we can permute the items of several collections \texttt{COLLECTIONS} with respect to a permutation belonging to a given set of permutations \texttt{PERMUTATIONS} in such a way that \texttt{one and the same permutation is used on all collections} (i.e., therefore the keyword \texttt{items_sync} which stands for \texttt{items synchronisation}):

  \begin{itemize}
    \item \texttt{COLLECTIONS} stands for a non-empty list of terms of the form \texttt{ARG} or \texttt{ARG.attr}, where \texttt{ARG} is an argument of the global constraint that corresponds to a collection, and \texttt{attr} is an attribute of \texttt{ARG} such that its type is itself a collection. In addition, we also have the following restrictions:
      \begin{enumerate}
        \item If \texttt{COLLECTIONS} contains one single element then this element has the form \texttt{ARG.attr}. This is done to allow to designate more than one single collection.
        \item All collections designated by \texttt{COLLECTIONS} have the same type as well as the same number of items.
      \end{enumerate}
    
    The \texttt{same permutation} of \texttt{PERMUTATIONS} is applied on the items of the different collections referenced by \texttt{COLLECTIONS}.
    
  - As for the symmetry keyword \texttt{items}, \texttt{PERMUTATIONS} represents a set of permutations. It can take the same set of values as before, namely:
    \begin{enumerate}
      \item all stands for \texttt{all possible permutations}.
    \end{enumerate}
2. reverse stands for the set that only contains the permutation that maps the sequence \( e_1, e_2, \ldots, e_n \) to \( e_n, e_{n-1}, \ldots, e_1 \).
3. shift stands for the set that only contains the permutation that maps the sequence \( e_1, e_2, \ldots, e_n \) to \( e_n, e_1, \ldots, e_{n-1} \).

**EXAMPLE 1**: As a first example, consider the `consecutive_groups_of_ones` constraint, which has two arguments `GROUP_SIZES` and `VARIABLES` respectively corresponding to a collection of positive integers and to a collection of 0-1 domain variables. The constraint imposes that the \( m \) successive maximum groups of consecutive ones of `VARIABLES` have sizes `GROUP_SIZES[1]`, `GROUP_SIZES[2]`, \ldots, `GROUP_SIZES[m]`. Note that, if we reverse the items of both `GROUP_SIZES` and `VARIABLES`, we still have a solution. Within the electronic catalogue this is represented by the following meta-data, `items_sync([GROUP_SIZES,VARIABLES],reverse)`, to which corresponds the following textual form:

\[
\text{items of \text{GROUP\_SIZES} and \text{VARIABLES} are simultaneously reversible.}
\]

For instance, since `consecutive_groups_of_ones([2,1],[1,1,0,0,0,1,0])` is a solution, `consecutive_groups_of_ones([1,2],[0,1,0,0,0,1,1])` is also a valid solution.

**EXAMPLE 2**: As a second example, consider the `nvector` constraint, which has two arguments `NVEC` and `VECTORS` respectively corresponding to a domain variable and to a collection of collections of domain variables, where all collections have the same number of items. The unique attribute of `VECTORS` is denoted by `vec` and its type is a collection of domain variables. Each collection is interpreted as a vector and two vectors are distinct if and only if they differ in at least one component. The `nvector` constraint enforces `NVEC` to be equal to the number of distinct vectors within `VECTORS`. If we permute the components of all vectors with respect to a same permutation we still have the same number of distinct vectors. Within the electronic catalogue this is represented by the following meta-data, `items_sync([VECTORS,vec],all)`, to which corresponds the following textual form:

\[
\text{items of \text{VECTORS}.\text{vec} are permutable (same permutation used).}
\]

For instance, since `nvector([2,\langle \text{vec} = \langle 1, 1, 8 \rangle, \text{vec} = \langle 5, 1, 6 \rangle, \text{vec} = \langle 1, 1, 8 \rangle\rangle])` is a solution, any permutation applied simultaneously to the three components of each vector leads to a solution, i.e.:

- `nvector([2,\langle \text{vec} = \langle 1, 1, 8 \rangle, \text{vec} = \langle 5, 1, 6 \rangle, \text{vec} = \langle 1, 1, 8 \rangle\rangle]),`
- `nvector([2,\langle \text{vec} = \langle 1, 8, 1 \rangle, \text{vec} = \langle 5, 6, 1 \rangle, \text{vec} = \langle 1, 8, 1 \rangle\rangle]),`
- `nvector([2,\langle \text{vec} = \langle 1, 1, 8 \rangle, \text{vec} = \langle 1, 5, 6 \rangle, \text{vec} = \langle 1, 1, 8 \rangle\rangle]),`
- `nvector([2,\langle \text{vec} = \langle 1, 8, 1 \rangle, \text{vec} = \langle 1, 6, 5 \rangle, \text{vec} = \langle 1, 8, 1 \rangle\rangle]),`
- `nvector([2,\langle \text{vec} = \langle 8, 1, 1 \rangle, \text{vec} = \langle 6, 1, 5 \rangle, \text{vec} = \langle 8, 1, 1 \rangle\rangle]),`
- `nvector([2,\langle \text{vec} = \langle 8, 1, 1 \rangle, \text{vec} = \langle 6, 5, 1 \rangle, \text{vec} = \langle 8, 1, 1 \rangle\rangle]).`

- `attrs(COLLECTION,PERMUTATION)` denotes the fact that we can permute the attributes of the collection `COLLECTION`, not necessarily all items, with respect to a permutation `PERMUTATION`. Attributes that are exchanged must have the same type under the hypothesis that they are ground (e.g., an attribute `attr1` of type `int` can be exchanged with an attribute `attr2` of type `dvar`.)
• \texttt{attrs\_sync(COLLECTION,PERMUTATION)} denotes the fact that we can permute the attributes of the collection \textit{COLLECTION}, \textit{necessarily all items}, with respect to a permutation \textit{PERMUTATION}. As before, attributes that are exchanged must have the same type under the hypothesis that they are ground.

\textbf{EXAMPLE:} As an example, consider the \texttt{crossing(NCROSS,SEGS)} constraint, which enforces NCROSS to be equal to the number of line-segments intersections between the line-segments defined by the \textit{SEGS} collection. Each line-segment is defined by the coordinates (\textit{ox,oy}) and (\textit{ex,ey}) of its two extremities. Note that we can exchange the role of the \textit{x} and \textit{y} axes without affecting the number of line-segments intersections. Within the electronic catalogue this is represented by the following meta-data, \texttt{attrs\_sync(SEGS,[[\textit{ox,oy}],[\textit{ex,ey}]])}, to which corresponds the following textual form:

\begin{quote}
\textit{attributes of SEGS are permutable w.r.t. permutation (ox,oy) (ex,ey)}
\end{quote}

(permuation applied to all items).

For instance, since \texttt{crossing((3, (\textit{ox}−1 \textit{oy}−4 \textit{ex}−9 \textit{ey}−2,\textit{ox}−1 \textit{oy}−1 \textit{ex}−3 \textit{ey}−5,\textit{ox}−3 \textit{oy}−2 \textit{ex}−7 \textit{ey}−4,\textit{ox}−9 \textit{oy}−1 \textit{ex}−9 \textit{ey}−4))}) is a solution, \texttt{crossing((3, (\textit{ox}−4 \textit{oy}−1 \textit{ex}−2 \textit{ey}−9,\textit{ox}−1 \textit{oy}−1 \textit{ex}−5 \textit{ey}−3,\textit{ox}−2 \textit{oy}−3 \textit{ex}−4 \textit{ey}−7,\textit{ox}−1 \textit{oy}−9 \textit{ex}−4 \textit{ey}−9))}) is also a valid solution.

• \texttt{vals(ATTRIBUTES,PARTITION,PAIRS,SOURCE,TARGET)} denotes the fact that we can permute some source value with some distinct target value. The kind of value permutation we can perform is parameterized by five parameters:

− \texttt{ATTRIBUTES} is a list of paths of the form \texttt{ARG0 or ARG1· · · ARGn.attr (n \geq 1)}, where:
  * \texttt{ARG0} is an argument of the global constraint of type \texttt{domain variable}, \texttt{integer}, or collection of \texttt{domain variables} or \texttt{integers}.
  * \texttt{ARG1· · · ARGn.attr} is a path to an integer attribute or to a collection of integers attribute of the global constraint. \texttt{ARG1, ARG2, · · · , ARGn} are collections and \texttt{attr} is an attribute of \texttt{ARGn} of type \texttt{domain variable}, \texttt{integer}, or collection of \texttt{domain variables} or \texttt{integers}. In this last context, all collections have the same number of items since we can only...
exchange tuples of values that have the same number of components. The path does not necessarily start from a top level collection. Its purpose is to define the scope where the exchange of values, or tuples of values, will take place. Note that:

- The case corresponding to \( \text{ARG0} \) allows to express the fact that the value of an integer argument can be changed in such a way that we still have a solution.
- The case when \( \text{ARG1} \) is not a top level collection allows to express the fact that the exchange of value takes place within a nested collection. In this context this implicitly defines several scopes for the exchange of values.
- The case where \( \text{ARG1, \ldots, ARGn.attr} \) is a path to a collection of variables or integers allows expressing swap between tuples of values (i.e., the exchange of values is generalized to the exchange of tuples of values).

- \text{PARTITION} usually defines a partition \( P \) of integer values. Only when \( \text{ARG1, \ldots, ARGn.attr} \) is a path to a collection of variables or integers, \text{PARTITION} defines a partition of tuples of integer values. For the time being we focus on the first case, i.e., a partition of integer values. Its aim is to define classes of values from which the source and target values will be selected. In order to define a partition \( P \) we first introduce the notion of set of values generator. Within these definitions, \( u \) and \( v \) both denote (1) an integer value, or (2) an argument of the constraint of type \text{integer} or \text{domain variable}, or (3) a term of the form \(|\text{ARG}|\) where \( \text{ARG} \) is an argument of type collection denoting the number of items of the collection, (4) a sum or difference of elements of the form (1), (2) or (3). We have two kinds of generators, namely:

- A basic set of values generator is defined by one of those:
  - \( \text{ARG.attr} \), where \( \text{ARG} \) is an argument of type collection and \( \text{attr} \) is an attribute of \( \text{ARG} \) of type \text{integer} or \text{domain variable}, denotes the set of all values assigned to \( \text{ARG.attr} \).
  - \( \text{notin}(\text{ARG.attr}) \), where \( \text{ARG} \) is an argument of type collection and \( \text{attr} \) is an attribute of \( \text{ARG} \) of type \text{integer} or \text{domain variable}, denotes the set of all elements of \( \mathbb{Z} \) that are not assigned to \( \text{ARG.attr} \).
  - \( \text{diff}(\text{ARG1.attr1, ARG2.attr2}) \), where \( \text{ARG1} \) (respectively \( \text{ARG2} \)) is an argument of type collection and \( \text{attr1} \) (respectively \( \text{attr2} \)) is an attribute of \( \text{ARG1} \) (respectively \( \text{ARG2} \)) of type \text{integer} or \text{domain variable}, denotes the set of all elements of \( \mathbb{Z} \) that are assigned to \( \text{ARG1.attr1} \) but not to \( \text{ARG2.attr2} \).
  - \( u \), denotes the set \( \{u\} \).
  - \( \text{cmp}(u, (\text{cmp} \in \{=, \neq, <, \geq, >, \leq\})) \), denotes the set of all integers \( e \) such that the comparison \( e \text{ cmp } u \) holds.
2.1. DESCRIBING THE ARGUMENTS OF A GLOBAL CONSTRAINT

- \(\text{in}(u, v), (u \leq v)\), denotes the set of all integers located in interval \([u, v]\).
- \(\text{notin}(u, v), (u \leq v)\), denotes the set of all integers not located in interval \([u, v]\).
- \(\text{mod}(u, v), (0 < v < u, u, v \in \mathbb{N}^+)\), denotes all integer values in \(\mathbb{Z}\) that have \(v\) as remainder when divided by \(u\)\(^7\).

* Given set of values generators \(S_1, S_2, \ldots, S_n\) \((n \geq 2)\), a compound set of values generator is defined by:
  - \([S_1, S_2, \ldots, S_n]\) denotes all values that are in at least one of the sets \(S_1, S_2, \ldots, S_n\).
  - \(\text{notin}([S_1, S_2, \ldots, S_n])\) denotes all values of \(\mathbb{Z}\) that are not in any set \(S_1, S_2, \ldots, S_n\).

We now describe the different partition generators. Within the description, \(S\) and \(D\) denote set of values generators. Classes of a partition are ordered. Unless explicitly specified, classes are ordered with respect to the smallest element they contain.

- \(\text{int}\) denotes a partition \(P\) where, to each element of \(\mathbb{Z}\) corresponds a specific class of \(P\) containing just that element.
- \(\text{int}(S)\) denotes a partition \(P\) where, to each element of \(S\) corresponds a specific class of \(P\) containing just that element.
- \(\text{all}\) denotes a partition \(P\) containing one single class of values corresponding to all integer values in \(\mathbb{Z}\).
- \(\text{all}(S)\) denotes a partition \(P\) containing one single class of values corresponding to the elements of \(S\).
- \(\text{comp}(S)\) denotes partition \(P\) containing two classes of values: a first class corresponding to the elements of \(S\), and a second class consisting of all elements of \(\mathbb{Z}\) that are not in \(S\).
- \(\text{comp_diff}(S, D)\) denotes partition \(P\) containing two classes of values: a first class corresponding to the elements of \(S\) but not in \(D\), and a second class consisting of all elements of \(\mathbb{Z}\) that are neither in \(S\) nor in \(D\).
- \(\text{intervals}(u), (u > 0)\), denotes a partition \(P\) containing intervals of the form \([k \cdot u, k \cdot u + u - 1], k \in \mathbb{Z}\).
- \(\text{mod}(u), (u > 0)\), denotes a partition \(P\) such that each class of \(P\) is made up from all integers in \(\mathbb{Z}\) that have the same remainder when divided by \(u\)\(^8\).
- \(\text{part}(P)\), where \(P\) is a collection of collections of integers passed as one of the arguments of the constraint, where each integer occurs once, denotes a partition \(P\) such that each class corresponds to the elements of one of the collections of \(P\). Classes are ordered with respect to their occurrence in \(P\).

\(^7\)remainder\((a, n) = a - n\lfloor\frac{a}{n}\rfloor\).
\(^8\)remainder\((a, n) = a - n\lfloor\frac{a}{n}\rfloor\).
When \textsc{partition} defines a partition of tuples, where each tuple consists of \( k \) integers, \textsc{partition} can only be set to \texttt{int}. In this context \texttt{int} denotes a partition \( \mathcal{P} \) where, to each element of \( \mathbb{Z}^k \) corresponds a specific class of \( \mathcal{P} \) containing just that element.

- \textsc{pairs} is one of the symbols \( \neq, =, \leq, \geq, >, \leq \), or \texttt{dontcare}. It specifies a set of pairs \( \{(p_{i_1}, p_{j_1}), (p_{i_2}, p_{j_2}), \ldots, (p_{i_n}, p_{j_n})\} \) of elements of the partition \( \mathcal{P} \) such that, when \textsc{pairs} is different from \texttt{dontcare}, the condition \( i_k \textsc{pairs} j_k \) holds for all \( k \in [1, n] \). The aim of the \textsc{pairs} parameter is to allow to specify which partitions of \( \mathcal{P} \) the source value \( u \) and the target value \( v \) should belong to. In fact there should exist a pair \( (p_{i_k}, p_{j_k}), (k \in [1, n]) \), such that \( u \in p_{i_k} \) and \( v \in p_{j_k} \).

- \textsc{source} is one of the options \texttt{all} or \texttt{dontcare}:
  * When set to \texttt{all} it indicates that all occurrences of the source value should be replaced by the target value. All occurrences of the target value, if it is used, should also be replaced by the source value.
  * When set to \texttt{dontcare} it tells that not necessarily all occurrences of the source value should be replaced. The target value is left unchanged.

- \textsc{target} is one of the options \texttt{in} or \texttt{dontcare}:
  * When set to \texttt{in} it indicates that the target value should correspond to an already existing value of \texttt{arg.attr}.
  * When set to \texttt{dontcare} it tells that the target value can either correspond to an already existing value of \texttt{arg.attr}, or designate a new value.

We now define the set of conditions we must have in order to exchange a source and a target values. Given,

1. a ground instance of a global constraint \( C \),
2. a path \textsc{path} that designates either an argument of type integer, or an integer attribute of a collection that occurs, possibly in a nested way, as one of the arguments of \( C \),
3. the sets of values \( \mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_h \) that are assigned to \textsc{path} in the ground instance of \( C \),\(^{10}\)
4. a partition of integer values \( \mathcal{P} \) derived from \textsc{partition},
5. a set of pairs \( \{(p_{i_1}, p_{j_1}), (p_{i_2}, p_{j_2}), \ldots, (p_{i_n}, p_{j_n})\} \) of elements of the partition \( \mathcal{P} \) such that the condition \( \textsc{pairs} = \texttt{dontcare} \lor i_k \textsc{pairs} j_k \) holds for all \( k \in [1, n] \),
6. a \textsc{target} option,

given one of the sets of values \( \mathcal{V}_\alpha, (1 \leq \alpha \leq h) \), a source value \( u \) can be permuted with a target value \( v \) if and only if the following conditions are all satisfied:

\(^{9}\)When \textsc{pairs} is equal to \texttt{dontcare} we just consider all possible pairs.

\(^{10}\)We may have more than one set when the path does not start from a top level collection.
2.1. DESCRIBING THE ARGUMENTS OF A GLOBAL CONSTRAINT

1. \( u \neq v \) (source and target values should be distinct),
2. \( u \in V_\alpha \) (source value, i.e., value that is replaced, should be part of the solution),
3. \( \exists k | u \in p_{ik} \land v \in p_{jk} \) (source and target values should be located in the appropriate partition classes),
4. \( \text{TARGET} = \text{in} \Rightarrow v \in V_\alpha \) (if \( \text{TARGET} = \text{in} \) then the target value should also be part of the solution).

If \( \text{SOURCE} \) is equal to all we replace each occurrence of \( u \) by \( v \), and conversely each occurrence of \( v \) by \( u \). Otherwise we replace at least one occurrence of \( u \) by \( v \).

Without loss of generality, when \( \text{PATH} \) designates a collection of integer values or domain variables, the exchange of tuples of values is defined in a similar way.

We now provide a number of examples of value symmetry and illustrate how to encode them with the five parameters we just introduced. We start from the most common value symmetry, namely exchanging all occurrences of two distinct values or replacing all occurrences of a value by an unused value.

**EXAMPLE 1:** As a first example, consider the \texttt{alldifferent(VARIABLES)} constraint, which enforces all variables of the collection \( \text{VARIABLES} \) to take distinct values. Note that we can exchange two assigned values of \( \text{VARIABLES} \), or replace an assigned value of \( \text{VARIABLES} \) by a new value, i.e., a value that is not yet assigned to any variable of \( \text{VARIABLES} \). Within the electronic catalogue this is represented by the following meta-data, \texttt{vals([\text{VARIABLES.var}], int, \neq, all, dontcare)}, to which corresponds the following textual form:

Two distinct values of \( \text{VARIABLES}.\text{var} \) can be swapped; a value of \( \text{VARIABLES}.\text{var} \) can be renamed to any unused value.

For instance, since \texttt{alldifferent((5, 1, 9, 3))} is a solution, we can replace value 9 by a not yet assigned value, 0 for instance, and get another valid solution \texttt{alldifferent((5, 1, 0, 3))}.

The five parameters of \texttt{vals([\text{VARIABLES.var}], int, \neq, all, dontcare)} have the following meaning:

- \texttt{[VARIABLES.var]} indicates that the modification takes place within the values assigned to the \texttt{var} attribute of the \texttt{VARIABLES} collection.
- \texttt{int} defines the partition of values \( \mathcal{P} = \ldots, \{-1\}, \{0\}, \{1\}, \ldots \).
- \texttt{\neq} indicates that the exchange of values takes place between two distinct elements of \( \mathcal{P} \).
- \texttt{all} specifies that all occurrences of the source value have to be exchanged with all occurrences of the target value.
- \texttt{dontcare} tells that the source value can be replaced by an already existing value or by a new value, i.e., a value not already used in \texttt{VARIABLES.var}.
EXAMPLE 2: As a second example, consider the $n$value($NVAL, VARIABLES$) constraint, which enforces $NVAL$ to be equal to the number of distinct values assigned to the variables of the collection $VARIABLES$. Note that we can exchange all occurrences of two distinct values of $VARIABLES$, or replace all occurrences of an assigned value of $VARIABLES$ by a new value, i.e., a value that is not yet assigned to any variable of $VARIABLES$. Within the electronic catalogue this is represented by the following meta-data, $vals([ VARIABLES.var], int, \neq, all, dontcare)$, to which corresponds the following textual form:

*All occurrences of two distinct values of $VARIABLES$.var can be swapped; all occurrences of a value of $VARIABLES$.var can be renamed to any unused value.*

For instance, since $nvalue(4, \langle 3, 1, 7, 1, 6 \rangle)$ is a solution, we can replace all occurrences of value 1 by a not yet assigned value, 8 for instance, and get another valid solution $nvalue(4, \langle 3, 8, 7, 8, 6 \rangle)$. We can also swap all occurrences of value 1 and value 3, and get another valid solution $nvalue(4, \langle 1, 3, 7, 3, 6 \rangle)$.

The five parameters of $vals([VARIABLES.var], int, \neq, all, dontcare)$ have the following meaning:

- $[VARIABLES.var]$ indicates that the modification takes place within the values assigned to the var attribute of the $VARIABLES$ collection.
- $int$ defines the partition of values $\mathcal{P} = \ldots, \{-1\}, \{0\}, \{1\}, \ldots$.
- $\neq$ indicates that the exchange of values takes place between two distinct elements of $\mathcal{P}$.
- $all$ specifies that all occurrences of the source value have to be exchanged with all occurrences of the target value.
- $dontcare$ tells that the source value can be replaced by an already existing value or by a new value, i.e., a value not already used in $VARIABLES$.var.

We now introduce a third and a fourth example where the meta-data used for describing value symmetry, $vals([VARIABLES.var], int, \neq, all, dontcare)$, is replaced by $vals([VARIABLES.var], int, \neq, all, in)$, i.e., we are not allowed to introduce an unused value.
2.1. DESCRIBING THE ARGUMENTS OF A GLOBAL CONSTRAINT

EXAMPLE 3: As a third example, consider the \texttt{all}_{\text{min}}_{\text{dist}}(\text{MINDIST, VARIABLES}) constraint, which enforces for each pair \((\text{var}_i, \text{var}_j)\) of distinct variables of the collection \text{VARIABLES} that \(|\text{var}_i - \text{var}_j| \geq \text{MINDIST}\). Note that we can exchange two occurrences of distinct values of \text{VARIABLES}, but we cannot replace an existing value \(u\) by a new value \(v\) (since the new value \(v\) may be too close from another existing value \(w\), i.e., \(|v - w| < \text{MINDIST}\)). Within the electronic catalogue this is represented by the following meta-data, \texttt{vals([VARIABLES.var], \text{int}, \neq, \text{all.in})}, to which corresponds the following textual form:

\textit{Two distinct values of VARIABLES.var can be swapped.}

For instance, since \texttt{all}_{\text{min}}_{\text{dist}}(2, \langle 5, 1, 9, 3 \rangle) \) is a solution, we can swap values 5 and 9, and get another valid solution \texttt{all}_{\text{min}}_{\text{dist}}(2, \langle 9, 1, 5, 3 \rangle). The five parameters of \texttt{vals([VARIABLES.var], \text{int}, \neq, \text{all.in})} have the following meaning:

\begin{itemize}
  \item \texttt{[VARIABLES.var]} indicates that the modification takes place within the values assigned to the \texttt{var} attribute of the \texttt{VARIABLES} collection.
  \item \texttt{int} defines the partition of values \(P = \ldots \langle -1 \rangle, \langle 0 \rangle, \langle 1 \rangle, \ldots \)
  \item \texttt{\neq} indicates that the exchange of values takes place between two distinct elements of \(P\).
  \item \texttt{all} specifies that all occurrences of the source value have to be exchanged with all occurrences of the target value.
  \item \texttt{in} tells that the source value has to be replaced by an already existing value in \texttt{VARIABLES.var}.
\end{itemize}

EXAMPLE 4: As a fourth example, consider the \texttt{minimum(MIN, VARIABLES)} constraint, which enforces \texttt{MIN} to be equal to the minimum value of the collection \texttt{VARIABLES}. Note that we can exchange all occurrences of two distinct values of \texttt{VARIABLES}, but we cannot replace an existing value \(u\) by a new value \(v\) (since the new value \(v\) may be smaller than \texttt{MIN}). Within the electronic catalogue this is represented by the following meta-data, \texttt{vals([VARIABLES.var], \text{int}, \neq, \text{all.in})}, to which corresponds the following textual form:

\textit{All occurrences of two distinct values of VARIABLES.var can be swapped.}

For instance, since \texttt{minimum}(2, \langle 3, 2, 7, 2, 6 \rangle) \) is a solution, we can swap values 2 and 6, and get another valid solution \texttt{minimum}(2, \langle 3, 6, 7, 6, 2 \rangle). The five parameters of \texttt{vals([VARIABLES.var], \text{int}, \neq, \text{all.in})} have the following meaning:

\begin{itemize}
  \item \texttt{[VARIABLES.var]} indicates that the modification takes place within the values assigned to the \texttt{var} attribute of the \texttt{VARIABLES} collection.
  \item \texttt{int} defines the partition of values \(P = \ldots \langle -1 \rangle, \langle 0 \rangle, \langle 1 \rangle, \ldots \)
  \item \texttt{\neq} indicates that the exchange of values takes place between two distinct elements of \(P\).
  \item \texttt{all} specifies that all occurrences of the source value have to be exchanged with all occurrences of the target value.
  \item \texttt{in} tells that the source value has to be replaced by an already existing value in \texttt{VARIABLES.var}.
\end{itemize}
CHAPTER 2. DESCRIBING GLOBAL CONSTRAINTS

We now present three examples where, using the partition generator \( \text{comp}(S) \), we consider two classes of values: a first class consisting of elements of \( S \) and a second class of elements of \( \mathbb{Z} \) not in \( S \). The first example corresponds to a value symmetry where values from the same class are exchanged, while the two other examples consider permutation of values between distinct classes with respect to a given class ordering.

**EXAMPLE 5:** As a fifth example, consider the `among(NVAR, VARIABLES, VALUES)` constraint, which enforces \( \text{NVAR} \) to be equal to the number of variables of the collection `VARIABLES` that are assigned a value in `VALUES`. We focus on exchanges of values that take place within `VARIABLES`. Note that, given a value that both occurs in `VARIABLES` and in `VALUES`, we can replace it by any value in `VALUES`. But we can also replace a value that occurs in `VARIABLES`, but not in `VALUES`, by any value not in `VALUES`. Within the electronic catalogue this is represented by the following meta-data, `\( \text{vals}([\text{VARIABLES.var}], \text{comp}(\text{VALUES.val}), =, \text{dontcare}, \text{dontcare}) \)`, to which corresponds the following textual form:

An occurrence of a value of `VARIABLES.var` that belongs to `VALUES.val` (resp. does not belong to `VALUES.val`) can be replaced by any other value in `VALUES.val` (resp. not in `VALUES.val`).

For instance, since `\( \text{among}(3, \{4, 5, 4, 1\}, \{1, 5, 8\}) \)` is a solution, we can swap the first occurrence of value 5 with the second occurrence of value 4 in `VARIABLES.var`, and get another valid solution `\( \text{among}(3, \{4, 4, 5, 5, 1\}, \{1, 5, 8\}) \)`. The five parameters of `\( \text{vals}([\text{VARIABLES.var}], \text{comp}(\text{VALUES.val}), =, \text{dontcare}, \text{dontcare}) \)` have the following meaning:

- `\[\text{VARIABLES.var} \]` indicates that the modification takes place within the values assigned to the `var` attribute of the `VARIABLES` collection.
- `\text{comp}(\text{VALUES.val})` defines two set of values, a first set \( S_1 \) corresponding to all values in `VALUES.val`, and a second set \( S_2 \) corresponding to all values not in `VALUES.val`.
- `=` indicates that the exchange of values takes place within the same set, i.e., within \( S_1 \) or within \( S_2 \).
- `\text{dontcare}` specifies that one occurrence of the source value has to be replaced by the target value.
- `\text{dontcare}` tells that the source value can be replaced by an already existing value or by a new value, i.e., a value not already used in `VARIABLES.var`. 


2.1. DESCRIBING THE ARGUMENTS OF A GLOBAL CONSTRAINT

EXAMPLE 6: As a sixth example, consider the `atleast(N, VARIABLES, VALUE)` constraint, which enforces at least N variables of the collection VARIABLES to be assigned value VALUE. Note that, given an occurrence of value that belongs to VARIABLES that is different from VALUE, we can replace it by any other value that is also different from VALUE. But we can also replace it by value VALUE since this does not decrease the number of variables that are assigned value VALUE. Within the electronic catalogue this is represented by the following meta-data, `vals([VARIABLES.var], comp(VALUE), ≥, dontcare, dontcare)`, to which corresponds the following textual form:

An occurrence of a value of VARIABLES.var that is different from VALUE can be replaced by any other value.

For instance, since `atleast(2, ⟨4, 2, 4, 5, 2⟩, 4)` is a solution, we can replace the second occurrence of value 2 with a value that is different from value 4, e.g., value 8, and get another valid solution `atleast(2, ⟨4, 2, 4, 5, 8⟩, 4)`. We can also replace the second occurrence of value 2 with value 4 and get another valid solution `atleast(2, ⟨4, 2, 4, 5, 4⟩, 4)`. The five parameters of `vals([VARIABLES.var], comp(VALUE), ≥, dontcare, dontcare)` have the following meaning:

- `[VARIABLES.var]` indicates that the modification takes place within the values assigned to the var attribute of the VARIABLES collection.
- `comp(VALUE)` defines two set of values, a first set $S_1$ containing only value VALUE, and a second set $S_2$ corresponding to all values different from VALUE.
- $≥$ indicates that the the source and target values should respectively belong to sets $S_i$ and $S_j$ where $i ≥ j$:
  1. If the source value is different from VALUE (i.e., the source value belongs to $S_2$), then the target value can indifferently be equal or not equal to VALUE (i.e., the target value belongs to $S_1$ or $S_2$).
  2. If the source value is equal to VALUE (i.e., the source value belongs to $S_1$), then the target value is equal to VALUE (i.e., the target value also belongs to $S_1$). But in this case no exchange can take place since the source and target values are identical.
- `dontcare` specifies that one occurrence of the source value has to be replaced by the target value.
- `dontcare` tells that the source value can be replaced by an already existing value or by a new value, i.e., a value not already used in VARIABLES.var.

*Within the collection VARIABLES, this swap does not change the number of variables that are assigned value VALUE.*
EXAMPLE 7: As a seventh example, consider the $\text{atmost}(N, \text{VARIABLES}, \text{VALUE})$ constraint, which enforces at most $N$ variables of the collection $\text{VARIABLES}$ to be assigned value $\text{VALUE}$. Note that, given an occurrence of value that belongs to $\text{VARIABLES}$, and that is different from $\text{VALUE}$, we can replace it by any other value that is also different from $\text{VALUE}$. But we can also replace an occurrence of value $\text{VALUE}$ by a value that is different from $\text{VALUE}$, since this does not increase the number of variables that are assigned value $\text{VALUE}$.

Within the electronic catalogue this is represented by the following meta-data, $\text{vals}([\text{VARIABLES}.\text{var}], \text{comp}(\text{VALUE}), \leq, \text{dontcare}, \text{dontcare})$, to which corresponds the following textual form:

$An occurrence of a value of \text{VARIABLES}.\text{var} can be replaced by any other value that is different from \text{VALUE}.$

For instance, since $\text{atmost}(1, \{4, 2, 4, 5\}, 2)$ is a solution, we can replace the second occurrence of value 4 with a value that is different from value 2, e.g., value 8, and get another valid solution $\text{atmost}(1, \{4, 2, 8, 5\}, 2)$. But, within $\text{atmost}(1, \{4, 2, 4, 5\}, 2)$, we can also replace value 2 with any other value, e.g. value 4 and get another valid solution $\text{atmost}(1, \{4, 4, 4, 5\}, 2)$.

The five parameters of $\text{vals}([\text{VARIABLES}.\text{var}], \text{comp}(\text{VALUE}), \leq, \text{dontcare}, \text{dontcare})$ have the following meaning:

- $[\text{VARIABLES}.\text{var}]$ indicates that the modification takes place within the values assigned to the $\text{var}$ attribute of the $\text{VARIABLES}$ collection.
- $\text{comp}(\text{VALUE})$ defines two set of values, a first set $S_1$ containing only value $\text{VALUE}$, and a second set $S_2$ corresponding to all values different from $\text{VALUE}$.
- $\leq$ indicates that the the source and target values should respectively belong to sets $S_i$ and $S_j$ where $i \leq j$:
  
  1. If the source value is different from $\text{VALUE}$ (i.e., the source value belongs to $S_2$), then the target value is also different from $\text{VALUE}$ (i.e., the target value belongs to $S_2$). This supports the fact that we do not want to increase the number of occurrences of value $\text{VALUE}$.
  2. If the source value is equal to $\text{VALUE}$ (i.e., the source value belongs to $S_1$), then there is no restriction on the target value (i.e., the target value belongs to $S_1$ or to $S_2$). But the set $S_1$ is not relevant since the target value would also be fixed to $\text{VALUE}$, and, in this context, no exchange can take place.
- $\text{dontcare}$ specifies that one occurrence of the source value has to be replaced by the target value.
- $\text{dontcare}$ tells that the source value can be replaced by an already existing value or by a new value, i.e., a value not already used in $\text{VARIABLES}.\text{var}$.

$^4$Within the collection $\text{VARIABLES}$, this swap does not change the number of variables that are assigned value $\text{VALUE}$.

We now illustrate the fact that the scope of value symmetry can sometimes be extended to several collections of variables. For this purpose we consider the $\text{common}$ constraint.
EXAMPLE 8: Consider the \texttt{common(NCOMMON1, NCOMMON2, VARIABLES1, VARIABLES2)} constraint, which enforces the two following conditions:

- \texttt{NCOMMON1} is the number of variables of the collection \texttt{VARIABLES1} taking a value in \texttt{VARIABLES2}.
- \texttt{NCOMMON2} is the number of variables of the collection \texttt{VARIABLES2} taking a value in \texttt{VARIABLES1}.

Note that we can exchange all occurrences of two distinct values of \texttt{VARIABLES1} or \texttt{VARIABLES2}, or replace all occurrences of an assigned value of \texttt{VARIABLES1} or \texttt{VARIABLES2} by a new value, i.e., a value that is not yet assigned to any variable of \texttt{VARIABLES1} and \texttt{VARIABLES2}. Within the electronic catalogue this is represented by the following meta-data, \texttt{vals([VARIABLES1.var, VARIABLES2.var], int, \neq, all, dontcare)}, to which corresponds the following textual form:

\textit{All occurrences of two distinct values in VARIABLES1.var or VARIABLES2.var can be swapped; all occurrences of a value in VARIABLES1.var or VARIABLES2.var can be renamed to any unused value.}

For instance, since \texttt{common(3, 4, (1, 9, 1, 5), (2, 1, 9, 6, 9))} is a solution, we can replace all occurrences of value 1 by a not yet assigned value, 7 for instance, and get another valid solution \texttt{common(3, 4, (7, 9, 7, 5), (2, 7, 9, 9, 6, 9))}.

The five parameters of \texttt{vals([VARIABLES1.var, VARIABLES2.var], int, \neq, all, dontcare)} have the following meaning:

- \texttt{[VARIABLES1.var, VARIABLES2.var]} indicates that the modification takes place within the values assigned to the \texttt{var} attribute of the \texttt{VARIABLES1} and \texttt{VARIABLES2} collections.
- \texttt{int} defines the partition of values \texttt{P} = \ldots, \{-1\}, \{0\}, \{1\}, \ldots.
- \texttt{\neq} indicates that the exchange of values takes place between two distinct elements of \texttt{P}.
- \texttt{all} specifies that all occurrences of the source value have to be exchanged with all occurrences of the target value.
- \texttt{dontcare} tells that the source value can be replaced by an already existing value or by a new value, i.e., a value not already used in \texttt{VARIABLES1} or \texttt{VARIABLES2}.

We now present an example that illustrates the fact that value symmetry can also occur between two arguments that both correspond to a domain variable, i.e., not just between the variables of a collection of variables. For this purpose we consider the \texttt{leq} constraint.
EXAMPLE 9: Consider the \texttt{leq}((VAR1,VAR2)) constraint, which enforces VAR1 to be less than or equal to VAR2. Note that VAR1 can be decreased to any value, and that VAR1 can be increased up to VAR2. Similarly, VAR2 can be increased to any value, and VAR2 can be decreased down to VAR1. Within the electronic catalogue this is respectively represented by the following meta-data, \texttt{vals}([VAR1], int(\leq (VAR2)), \neq, all, dontcare) and \texttt{vals}([VAR2], int(\geq (VAR1)), \neq, all, dontcare), to which corresponds the following textual form:

VAR1 can be replaced by any value \leq VAR2;
VAR2 can be replaced by any value \geq VAR1.

For instance, since \texttt{leq}(2, 9) is a solution, we can replace value 2 by any value less than or equal to 9, e.g. value 5 and get another valid solution \texttt{leq}(5, 9). But, within \texttt{leq}(2, 9), we can also replace value 9 with any other value greater than or equal to 2, e.g. value 4 and get another valid solution \texttt{leq}(2, 4).

The five parameters of \texttt{vals}([VAR1], int(\leq (VAR2)), \neq, all, dontcare) have the following meaning:

- \texttt{[VAR1]} indicates that the modification takes place within the value assigned to the argument \texttt{VAR1} of the constraint \texttt{leq}.
- \texttt{int(\leq (VAR2))} defines the partition of values \texttt{P} = \ldots, \{VAR2 - 2\}, \{VAR2 - 1\}, \{VAR2\} (i.e., we only consider values that are less than or equal to VAR2).
- \texttt{\neq} indicates that the exchange of values takes place between two distinct elements of \texttt{P}.
- \texttt{all} specifies that all occurrences of the source value have to be replaced by the target value. Note that, since the scope of the change is reduced to one single variable, we have one occurrence of the source value and no occurrence of the target value.
- \texttt{dontcare} tells that the source value will be replaced by a new value.

The meta-data \texttt{vals}([VAR2], int(\geq (VAR1)), \neq, all, dontcare) has a similar explanation.

We now present two examples related to the \texttt{k.disjoint} constraint. The first example illustrates the fact that the path specifying the scope of the exchange can contain more than one collection. The second example exemplifies the fact that the path specifying the scope of the exchange does not necessarily start with a top level collection.
EXAMPLE 10: Consider the \texttt{k.disjoint(SETS)} constraint which, given \(|SETS|\) sets of domain variables, enforces that no value is assigned to more than one set. Note that we can swap all the occurrences of two values, or replace all occurrences of a value by a value that is not yet used. Within the electronic catalogue this is represented by the following meta-data, \texttt{vals([SETS.set.var], \texttt{int, \neq, all, dontcare})}, to which corresponds the following textual form:

\emph{All occurrences of two distinct values of \texttt{SETS.set.var} can be swapped; all occurrences of a value of \texttt{SETS.set.var} can be renamed to any unused value.}

For instance, since \texttt{k.disjoint([set – \langle 1, 9, 1, 5 \rangle, set – \langle 7, 2, 7 \rangle])} is a solution, we can replace value 1 by any value that is different from the already used values 2, 5, 7, and 9, e.g. value 3, and get another valid solution \texttt{k.disjoint([set – \langle 3, 9, 3, 5 \rangle, set – \langle 7, 2, 7 \rangle])}. From the solution \texttt{k.disjoint([set – \langle 1, 9, 1, 5 \rangle, set – \langle 7, 2, 7 \rangle])}, we can also swap all occurrences of two values, e.g. values 1 and 2, and get another valid solution \texttt{k.disjoint([set – \langle 2, 9, 2, 5 \rangle, set – \langle 7, 1, 7 \rangle])}.

The five parameters of \texttt{vals([SETS.set.var], \texttt{int, \neq, all, dontcare})} have the following meaning:

- \texttt{[SETS.set.var]} indicates that the modification takes place within the values assigned to the \texttt{var} attribute of the \texttt{SETS.set} collections.
- \texttt{int} defines the partition of values \(P = \ldots, \{-1\}, \{0\}, \{1\}, \ldots\)
- \texttt{\neq} indicates that the exchange of values takes place between two distinct elements of \(P\).
- \texttt{all} specifies that all occurrences of the source value have to be exchanged with all occurrences of the target value.
- \texttt{dontcare} tells that the source value can be replaced by an already existing value or by a new value, i.e., a value not already used in \texttt{SETS.set.var}.\hfill
EXAMPLE 11: Consider the $k_{\text{disjoint}}(\text{SETS})$ constraint which, given $|\text{SETS}|$ sets of domain variables, enforces that no value is assigned to more than one set. Note that, within any set, we can replace any occurrence of a value by another value that is already used in the same set. Within the electronic catalogue this is represented by the following meta-data, $\text{vals}([\text{VARIABLES}.\text{var}], \text{int}, \neq, \text{dontcare}, \text{in})$, to which corresponds the following textual form:

An occurrence of a value of $\text{VARIABLES}.\text{var}$ can be replaced by any value of $\text{VARIABLES}.\text{var}$.

For instance, since $k_{\text{disjoint}}((\text{set} - \langle 1, 9, 1, 5 \rangle, \text{set} - \langle 7, 2, 7 \rangle))$ is a solution, we can replace within the first set the first occurrence of value 1 by the already used value 5, and get another valid solution $k_{\text{disjoint}}((\text{set} - \langle 5, 9, 1, 5 \rangle, \text{set} - \langle 7, 2, 7 \rangle))$.

The five parameters of $\text{vals}([\text{VARIABLES}.\text{var}], \text{int}, \neq, \text{dontcare}, \text{in})$ have the following meaning:

- $[\text{VARIABLES}.\text{var}]$ indicates that the modification takes place within the values assigned to the $\text{var}$ attribute of the $\text{VARIABLES}.\text{var}$ collections. Note that since the corresponding path does not start from a top level collection (i.e., $\text{VARIABLES}$ does not correspond to an argument of the $k_{\text{disjoint}}$ constraint), this represents one set of values for each set: the scope of value symmetry is located within one single set.

- $\text{int}$ defines the partition of values $P = \ldots, \{-1\}, \{0\}, \{1\}, \ldots$.

- $\neq$ indicates that the exchange of values takes place between two distinct elements of $P$.

- $\text{dontcare}$ specifies that one occurrence of the source value has to be replaced by the target value.

- $\text{in}$ tells that the source value has to be replaced by an already existing value in $\text{VARIABLES}.\text{var}$.

We present a last example where the path specifying the scope of the exchange does not end with an attribute but rather with a collection. This can be seen as a generalisation of value symmetry where, instead of exchanging values, we exchange tuples of values. This kind of value symmetry occurs in constraints like $\text{cond}_{\text{lex}}_{\text{cost}}$, $\text{in}_{\text{relation}}$, $\text{npair}$, $n_{\text{vector}}$, $n_{\text{vectors}}$, or $\text{pattern}$. 
2.1. DESCRIBING THE ARGUMENTS OF A GLOBAL CONSTRAINT

**EXAMPLE 12:** Consider the \texttt{nvector(NVEC, VECTORS)} constraint which enforces an equality between \texttt{NVEC} and the number of distinct tuples of values taken by the vectors of the collection \texttt{VECTORS}. Note that we can swap all the occurrences of two tuples of values, or replace all occurrences of a tuple of values by a tuple of values that is not yet used. Within the electronic catalogue this is represented by the following meta-data, \texttt{vals([VECTORS.vec], int, \neq, all, dontcare)}, to which corresponds the following textual form:

\begin{quote}
All occurrences of two distinct tuples of values of \texttt{VECTORS.vec} can be swapped; all occurrences of a tuple of values of \texttt{VECTORS.vec} can be renamed to any unused tuple of values.
\end{quote}

For instance, since \texttt{nvector(2, \langle vec-\langle 5, 6 \rangle, vec-\langle 9, 2 \rangle, vec-\langle 5, 6 \rangle \rangle)} is a solution, we can replace all the occurrences of the tuple of values \texttt{\langle 5, 6 \rangle} by any unused tuple of values, e.g. the tuple of values \texttt{\langle 1, 2 \rangle}, and get another valid solution \texttt{nvector(2, \langle vec-\langle 1, 2 \rangle, vec-\langle 9, 2 \rangle, vec-\langle 1, 2 \rangle \rangle)}.

The five parameters of \texttt{vals([VECTORS.vec], int, \neq, all, dontcare)} have the following meaning:

- \texttt{[VECTORS.vec]} indicates that the modification takes place within the tuples of values assigned to the \texttt{vec} attribute of the \texttt{VECTORS} collections.
- \texttt{int} defines the partition of values \texttt{P = \mathbb{Z}^{[VECTORS]}}.
- \texttt{\neq} indicates that the exchange of tuple of values takes place between two distinct elements of \texttt{P}.
- \texttt{all} specifies that all occurrences of the source tuple of values have to be exchanged with all occurrences of the target tuple of values.
- \texttt{dontcare} tells that the source tuple of values can be replaced by an already existing tuple of values or by a new tuple of values, i.e., a tuple of values not already used in \texttt{VECTORS.vec}.

- **\texttt{translate}**(ATTRIBUTES) denotes the fact that we add a constant to some collection attributes (i.e., we express the fact that solutions are preserved under some specific translation). \texttt{ATTRIBUTES} is a list of terms of the form \texttt{ARG1}, or \texttt{ARG2.attr}, or \texttt{ARG3.attr\_i.attr\_j}, where:

  - \texttt{ARG1} is an argument of the global constraint of type \texttt{domain variable} or \texttt{integer}.
  - \texttt{ARG2} is an argument of the global constraint that corresponds to a collection, and \texttt{attr} is an attribute of \texttt{ARG2} of type \texttt{domain variable} or \texttt{integer}.
  - \texttt{ARG3} is an argument of the global constraint that corresponds to a collection, and \texttt{attr\_i} is an attribute of \texttt{ARG3} of type \texttt{collection}, and \texttt{attr\_j} is an attribute of \texttt{ARG3.attr\_i} of type \texttt{domain variable} or \texttt{integer}.

Its purpose is to define all the elements that have to be simultaneously incremented by one and the same constant.

  - The case corresponding to \texttt{ARG1} is motivated by the fact that we sometimes want to increment an argument that is a domain variable or an integer.
The case corresponding to ARG2 Attr is the standard case where we want to express that we increment attribute attr of all items of a collection that is passed as an argument of the global constraint.

Finally, the last case ARG3 Attr_i, Attr_j corresponds to the fact that we want to increment attribute Attr_j of all items corresponding to ARG3 Attr_i.

We now provide two examples, where the translation is respectively applied on one single attribute and on two attributes of a collection.

**EXAMPLE 1:** Consider the all_min_dist(MINDIST, VARIABLES) constraint which enforces for each pair (var_i, var_j) of distinct variables of the collection VARIABLES that |var_i − var_j| ≥ MINDIST. Note that we can add one and the same constant to all variables of the collection VARIABLES since this does not change the difference between any pair of variables. Within the electronic catalogue this is represented by the following meta-data, translate([VARIABLES.var]), to which corresponds the following textual form:

One and the same constant can be added to the var attribute of all items of VARIABLES.

For instance, since all_min_dist(2, {5, 1, 9, 3}) is a solution, we can add the constant 6 to all items of the collection {5, 1, 9, 3}, and get another valid solution all_min_dist(2, {11, 7, 15, 9}).

**EXAMPLE 2:** Consider the cumulative(TASKS, LIMIT) constraint which enforces that at each point in time, the cumulated height of the set of tasks that overlap that point, does not exceed a given limit. Note that we can add one and the same constant to all origin and end attributes of the different tasks of the TASKS collection. This operation simply shifts the overall schedule by a given constant without affecting the maximum resource consumption. Within the electronic catalogue this is represented by the following meta-data, translate([TASKS.origin, TASKS.end]), to which corresponds the following textual form:

One and the same constant can be added to the origin and end attributes of all items of TASKS.

For instance, since

\[
\begin{pmatrix}
\text{origin} - 1 & \text{duration} - 3 & \text{end} - 4 & \text{height} - 1 \\
\text{origin} - 2 & \text{duration} - 9 & \text{end} - 11 & \text{height} - 2 \\
\text{origin} - 3 & \text{duration} - 10 & \text{end} - 13 & \text{height} - 1, 8 \\
\text{origin} - 6 & \text{duration} - 6 & \text{end} - 12 & \text{height} - 1 \\
\text{origin} - 7 & \text{duration} - 2 & \text{end} - 9 & \text{height} - 3
\end{pmatrix}
\]

is a solution, we can add the constant 2 to all origin and end attributes, and get another valid solution

\[
\begin{pmatrix}
\text{origin} - 3 & \text{duration} - 3 & \text{end} - 6 & \text{height} - 1 \\
\text{origin} - 4 & \text{duration} - 9 & \text{end} - 13 & \text{height} - 2, 8 \\
\text{origin} - 5 & \text{duration} - 10 & \text{end} - 15 & \text{height} - 1 \\
\text{origin} - 8 & \text{duration} - 6 & \text{end} - 14 & \text{height} - 1 \\
\text{origin} - 9 & \text{duration} - 2 & \text{end} - 11 & \text{height} - 3
\end{pmatrix}
\]
We conclude by listing other types of symmetries that we may also consider in the future, namely:

- In the context of graph constraints we can usually relabel the vertices of the corresponding graph. This is for instance the case of the circuit constraint where the index attribute corresponds to the name of a vertex.

- In the context of constraints on a matrix we can have symmetries on both the rows and the columns of the matrix. On the one hand, since a row corresponds to a collection this can be currently expressed. On the other hand, since a column corresponds to all the $i^{th}$ items of the collections corresponding to the rows, this currently cannot be expressed.

- Given a collection of items, we want to express a symmetry on different subsets of items: more precisely, on all items for which a given attribute is assigned the same value. As an illustrative example consider the cumulatives constraint. We would like to express the possibility of translating the origin of all tasks that are assigned the same machine.

- Given a collection of items we can sometimes multiply by $-1$ all occurrences of one of its attributes. This usually corresponds to a mirror symmetry. This is for instance the case for the origin attribute of the cumulative constraint.

2.2 Describing global constraints in terms of graph properties

Through a practical example, we first present in a simplified form the basic principles used for describing the meaning of global constraints in terms of graph properties. We then give the full details about the different features used in the description process.

2.2.1 Basic ideas and illustrative example

Within the graph-based representation, a global constraint is represented as a digraph where each vertex corresponds to a variable and each arc to a binary arc constraint between the variables associated with the extremities of the corresponding arc. The main difference with classical constraint networks [122], stems from the fact that we do not force any more all arc constraints to hold. We rather consider this graph from which we discard all the arc constraints that do not hold as well as all isolated vertices (i.e, vertices not involved any more in any arc) and impose one or several graph properties on this remaining graph. These properties can for instance be a restriction on the number of connected components, on the size of the smallest connected component or on the size of the largest connected component.
EXAMPLE: We give an example of interpretation of such graph properties in terms of global constraints. For this purpose we consider the sequence $s$ of values $1 3 1 2 8 2 3 6 8 8 3$ from which we construct the following graph $G$:

- To each value associated with a position in $s$ corresponds a vertex of $G$.
- There is an arc from a vertex $v_1$ to a vertex $v_2$ if these vertices correspond to the same value.

Figure 2.1 depicts graph $G$. Since $G$ is symmetric, we omit the directions of the arcs. We have the following correspondence between graph properties and constraints on the sequence $s$:

- The number of connected components of $G$ corresponds to the number of distinct values of $s$.
- The size of the smallest connected component of $G$ is the smallest number of occurrences of the same value in $s$.
- The size of the largest connected component of $G$ is the largest number of occurrences of the same value in $s$.

As a result, in this context, putting a restriction on the number of connected components of $G$ can be seen as a global constraint on the number of distinct values of a sequence of variables. Similar global constraints can be associated with the two other graph properties.

We now explain how to generate the initial graph associated with a global constraint. A global constraint has one or more arguments, which usually correspond to an integer value, to one variable or to a collection of variables. Therefore we have to describe the process that allows for generating the vertices and the arcs of the initial graph from the arguments of a global constraint under consideration. For this purpose we will take a concrete example.

Consider the constraint $\text{nvalue}(\text{NVAL}, \text{VARIABLES})$ where $\text{NVAL}$ and $\text{VARIABLES}$ respectively correspond to a domain variable and to a collection of domain variables $\langle \text{var} - V_1, \text{var} - V_2, \ldots, \text{var} - V_m \rangle$. This constraint holds if $\text{NVAL}$ is equal to the number of distinct values assigned to the variables $V_1, V_2, \ldots, V_m$. We first show how to generate the initial graph associated with the $\text{nvalue}$ constraint. We then describe the arc constraint associated with each arc of this graph. Finally, we give the graph

\[ \text{var} \] corresponds to the name of the attribute used in the collection of variables.
property we impose on the final graph.

To each variable of the collection VARIABLES corresponds a vertex of the initial graph. We generate an arc between each pair of vertices. To each arc, we associate an equality constraint between the variables corresponding to the extremities of that arc. We impose that \( NVAL \), the variable corresponding to the first argument of \( nvalue \), be equal to the number of strongly connected components of the final graph. This final graph consists of the initial graph from which we discard all arcs such that the corresponding equality constraint does not hold.

Part (A) of Figure 2.2 shows the graph initially generated for the constraint \( nvalue (NVAL, \langle var - V_1, var - V_2, var - V_3, var - V_4 \rangle) \), where \( NVAL, V_1, V_2, V_3 \) and \( V_4 \) are domain variables. Part (B) presents the final graph associated with the ground instance \( nvalue(3, \langle var - 5, var - 5, var - 1, var - 8 \rangle) \). For each vertex of the initial and final graph we respectively indicate the corresponding variable and the value assigned to that variable. We have removed from the final graph all the arcs associated with equalities that do not hold. The constraint \( nvalue(3, \langle var - 5, var - 5, var - 1, var - 8 \rangle) \) holds since the final graph contains three strongly connected components, which in the context of the definition of the \( nvalue \) constraint, can be reinterpreted as the fact that \( NVAL \) is the number of distinct values assigned to variables \( V_1, V_2, V_3, V_4 \).

![Figure 2.2: Initial and final graph associated with nvalue](image)

Now that we have illustrated the basic ideas for describing a global constraint in terms of graph properties, we go into more details.

### 2.2.2 Ingredients used for describing global constraints

We first introduce the basic ingredients used for describing a global constraint and illustrate them shortly on the example of the \( nvalue \) constraint introduced in the previous section on page 40. We then go through each basic ingredient in more detail. The graph-based description is founded on the following basic ingredients:

- **Data types and restrictions** used in order to describe the arguments of a global constraint. Data types and restrictions were already described in the previous section (from page 6 to page 18).

- **Collection generators** used in order to derive new collections from the arguments of a global constraint for one of the following reasons:
– Collection generators are sometimes required since the initial graph of a global constraint cannot always be directly generated from the arguments of the global constraint. The \texttt{nvalue(NVAL, VARIABLES)} constraint did not require any collection generator since the vertices of its initial graph were directly generated from the \texttt{VARIABLES} collection.

– A second use of collection generators is for deriving a collection of items for different sets of vertices of the final graph. This is sometimes required when we use set generators (see the last item of the enumeration).

- \textbf{Elementary constraints} associated with the arcs of the initial and final graph of a global constraint. The \texttt{nvalue} constraint was using an \textit{equality} constraint, but other constraints are usually required.

- \textbf{Graph generators} employed for constructing the initial graph of a global constraint. In the context of the \texttt{nvalue} constraint the initial graph was a \textit{clique}. As we will see later, other patterns are needed for generating an initial graph.

- \textbf{Graph properties} and \textit{graph classes} used for constraining the final graph we want to obtain. In the context of the \texttt{nvalue} constraint we were using the \textit{number of strongly connected components} for counting the number of distinct values.

- \textbf{Set generators} that may be used for generating specific sets of vertices of the final graph on which we want to enforce a given constraint. Since the \texttt{nvalue} constraint enforces a graph property on the final graph (and not on subparts of the final graph) we did not use this feature.

We first start to explain each ingredient separately and then show how one can describe most global constraints in terms of these basic ingredients.

\textbf{Collection generators}

The vertices of the initial graph are usually directly generated from collections of items that are arguments of the global constraint \texttt{G} under consideration. However, it sometimes happens that we would like to derive a new collection from existing arguments of \texttt{G} in order to produce the vertices of the initial graph.

\textbf{EXAMPLE:} This is for instance the case of the \texttt{element(INDEX, TABLE, VALUE)} constraint, where \texttt{INDEX} and \texttt{VALUE} are domain variables that we would like to group as a single item \texttt{I} (with two attributes) of a new derived collection. This is in fact done in order to generate the following initial graph:

- The item \texttt{I} as well as all items of \texttt{TABLE} constitute the vertices,
- There is an arc from \texttt{I} to each item of the \texttt{TABLE} collection.

We provide the following mechanism for deriving new collections:

- In a first phase we declare the name of the new collection as well as the names of its attributes and their respective types. This is achieved exactly in the same way as those collections that are used in the arguments of a global constraint (see page 8).
2.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES

EXAMPLE: Consider again the example of the `element(INDEX, TABLE, VALUE)` constraint. The declaration `ITEM = collection(index = dvar, value = dvar)` introduces a new collection called `ITEM` where each item has an index and a value attribute. Both attributes correspond to domain variables.

- In a second phase we give a list of patterns that are used for generating the items of the new collection. A pattern `o - item(a_1 - v_1, a_2 - v_2, ..., a_n - v_n)` or `item(a_1 - v_1, a_2 - v_2, ..., a_n - v_n)` specifies for each attribute `a_i (1 \leq i \leq n)` of the new collection how to fill it.\(^{12}\) This is done by providing for each attribute `a_i` one of the following expression `v_i`:
  - A constant.
  - An argument of the global constraint `G`.
  - An expression `c.a`, where `a` is an attribute of a collection `c`, such that `c` is an argument of the global constraint `G` or a derived collection that was previously declared. An expression of this form is called a direct reference to an attribute of a collection.
  - An expression `c_1.c_2.a`, where `a` is an attribute of a collection `c_2`, and `c_2` is an attribute of a collection `c_1` such that `c_1` is an argument of the global constraint `G` or a derived collection that was previously declared. An expression of this form is called an indirect reference to an attribute of a collection.

This expression `v_i` must be compatible with the type declaration of the corresponding attribute of the new collection.

EXAMPLE: We continue the example of the `element(INDEX, TABLE, VALUE)` constraint and the derived collection `ITEM = collection(index = dvar, value = dvar)`. The pattern `item(index = INDEX, value = VALUE)` indicates that:

- The index attribute of the `ITEM` collection will be generated by using the `INDEX` argument of the `element` constraint. Since `INDEX` is a domain variable, it is compatible with the declaration `ITEM = collection(index = dvar, value = dvar)` of the new collection.
- The value attribute of the `ITEM` collection will be generated by using the `VALUE` argument of the `element` constraint. `VALUE` is also compatible with the declaration statement of the new collection.

We now describe how we use the pattern for generating the items of a derived collection. We have the following two cases:

- If the pattern `o - item(a_1 - v_1, a_2 - v_2, ..., a_n - v_n)` does not contain any direct or indirect reference to an attribute of a collection then we generate one single item for such pattern.\(^{13}\) In this context the value `v_i` of the attribute `a_i (1 \leq i \leq n)` corresponds to a constant, to an argument of the global constraint or to a new derived collection.

\(^{12}\) `o` is one of the comparison operators `=, \neq, <, \geq, >, \leq`. When omitted its default value is `=`.

\(^{13}\) In this first case the value of `o` is irrelevant.
• If the pattern \( o - \text{item}(a_1 - v_1, a_2 - v_2, \ldots, a_n - v_n) \), where \( o \) is one of the comparison operators \( =, \neq, <, \geq, >, \leq \), contains one or several direct or indirect references to an attribute of a collection\(^{14}\) we denote by:

\[
- \mathcal{D} \text{ the set of indices of the positions corresponding to a direct reference to an attribute of a collection within item}(a_1 - v_1, a_2 - v_2, \ldots, a_n - v_n).
- \mathcal{I} \text{ the set of indices of the positions corresponding to an indirect reference to an attribute of a collection within item}(a_1 - v_1, a_2 - v_2, \ldots, a_n - v_n).
\]

In this context, let \( c_{\alpha_1}, c_{\alpha_2}, \ldots, c_{\alpha_m} \) and \( a_{\alpha_1}, a_{\alpha_2}, \ldots, a_{\alpha_m} \) respectively denote the corresponding collections and attributes.

- The pattern used for creating an item of the new collection,
- The pattern used for creating an item of the new collection,

We illustrate this generation process on a set of examples. Each example is described by providing:

- The global constraint and its arguments,
- The declaration of the new derived collection,
- The pattern used for creating an item of the new collection,
- The items generated by applying this pattern to the global constraint,
- A comment about the generation process.

We first start with four examples that do not mention any references to an attribute of a collection. A box surrounds an argument of a global constraint that is mentioned in a generated item.

\(^{14}\)This collection is an argument of the global constraint or corresponds to a newly derived collection.
2.2. **Describing Global Constraints in Terms of Graph Properties**

**Example**

<table>
<thead>
<tr>
<th>Constraint</th>
<th>description</th>
</tr>
</thead>
</table>
| \( \text{element}([\text{INDEX} \ \text{TABLE} \ \text{VALUE}]) \) | \*We generate one single item where the two attributes \text{INDEX} and \text{VALUE} respectively take the first argument \text{INDEX} and the third argument \text{VALUE} of the \text{element} constraint.\

<table>
<thead>
<tr>
<th>Example Constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{lex}_{\leq} \text{(VECTOR1, VECTOR2)} )</td>
<td>*We generate one single item where the three attributes \text{index}, \text{x}, and \text{y} take value 0.**</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example Constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{in}_{\text{relation}}([\text{VARIABLES} \ \text{TUPLES_OF_VALS}) )</td>
<td>*We generate one single item where the unique attribute \text{vec} takes the first argument of the \text{in}_{\text{relation}} constraint as its value.**</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example Constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{domain}_{\text{constraint}}([\text{VAR} \ \text{VALUES}) )</td>
<td>*We generate one single item where the two attributes \text{var01} and \text{value} respectively take value 1 and the first argument of the \text{domain}_{\text{constraint}} constraint.**</td>
</tr>
</tbody>
</table>

We continue with three examples that mention one or several direct references to an attribute of some collections. We now need to explicitly give the items of these collections in order to generate the items of the derived collection.
EXAMPLE

CONSTRAINT : `lex_lesseq(VECTOR1, VECTOR2)`

VECTOR1 : `(var−5, var−2, var−3, var−1)`

VECTOR2 : `(var−5, var−2, var−6, var−2)`

DERIVED COLLECTION: COMPONENTS — collection(index − int,
  x − dvar, y − dvar)

PATTERN(S) : `item(index − VECTOR1.key`,
  x − VECTOR1.var, `y − VECTOR2.var)`

GENERATED ITEM(S) : `(index − 1 x − 5 y − 5, index − 2 x − 2 y − 2, index − 3 x − 3 y − 6, index − 4 x − 1 y − 2)`

The pattern mentions three references `VECTOR1.key`, `VECTOR1.var` and `VECTOR2.var` to the collections `VECTOR1` and `VECTOR2` used in the arguments of the `lex_lesseq` constraint. ∀i₁ ∈ [1, |VECTOR1|], ∀i₂ ∈ [1, |VECTOR2|] such that i₁ = i₂ we generate an item `index − v₁ x − v₂ y − v₃` where:

`v₁ = i₁`, `v₂ = VECTOR1[i₁].var`, `v₃ = VECTOR2[i₁].var`.

This leads to the four items listed in the GENERATED ITEM(S) field.

---

¹As defined in Section 2.1.2 on page 8, `key` is an implicit attribute corresponding to the position of an item within a collection.

²We use an equality since this is the default value of the comparison operator `o` when we do not use a pattern of the form `o − item(...)`. 
2.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES

**EXAMPLE**

**CONSTRAINT** : `cumulatives(TASKS, MACHINES, CTR)`

**TASKS** :
- (machine - 1 origin - 1 duration - 4 end - 5 height - 1,
  machine - 1 origin - 4 duration - 2 end - 6 height - 3,
  machine - 1 origin - 2 duration - 3 end - 5 height - 2,
  machine - 2 origin - 5 duration - 2 end - 7 height - 2)

**DERIVED COLLECTION** : `TIME_POINTS` - collection(idm - int,
  duration - dvar, point - dvar)

**PATTERN(S)** :
- item(idm - TASKS.machine,
  duration - TASKS.duration, point - TASKS.origin)
- item(idm - TASKS.machine,
  duration - TASKS.duration, point - TASKS.end)

**GENERATED ITEM(S)** :
- (idm - 1 duration - 4 point - 1,
  idm - 1 duration - 2 point - 4,
  idm - 1 duration - 3 point - 2,
  idm - 2 duration - 2 point - 5,
  idm - 1 duration - 4 point - 5,
  idm - 1 duration - 2 point - 6,
  idm - 1 duration - 3 point - 5,
  idm - 2 duration - 2 point - 7)

The two patterns mention the references TASKS.machine, TASKS.duration, TASKS.origin and TASKS.end of the TASKS collection used in the arguments of the `cumulatives` constraint. \( \forall i \in [1,|\text{TASKS}|] \), we generate two items:

\[ \text{idm} - u_1 \text{ duration} - u_2 \text{ point} - u_3 , \ \text{idm} - v_1 \text{ duration} - v_2 \text{ point} - v_3 \]

where:

\[ u_1 = \text{TASKS}[i].\text{machine}, \ u_2 = \text{TASKS}[i].\text{duration}, \ u_3 = \text{TASKS}[i].\text{origin}, \ v_1 = \text{TASKS}[i].\text{machine}, \ v_2 = \text{TASKS}[i].\text{duration}, \ v_3 = \text{TASKS}[i].\text{end} \]

This leads to the eight items listed in the `GENERATED ITEM(S)` field.

**EXAMPLE**

**CONSTRAINT** : `golomb(VARIABLES)`

**VARIABLES** :
- (var - 0, var - 1, var - 4, var - 6)

**DERIVED COLLECTION** : `PAIRS` - collection(x - dvar, y - dvar)

**PATTERN(S)** :
- `item(x - VARIABLES.var, y - VARIABLES.var)`

**GENERATED ITEM(S)** :
- \( (x - 1 \ y - 0, \ x - 4 \ y - 0, \ x - 4 \ y - 1, \ x - 6 \ y - 0, \ x - 6 \ y - 1, \ x - 6 \ y - 4) \)

The pattern mentions two references VARIABLES.var and VARIABLES.var to the VARIABLES collection used in the arguments of the `golomb` constraint. \( \forall i_1 \in [1,|\text{VARIABLES}|], \forall i_2 \in [1,|\text{VARIABLES}|] \) such that \( i_1 > i_2 \) we generate the item:

\[ u_1 = \text{VARIABLES}[i_1].\text{var}, \ u_2 = \text{VARIABLES}[i_2].\text{var} \]

This leads to the six items listed in the `GENERATED ITEM(S)` field.

*We use the comparison operator > since we have a pattern of the form \( > \text{item}(\ldots) \).*
CHAPTER 2. DESCRIBING GLOBAL CONSTRAINTS

We finish with an example that mentions an indirect reference to an attribute of a collection.

**EXAMPLE**

**CONSTRAINT**: \(\text{cumulative}\_\text{convex}(\text{TASKS}, \text{LIMIT})\)

**TASKS**: \(\langle \text{points} - \langle \text{var} - 2, \text{var} - 1, \text{var} - 5 \rangle \text{height} - 1, \langle \text{points} - \langle \text{var} - 4, \text{var} - 5, \text{var} - 7 \rangle \text{height} - 2, \langle \text{points} - \langle \text{var} - 14, \text{var} - 15 \rangle \text{height} - 2 \rangle\)\)

**DERIVED COLLECTION**: \(\text{INSTANTS} - \text{collection(}\text{instant} - \text{int}\)\)

**PATTERN**(S): \(\text{item(}\text{instant} - \text{TASKS.points.var}\)\)

**GENERATED ITEM**(S): \(\langle \text{instant} - 2, \text{instant} - 1, \text{instant} - 5, \text{instant} - 4, \text{instant} - 5, \text{instant} - 7, \text{instant} - 14, \text{instant} - 15 \rangle\)

The pattern mentions the indirect reference \(\text{TASKS.points.var}\) of the \(\text{TASKS}\) collection used in the arguments of the \(\text{cumulative}\_\text{convex}\) constraint. \(\forall i \in [1, |\text{TASKS}|], \forall j \in [1, |\text{TASKS}[i].\text{points}|]\) we generate the item \(\text{instant} - u_{ij}\) where:

\[u_{ij} = \text{TASKS}[i].\text{points}[j]\]

This leads to the eight items listed in the \(\text{GENERATED ITEM}(S)\) field.

**Elementary constraints attached to the arcs**

This section describes the constraints that are associated with the arcs of the initial graph of a global constraint. These constraints are called *arc constraints*. To each arc one can associate one or several arc constraints. An arc will belong to the final graph if and only if all its arc constraints hold. An arc constraint from a vertex \(v_1\) to a vertex \(v_2\) mentions variables and/or values associated with \(v_1\) and \(v_2\). Before defining an *arc constraint*, we first need to introduce *simple arithmetic expressions* as well as *arithmetic expressions*. Simple arithmetic expressions and arithmetic expressions are defined recursively.

**Simple arithmetic expressions**  A *simple arithmetic expression* is defined by one of the following expressions.

- \(I\): \(I\) is an integer.
- \(\text{Arg}\): \(\text{Arg}\) is an argument of the global constraint of type \(\text{int}\) or \(\text{dvar}\).
- \(\text{Arg}\): \(\text{Arg}\) is a formal parameter provided by the arc generator\(^{15}\) of the graph-constraint.
- \(\text{Col.Attr}\): \(\text{Col}\) is a formal parameter provided by the arc generator or the collection used in the *For all items* of iterator.\(^{16}\) \(\text{Attr}\) is an attribute of the collection referenced by \(\text{Col}\).

---

\(^{15}\)Arc generators are described in Section 2.2.2 on page 52.

\(^{16}\)The *For all items* of iterator is described in Section 2.2.3 on page 70.
2.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES

**EXAMPLE:** As an example consider the first graph-constraint associated with the `global_cardinality_with_costs(VARIABLES, VALUES, MATRIX, COST)` constraint and its arc constraint variables.var = VALUES.val. Both, variables.var as well as VALUES.val are simple arithmetic expressions of the form Col[Attr):

- In variables.var, variables corresponds to the formal parameter provided by the arc generator `SELF -> collection(variables)`, while var is an attribute of the VARIABLES collection.
- In VALUES.val, VALUES corresponds to the collection denoted by the `For all items of iterator`, while val is an attribute of the VALUES collection.

**COL[EXP]ATTR:** Col is an argument of type collection, Attr one attribute of Col and Exp an arithmetic expression. Col[Exp].Attr denotes the value of attribute Attr of the Expr<sup>_th_</sup> item of the collection denoted by Col.

**EXAMPLE:** As an example consider the `global_cardinality_with_costs(VARIABLES, VALUES, MATRIX, COST)` constraint and its second graph-constraint, which defines the COST variable. The expression `MATRIX[[variables.key - 1] * |VALUES| + values.key].c` is a simple arithmetic expression of the form Col[Exp].Attr:

- MATRIX is a collection of items `collection(i - int, j - int, c - int)` where all items are sorted in increasing order on attributes i, j (because of the restriction `increasing_seq(MATRIX, [i, j]))`).
- MATRIX[[variables.key - 1] * |VALUES| + values.key].c denotes the value of attribute c of an item of the MATRIX collection. The position of this item within the MATRIX collection depends on the position of a variable of the VARIABLES collection<sup>_a_</sup> as well as on the position of a value of the VALUES collection.<sup>_b_</sup>

<sup>_a_</sup>This position is denoted by the expression variables.key. As defined in Section 2.1.2 on page 8, key is an implicit attribute corresponding to the position of an item within a collection.

<sup>_b_</sup>This position is denoted by the expression values.key.

**Arithmetic expressions** An arithmetic expression is recursively defined by one of the following expressions:

- A simple arithmetic expression.
- Exp<sub>1</sub> Op Exp<sub>2</sub>
  - Exp<sub>1</sub> is an arithmetic expression,
  - Op is one of the following symbols `+`, `-`, `*`/<sup>17</sup>,
  - Exp<sub>2</sub> is an arithmetic expression.
- |Collection|
  - Collection is an argument of type collection and |Collection| denotes the number of items of that collection.

<sup>17</sup>/ denotes an integer division, a division in which the fractional part is discarded.
CHAPTER 2. DESCRIBING GLOBAL CONSTRAINTS

- \(|\text{Exp}|\) – \(\text{Exp}\) is an arithmetic expression, and \(|\text{Exp}|\) denotes the absolute value of this expression.

- \(\text{sign}(\text{Exp})\) – \(\text{Exp}\) is an arithmetic expression, and \(\text{sign}(\text{Exp})\) the sign of \(\text{Exp}\) (\(-1\) if \(\text{Exp}\) is negative, \(0\) if \(\text{Exp}\) is equal to \(0\), \(1\) if \(\text{Exp}\) is positive).

**EXAMPLE:** An example of use of \(\text{sign}\) can be found in the last part of the arc constraint of the crossing constraint:

\[
\text{sign}((s2.ox - s1.ex)*(s1.ey - s1.oy) - (s1.ex - s1.ox)*(s2.oy - s1.ey)) \\
\text{sign}((s2.ex - s1.ex)*(s2.oy - s1.oy) - (s2.ox - s1.ox)*(s2.ey - s1.ey))
\]

- \(\text{card_set(\text{Set})}\) – \(\text{Set}\) is a reference to a set of integers or to a set variable. \(\text{card_set(\text{Set})}\) denotes the number of elements of that set.

**EXAMPLE:** An example of use of \(\text{card_set}\) can be found in the symmetric gcc constraint: \(\text{vars.nocc} = \text{card_set(vars.var)}\).

- \(\text{SimpleExp}_1 \mod \text{SimpleExp}_2\), \(\min(\text{SimpleExp}_1, \text{SimpleExp}_2)\) or \(\max(\text{SimpleExp}_1, \text{SimpleExp}_2)\)

  - \(\text{SimpleExp}_1\) is a simple arithmetic expression.
  - \(\text{SimpleExp}_2\) is a simple arithmetic expression.

**Arc constraints** Now that we have introduced simple arithmetic expressions as well as arithmetic expressions we define an arc constraint. An arc constraint is recursively defined by one of the following expressions:

- \(\text{TRUE}\)
  This stands for an arc constraint that always holds. As a result, the corresponding arc always belongs to the final graph.

**EXAMPLE:** An example of use of \(\text{TRUE}\) can be found in the \text{sum ctr} constraint: \text{sum ctr(VARIABLES, CTR, VAR)} constraint, where it is used in order to enforce keeping all items of the VARIABLES collection in the final graph.

- \(\text{Exp}_1 \text{ Comparison } \text{Exp}_2\)
  - \(\text{Exp}_1\) is an arithmetic expression.
  - \(\text{Comparison}\) is one of the comparison operators \(\leq, \geq, <, >, =, \neq\).
  - \(\text{Exp}_2\) is an arithmetic expression.
2.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES

**EXAMPLE:** As an example of such arc constraint, the second graph-constraint of the `cumulative(TASKS, LIMIT)` constraint uses the following arc constraints:

- $\text{tasks1.duration} > 0$.
- $\text{tasks2.origin} \leq \text{tasks1.origin}$.
- $\text{tasks1.origin} < \text{tasks2.end}$.

The conjunction of these three arc constraints can be interpreted in the following way: an arc from a task $\text{tasks1}$ to a task $\text{tasks2}$ will belong to the final graph if and only if $\text{tasks2}$ overlaps the origin of $\text{tasks1}$.

- **Exp**$_1$ **SimpleCtr** **Exp**$_2$
  - **Exp**$_1$ is an *arithmetic expression*.
  - **SimpleCtr** is an argument of type *atom* that can only take one of the values $\leq, \geq, <, >, =, \neq$.
  - **Exp**$_2$ is an *arithmetic expression*.

**EXAMPLE:** An example of use of such an arc constraint can be found in the `change(NCHANGE, VARIABLES, CTR)` constraint: $\text{variables1.var} \neq \text{CTR} \text{variables2.var}$. Within this expression, $\text{variables1}$ and $\text{variables2}$ correspond to consecutive items of the VARIABLES collection.

- **Exp**$_1$ ¬**SimpleCtr** **Exp**$_2$
  - **Exp**$_1$ is an *arithmetic expression*.
  - **SimpleCtr** is an argument of type *atom* that can only take one of the values $\leq, \geq, <, >, =, \neq$.
  - **Exp**$_2$ is an *arithmetic expression*.

**EXAMPLE:** An example of use of such an arc constraint can be found in the `change_continuity(NB Period, Change, NB Period, Continuity, MIN Size, Change, MAX Size, Change, MIN Size, Continuity, MAX Size, Continuity, VARIABLES, CTR)` constraint: $\text{variables1.var} \neq \text{CTR} \text{variables2.var}$. Within this expression, $\text{variables1}$ and $\text{variables2}$ correspond to consecutive items of the VARIABLES collection.

- **constraint**($\text{Exp}_1, \ldots, \text{Exp}_n$)
  - **constraint** is a global constraint defined in the catalogue for which there exists a graph-based and/or an automaton-based representation.
  - $\text{Exp}_1, \ldots, \text{Exp}_n$ correspond to the arguments of the global constraint **constraint**. Each argument should be a *simple arithmetic expression* that is compatible with the type declaration of the argument of **constraint**.
EXAMPLE: An example of such arc constraint can be found in the definition of `diffn`: `diffn(ORTHOTOPES)` uses the `two_orth_do_not_overlap(ORTHOTYPE1, ORTHOTYPE2)` global constraint for defining its arc constraint. Since `ORTHOTOPES` is a collection of type `collection(ori − dvar, siz − dvar, end − dvar)` and since both `ORTHOTYPE1` and `ORTHOTYPE2` correspond to items of `ORTHOTOPES` there is no type compatibility problem between the call to `two_orth_do_not_overlap` and its definition.

- **ArcCtr1 LogicalConnector ArcCtr2**
  - `ArcCtr1` is an arc constraint,
  - `LogicalConnector` is one of the logical connectors `∨, ∧, ⇒, ⇔`,
  - `ArcCtr2` is an arc constraint.

EXAMPLE: As shown by the following example, `minimum(MIN, VARIABLES)` uses this kind of arc constraint: `variables1 = variables2 ∨ variables1.var < variables2.var`, where `variables1` and `variables2` correspond to items of the `VARIABLES` collection, holds if and only if one of the following conditions holds:
  - `variables1` and `variables2` correspond to the same item of the `VARIABLES` collection,
  - The `var` attribute of `variables1` is strictly less than the `var` attribute of `variables2`.

**Graph generators**

This section describes how to generate the initial graph associated with a global constraint. Initial graphs correspond to directed hypergraphs [54], which have a very regular structure. They are defined in the following way:

- The vertices of the directed hypergraph are generated from collections of items such that each item corresponds to one vertex of the directed hypergraph. These collections are either collections that arise as arguments of the global constraint, or collections that are derived from one or several arguments of the global constraint. In this latter case these **derived collections** are computed by using the **collection generators** previously introduced (see Section 2.2.2 on page 42).

- To all arcs of the directed hypergraph corresponds the same arc constraint that involves vertices in a given order.\(^\text{18}\) These arc constraints, which are mainly unary and binary constraints, were described in the previous section (see Section 2.2.2 on page 48). We describe all the arcs of an initial graph with a set of predefined **arc generators**, which correspond to classical regular structures one can find in the graph literature [369, pages 140–153]. An arc generator of arity \(a\) takes \(n\) collections of items, denoted \(c_i(1 \leq i \leq n)\), as input and returns the corresponding hypergraph where the vertices are the items of the input collections.

\(^\text{18}\) Usually the edges of a hypergraph are not oriented [54, pages 1–2]. However for our purpose we need to define an order on the vertices of an edge since the corresponding arc constraint takes its arguments in a given order.
2.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES

$c_i(1 \leq i \leq n)$ and where all arcs involve $a$ vertices. Specific arc generators allow for giving an $a$-ary constraint for which $a$ is not fixed, which means that the corresponding hypergraph contains arcs involving various number of vertices.

Each arc generator has a name and takes one or several collections of items as input and generates a set of arcs. Each arc is made from a sequence of items $i_1 i_2 \ldots i_a$ and is denoted by $(i_1, i_2, \ldots, i_a)$. $a$ is called the *arity* of the arc generator. We have the following types of arc generators:

- **Arc generators with a fixed predefined arity.** In fact most arc generators have a fixed predefined arity of 2. The graphs they generate correspond to digraphs.
- **Arc generators that can be used with any arity $a$ greater than or equal to 1.** These arc generators generate directed hypergraphs where all arcs consist of $a$ items.
- **Arc generators that generate arcs that do not involve the same number of items.**

We now give the list of arc generators, listed in alphabetic order, and the arcs they generate. For each arc generator we point to a global constraint where it is used in practice. Finally, Figure 2.4 illustrates the different arc generators. At present the following arc generators are in use:

- **CHAIN** has a predefined arity of 2. It takes one collection $c$ and generates the following arcs:\n
  \[ \forall i \in [1, |c| - 1]: (c[i], c[i + 1]), \quad \forall i \in [1, |c| - 1]: (c[i + 1], c[i]). \]

  **EXAMPLE:** The arc generator **CHAIN** is for instance used in the `group_skip_isolated_item` constraint.

- **CIRCUIT** has a predefined arity of 2. It takes one collection $c$ and generates the following arcs:

  \[ \forall i \in [1, |c| - 1]: (c[i], c[i + 1]), \quad (c[|c|], c[1]). \]

  **EXAMPLE:** The arc generator **CIRCUIT** is for instance used in the `circular_change` constraint.

- **CLIQUE** can be used with any arity $a$ greater than or equal to 2. It takes one collection $c$ and generates the arcs: $\forall i_1 \in [1, |c|], \forall i_2 \in [1, |c|], \ldots, \forall i_a \in [1, |c|] : (c[i_1], c[i_2], \ldots, c[i_a])$.

  **EXAMPLE:** The arc generator **CLIQUE** is usually used with an arity $a = 2$. This is for instance the case of the `alldifferent` constraint.

---

19 As defined in Section 2.1.2 on page 8 we use the following notation: for a given collection $c$, $|c|$ and $c[i]$ respectively denote the number of items of $c$ and the $i^{th}$ item of $c$. 
CHAPTER 2. DESCRIBING GLOBAL CONSTRAINTS

• **CLIQUE**(*Comparison*), where *Comparison* is one of the comparison operators \( \leq, \geq, <, >, =, \neq \), can be used with any arity \( a \) greater than or equal to 2. It takes one collection \( c \) and generates the arcs:

\[
\forall i_1 \in [1, |c|], \\
\forall i_2 \in [1, |c|] \text{ such that } i_1 \text{ Comparison } i_2, \\
\forall i_a \in [1, |c|] \text{ such that } i_{a-1} \text{ Comparison } i_a : (c[i_1], c[i_2], \ldots, c[i_a]).
\]

**EXAMPLE:** The orchard(TREES) constraint is an example of constraint that uses the **CLIQUE\(<\)** arc generator with an arity \( a = 3 \). It generates an arc for each set of three trees.

• **CYCLE** has a predefined arity of 2. It takes one collection \( c \) and generates the following arcs:

\[
- \forall i \in [1, |c| - 1] (c[i], c[i + 1]) \text{ and } (c[i + 1], c[i]), \\
- (c[|c|], c[1]) \text{ and } (c[1], c[|c|]).
\]

The arc generator **CYCLE** is currently not used.

• **GRID**\((d_1, d_2, \ldots, d_n)\) takes a collection \( c \) consisting of \( d_1 \cdot d_2 \cdot \ldots \cdot d_n \) items and generates the arcs \((c[i], c[j])\) where \( i \) and \( j \) satisfy the following condition. There exists an integer \( \alpha (0 \leq \alpha \leq n - 1) \) such that (1) and (2) hold:

\[
(1) \ |i - j| = \prod_{1 \leq k \leq \alpha} d_k \quad \text{(when } \alpha = 0 \text{ we have } \prod_{1 \leq k \leq \alpha} = 1), \\
(2) \ \lfloor \frac{i}{\prod_{1 \leq k \leq \alpha + 1} d_k} \rfloor = \lfloor \frac{j}{\prod_{1 \leq k \leq \alpha + 1} d_k} \rfloor.
\]

**EXAMPLE:** The connect_points constraint uses the **GRID** arc generator.

• **LOOP** has a predefined arity of 2. It takes one collection \( c \) and generates the arcs: \( \forall i \in [1, |c|] : (c[i], c[i + 1]) \). **LOOP** is usually used in order to generate a loop on some vertices, so that they do not disappear from the final graph.

**EXAMPLE:** The global_contiguity(VARIABLES) constraint is an example of constraint that uses the **LOOP** arc generator so that each variable of the VARIABLES collection belongs to the final graph.

• **PATH** can be used with any arity \( a \) greater than or equal to 1. It takes one collection \( c \), and generates the following arcs: \( \forall i \in [1, |c| - a + 1] : (c[i], c[i + 1], \ldots, c[i + a - 1]) \).

**EXAMPLE:** **PATH** is for instance used in the sliding_sum(LOW, UP, SEQ, VARIABLES) constraint with an arity SEQ, where SEQ is an argument of the sliding_sum constraint.
2.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES

- **PATH_N** generates arcs that do not involve the same number of items. It takes one collection \( c \), and generates the following arcs: \( \forall i \in [1,|c|], \forall j \in [i,|c|] : (c[i], c[i+1], \ldots, c[j]) \).

**EXAMPLE:** **PATH_N** is for instance used in the size_max_starting_seq_alldifferent constraint.

- **PRODUCT** has a predefined arity of 2. It takes two collections \( c_1, c_2 \) and generates the arcs: \( \forall i \in [1,|c_1|], \forall j \in [1,|c_2|] : (c_1[i], c_2[j]). \)

**EXAMPLE:** **PRODUCT** is for instance used in the same(VARIABLES1, VARIABLES2) constraint for generating an arc from every item of the VARIABLES1 collection to every item of the VARIABLES2 collection.

- **PRODUCT(Comparison),** where Comparison is one of the comparison operators \( \leq, \geq, <, >, =, \neq \), has a predefined arity of 2. It takes two collections \( c_1, c_2 \) and generates the arcs: \( \forall i \in [1,|c_1|], \forall j \in [1,|c_2|] \) such that \( i \) Comparison \( j : (c_1[i], c_2[j]). \)

**EXAMPLE:** **PRODUCT(=)** is for instance used in the differ_from_at_least_K_pos(K, VECTOR1, VECTOR2) constraint in order to generate an arc between the \( i^{th} \) component of VECTOR1 and the \( i^{th} \) component of VECTOR2.

- **SELF** has a predefined arity of 1. It takes one collection \( c \) and generates the arcs: \( \forall i \in [1,|c|] : (c[i]) \).

**EXAMPLE:** **SELF** is for instance used in the among(VVAR, VARIABLES, VALUES) constraint in order to generate a unary arc constraint in(variables.var, VALUES) for each variable of the VARIABLES collection.

- **SYMMETRIC_PRODUCT** has a predefined arity of 2. It takes two collections \( c_1, c_2 \) and generates the following arcs: \( \forall i \in [1,|c_1|], \forall j \in [1,|c_2|] : (c_1[i], c_2[j]) \) and \( (c_2[j], c_1[i]) \).

**EXAMPLE:** **SYMMETRIC_PRODUCT** is for instance used in the inverse_within_range constraint.

- **SYMMETRIC_PRODUCT(Comparison),** where Comparison is one of the comparison operators \( \leq, \geq, <, >, =, \neq \), has a predefined arity of 2. It takes two collections \( c_1, c_2 \) and generates the arcs: \( \forall i \in [1,|c_1|], \forall j \in [1,|c_2|] \) such that \( i \) Comparison \( j : (c_1[i], c_2[j]) \) and \( (c_2[j], c_1[i]) \).
CHAPTER 2. DESCRIBING GLOBAL CONSTRAINTS

EXAMPLE: The two_orth_do_not_overlap constraint is an example of constraint that uses the SYMMETRIC_PRODUCT(=) arc generator.

- **VOID** takes one collection and does not generate any arc.

EXAMPLE: VOID is for instance used in the lex_lesseq constraint.

Finally, we can combine the PRODUCT arc generator with the arc generators from the following set Generator = \{CIRCUIT, CHAIN, CLIQUE, LOOP, PATH, VOID\}. This is achieved by using the construction PRODUCT(G_1, G_2) where G_1 and G_2 belong to Generator. It applies G_1 to the first collection c_1 passed to PRODUCT and G_2 to the second collection c_2 passed to PRODUCT. Finally, it applies PRODUCT on c_1 and c_2. In a similar way the PRODUCT(Comparison) arc generator is extended to PRODUCT(G_1, G_2, Comparison).

EXAMPLE: As an illustrative example, consider the alldifferent_same_value(NSAME, VARIABLES1, VARIABLES2) constraint, which uses the arc generator PRODUCT(CLIQUE,LOOP,=) on the collections VARIABLES1 and VARIABLES2. It generates the following arcs:

- Since the first argument of PRODUCT is CLIQUE it generates an arc between each pair of items of the VARIABLES1 collection.
- Since the second argument of PRODUCT is LOOP it generates a loop for each item of the VARIABLES2 collection.
- Since the third argument is the comparison operator = it finally generates an arc between an item of the VARIABLES1 collection and an item of the VARIABLES2 collection when the two items have the same position.

Figure 2.3 shows the generated graph under the hypothesis that VARIABLES1 and VARIABLES2 have respectively 3 and 3 items.

Figure 2.3: Example of initial graph generated by PRODUCT(CLIQUE, LOOP, =)

Figure 2.4 illustrates the different arc generators. On the one hand, for those arc generators that take one single collection, we apply them on the collection of items \{i − 1, i − 2, i − 3, i − 4\}. On the other hand, for those arc generators that take two collections, we apply them on \{i − 1, i − 2\} and \{i − 3, i − 4\}. We use the following pictogram for the graphical representation of a constraint network:

- A line for an arc constraint of arity 1,
2.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES

- An arrow for an arc constraint of arity 2,
- A closed line for an arc constraint with an arity strictly greater than 2. In this last case, since the vertices of an arc are ordered, a black circle at one of the extremities indicates the direction of the closed line. For instance consider the example of PATH_1 in Figure 2.4. The closed line that contains vertices 1, 2 and 3 means that a 3-ary arc constraint involves items 1, 2, and 3 in this specific order.

Dotted circles represent vertices that do not belong to the graph. This stems from the fact that the arc generator did not produce any arc involving these vertices. The leftmost lowest corner indicates the arity of the corresponding arc generator:
- An integer if it has a fixed predefined arity,
- \( n \) if it can be used with any arity greater than or equal to 1,
- \( * \) if it generates arcs that do not necessarily involve the same number of items.

Graph properties

We represent a global constraint as the search of a subgraph (i.e., a final graph) of a known initial graph, so that this final graph satisfies a given set of graph properties and eventually belongs to a specific graph class. Most graph properties have the form \( \text{Parameter Comparison Exp} \) or the form \( \text{Parameter} \in [\text{Exp}_1, \text{Exp}_2] \), where \( \text{Parameter} \) is a graph parameter \([53, 182]\), \( \text{Comparison} \) is one of the comparison operators \( =, <, \geq, >, \leq, \neq \), and \( \text{Exp}_1, \text{Exp}_2 \) are expressions that can be evaluated to an integer. Before defining each graph parameter, let’s first introduce some basic vocabulary on graphs.

Graph terminology and notations

A digraph \( G = (V(G), E(G)) \) is a pair where \( V(G) \) is a finite set, called the set of vertices, and where \( E(G) \) is a set of ordered pairs of vertices, called the set of arcs. The arc, path, circuit and strongly connected component of a graph \( G \) correspond to oriented concepts, while the edge, chain, cycle and connected component are non-oriented concepts. However, as reported in [53, page 6] an undirected graph can be seen as a digraph where to each edge we associate the corresponding two arcs. Parts (A) and (B) of Figure 2.5 respectively illustrate the terms for undirected graphs and digraphs.

- We say that \( e_2 \) is a successor of \( e_1 \) if there exists an arc that starts from \( e_1 \) and ends at \( e_2 \). In the same way, we say that \( e_2 \) is a predecessor of \( e_1 \) if there exists an arc that starts from \( e_2 \) and ends at \( e_1 \).

- A vertex of \( G \) that does not have any predecessor is called a source. A vertex of \( G \) that does not have any successor is called a sink.
<table>
<thead>
<tr>
<th>CIRCUIT</th>
<th>LOOP</th>
<th>PRODUCT (≠)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="CIRCUIT Diagram" /></td>
<td><img src="image2" alt="LOOP Diagram" /></td>
<td><img src="image3" alt="PRODUCT(≠) Diagram" /></td>
</tr>
<tr>
<td>CHAIN</td>
<td>PATH</td>
<td>PRODUCT (PATH, VOID)</td>
</tr>
<tr>
<td><img src="image4" alt="CHAIN Diagram" /></td>
<td><img src="image5" alt="PATH Diagram" /></td>
<td><img src="image6" alt="PRODUCT(PATH, VOID) Diagram" /></td>
</tr>
<tr>
<td>CLIQUE</td>
<td>PATH₁</td>
<td>SELF</td>
</tr>
<tr>
<td><img src="image7" alt="CLIQUE Diagram" /></td>
<td><img src="image8" alt="PATH₁ Diagram" /></td>
<td><img src="image9" alt="SELF Diagram" /></td>
</tr>
<tr>
<td>CLIQUE (≤)</td>
<td>PATH₂</td>
<td>SYMMETRIC_PRODUCT</td>
</tr>
<tr>
<td><img src="image10" alt="CLIQUE (≤) Diagram" /></td>
<td><img src="image11" alt="PATH₂ Diagram" /></td>
<td><img src="image12" alt="SYMMETRIC_PRODUCT Diagram" /></td>
</tr>
<tr>
<td>CLIQUE (≥)</td>
<td>PRODUCT</td>
<td>SYMMETRIC_PRODUCT (≤)</td>
</tr>
<tr>
<td><img src="image13" alt="CLIQUE (≥) Diagram" /></td>
<td><img src="image14" alt="PRODUCT Diagram" /></td>
<td><img src="image15" alt="SYMMETRIC_PRODUCT (≤) Diagram" /></td>
</tr>
</tbody>
</table>
| CLIQUE (<) | PRODUCT (<) | SYMMETRIC_PRODUCT (>)
| ![CLIQUE (<) Diagram](image16) | ![PRODUCT (<) Diagram](image17) | ![SYMMETRIC_PRODUCT (>) Diagram](image18) |
| CLIQUE (>) | PRODUCT (>) | SYMMETRIC_PRODUCT (≤) |
| ![CLIQUE (>) Diagram](image19) | ![PRODUCT (>) Diagram](image20) | ![SYMMETRIC_PRODUCT (≤) Diagram](image21) |
| CLIQUE (≠) | PRODUCT (≠) | SYMMETRIC_PRODUCT (≠) |
| ![CLIQUE (≠) Diagram](image22) | ![PRODUCT (≠) Diagram](image23) | ![SYMMETRIC_PRODUCT (≠) Diagram](image24) |
| GRID ([2,2]) | PRODUCT (≥) | SYMMETRIC_PRODUCT (≥) |
| ![GRID ([2,2]) Diagram](image25) | ![PRODUCT (≥) Diagram](image26) | ![SYMMETRIC_PRODUCT (≥) Diagram](image27) |
| CYCLE | PRODUCT (≤) | SYMMETRIC_PRODUCT (≠) |
| ![CYCLE Diagram](image28) | ![PRODUCT (≤) Diagram](image29) | ![SYMMETRIC_PRODUCT (≠) Diagram](image30) |

Figure 2.4: Examples of arc generators
2.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES

(A) Undirected graph
(B) Digraph

Figure 2.5: Graph terminology for an undirected graph and a digraph

- A sequence \((e_1, e_2, \ldots, e_k)\) of edges of \(G\) such that each edge has a common vertex with the previous edge, and the other vertex common to the next edge is called a chain of length \(k\). A chain where all vertices are distinct is called an elementary chain. Each equivalence class of the relation “\(e_i\) is equal to \(e_j\) or there exists a chain between \(e_i\) and \(e_j\)” is a connected component of the graph \(G\).

- A sequence \((e_1, e_2, \ldots, e_k)\) of arcs of \(G\) such that, for each arc \(e_i\) \((1 \leq i < k)\) the end of \(e_i\) is equal to the start of the arc \(e_{i+1}\), is called a path of length \(k\). A path where all vertices are distinct is called an elementary path. Each equivalence class of the relation “\(e_i\) is equal to \(e_j\) or there exists a path between \(e_i\) and \(e_j\)” is a strongly connected component of the graph \(G\).

- A chain \((e_1, e_2, \ldots, e_k)\) of \(G\) is called a cycle if the same edge does not occur more than once in the chain and if the two extremities of the chain coincide. A cycle \((e_1, e_2, \ldots, e_k)\) of \(G\) is called a circuit if for each edge \(e_i\) \((1 \leq i < k)\), the end of \(e_i\) is equal to the start of the edge \(e_{i+1}\).

- Given a graph \(G\), we define the reduced graph \(R(G)\) of \(G\) as follows: to each strongly connected component of \(G\) corresponds a vertex of \(R(G)\); to each arc of \(G\) that connects different strongly connected components corresponds an arc in \(R(G)\) (multiple arcs between the same pair of vertices are merged).

- The rank function associated with the vertices \(V(G)\) of a graph \(G\) that does not contain any circuit is defined in the following way:
  - The rank of the vertices that do not have any predecessor (i.e., the sources) is equal to 0,
  - The rank \(r\) of a vertex \(v\) that is not a source is the length of longest path \((e_1, e_2, \ldots, e_r)\) such that the start of the arc \(e_1\) is a source and the end of arc \(e_r\) is the vertex \(v\).

We now present the different notations used in the catalogue:

- \([k]\) corresponds to \(\{1, \cdots, k\}\) for \(k\) any positive integer.
- Given a set \(X\), \(|X|\) is the number of its elements.
• Given two sets $X$ and $Y$, $X \cup Y$ denotes the union of the two sets when they are disjoint.

• Given a digraph $G$ and $x \in V(G)$, $d^+_G(x) = |\{y : y \in V(G) : (x, y) \in E(G)\}|$ and $d^-_G(x) = |\{y : y \in V(G) : (y, x) \in E(G)\}|$.

• Given a digraph $G$ and $X$ a subset of $V(G)$, the subdigraph of $G$ induced by $X$ is the digraph $G[X]$ where $V(G[X]) = X$ and $E(G[X]) = X^2 \cap E(G)$. By aim of simplicity, we denote $G[V(G) - X]$ by $G - X$. Moreover, if $X = \{x\}$, we use $G - x$ instead of $G - \{x\}$.

• Given two digraphs $G_1$ and $G_2$ such that $V(G_1) \cap V(G_2) = \emptyset$, $G_1 \oplus G_2$ denotes the graph whose vertices set is $V(G_1) \cup V(G_2)$ and whose arcs set is $E(G_1) \cup E(G_2)$.

• Given a graph parameter $P \in \{\text{NCC}, \text{NSCC}\}$, a digraph $G$ and an integer $k$, $\text{CH}(G, k)$ is the number of connected components (respectively strongly connected components) of $G$ with cardinal $k$.

Given a graph parameter, for instance the number of connected components, $\text{NCC}_{\text{INITIAL}}$ will denote the number of connected components of the initial graph (i.e., the graph induced by the constraint under consideration), $\text{NCC}$ will denote the number of connected components of the final graph (i.e., a subgraph of the initial graph). The use of $\text{NCC}(G)$ will denote the number of connected components of the digraph $G$.

Given a global constraint $C$, and a graph parameter $P$ used in the description of $C$, $P$ (respectively $P$) denotes a lower bound (respectively upper bound) of $P$ among all possible final graphs compatible with the current status of $C$.

**Graph parameters** We list in alphabetic order the different graph parameters we consider for a final graph $G_f = (V(G_f), E(G_f))$ associated with a global constraint and give an example of constraint where they are used:

- **MAX_DRG**: largest distance between sources and sinks in the reduced graph associated with $G_f$ (adjacent vertices are at a distance of 1).

  **EXAMPLE**: We do not provide any example since **MAX_DRG** is currently not used.

- **MAX_ID**: number of predecessors of the vertex of $G_f$ that has the maximum number of predecessors without counting an arc from a vertex to itself.

  **EXAMPLE**: The circuit constraint uses the graph property **MAX_ID** = 1 in order to force each vertex of the final graph to have at most one predecessor.
2.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES

- **MAX_NCC**: number of vertices of the largest connected component of $G_f$.

  **EXAMPLE**: The *longest change*($\text{SIZE, VARIABLES, CTR}$) constraint uses the graph property $\text{MAX_NCC} = \text{SIZE}$ in order to catch in $\text{SIZE}$ the maximum number of consecutive variables of the $\text{VARIABLES}$ collection for which constraint $\text{CTR}$ holds.

- **MAX_NSCC**: number of vertices of the largest strongly connected component of $G_f$.

  **EXAMPLE**: The *tree* constraint covers a digraph by a set of trees in such a way that each vertex belongs to a distinct tree. It uses the graph-property $\text{MAX_NSCC} \leq 1$ in order to avoid to have any circuit involving more than one vertex.

- **MAX_OD**: number of successors of the vertex of $G_f$ that has the maximum number of successors without counting an arc from a vertex to itself.

  **EXAMPLE**: The *tour* constraint enforces to cover a graph with a Hamiltonian cycle. It uses the graph-property $\text{MAX_OD} = 2$ to enforce that each vertex of $G_f$ have at most two successors. Since the *tour* constraint uses the $\text{CLIQUE(\#)}$ arc generator the vertices of $G_f$ do not have any loop.

- **MIN_DRG**: smallest distance between sources and sinks in the reduced graph associated with $G_f$ (adjacent vertices are at a distance of 1).

  **EXAMPLE**: We do not provide any example since $\text{MIN_DRG}$ is currently not used by any constraint.

- **MIN_ID**: number of predecessors of the vertex of $G_f$ that has the minimum number of predecessors without counting an arc from a vertex to itself.

  **EXAMPLE**: The *tour* constraint enforces to cover a graph with a Hamiltonian cycle. It uses the graph-property $\text{MIN_ID} = 2$ to enforce that each vertex of $G_f$ have at most two predecessors. Since the *tour* constraint uses the $\text{CLIQUE(\#)}$ arc generator the vertices of $G_f$ do not have any loop.
• **MIN**\_**NCC**: number of vertices of the smallest connected component of \(G_f\).

**EXAMPLE:** Within the `group` constraint, each connected component of \(G_f\) corresponds to a maximum sequence of consecutive variables that take their value in a given set of values. Therefore, the graph-property \(\text{MIN\_NCC} = \text{MIN\_SIZE}\) enforces that the smallest sequence of such variables consist of \(\text{MIN\_SIZE}\) variables.

• **MIN**\_**NSCC**: number of vertices of the smallest strongly connected component of \(G_f\).

**EXAMPLE:** The `circuit(NODES)` constraint enforces covering a digraph with one circuit visiting once all its vertices. The graph-property \(\text{MIN\_NSCC} = |\text{NODES}|\) enforces that the smallest strongly connected component of \(G_f\) contain \(|\text{NODES}|\) vertices. Since \(|\text{NODES}|\) also corresponds to the number of vertices of the initial graph this means that \(G_f\) is a strongly connected component involving all the vertices. This is clearly a necessary condition for having a circuit visiting once all vertices.

\(^a\)Of course, this is not enough, and the description of the `circuit` constraint asks for some other properties.

• **MIN**\_**OD**: number of successors of the vertex of \(G_f\) that has the minimum number of successors without counting an arc from a vertex to itself.

**EXAMPLE:** The `tour` constraint enforces to cover a graph with a Hamiltonian cycle. It uses the graph-property \(\text{MIN\_OD} = 2\) to enforce that each vertex of \(G_f\) have at most two successors.

\(^a\)Since the `tour` constraint uses the `CLIQUE(\#)` arc generator the vertices of \(G_f\) do not have any loop.

• **NARC**: cardinality of the set \(E(G_f)\).

**EXAMPLE:** The `disjoint(VARIABLES1, VARIABLES2)` constraint enforces that each variable of the collection `VARIABLES1` take a value that is distinct from all the values assigned to the variables of the collection `VARIABLES2`. This is imposed by creating an arc from each variable of `VARIABLES1` to each variable of `VARIABLES2`. To each arc corresponds an equality constraint involving the variables associated with the extremities of the arc. Finally, the graph property \(\text{NARC} = 0\) forces \(G_f\) to be empty so that no value is both assigned to a variable of `VARIABLES1` as well as to a variable of `VARIABLES2`. 
2.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES

- **NARC_NO_LOOP** : cardinality of the set $E(G_f)$ without considering the arcs linking the same vertex (i.e., a loop).

  **EXAMPLE:** The constraint `alldifferent_same_value` uses the graph-property `NARC_NO_LOOP`.

- **NCC** : number of connected components of $G_f$.

  **EXAMPLE:** The `tree` constraint covers a digraph by $\text{NTREES}$ trees in such a way that each vertex belongs to a distinct tree. It uses the graph-property $\text{NCC} = \text{NTREES}$ in order to state that $G_f$ is made up from $\text{NTREES}$ connected components.

- **NSCC** : number of strongly connected components of $G_f$.

  **EXAMPLE:** The constraint `nvalue(nval, VARIABLES)` forces $nval$ to be equal to the number of distinct values assigned to the variables of the collection `VARIABLES`. This is enforced by using the graph-property $\text{NSCC} = nval$. Each strongly connected component of the final graph corresponds to the variables that are assigned to the same value.

- **NSINK** : number of vertices of $G_f$ that do not have any successor.

  **EXAMPLE:** The `same(VARIABLES1, VARIABLES2)` enforces that the variables of the `VARIABLES1` collection correspond to the variables of the `VARIABLES2` collection according to a permutation. We first create an arc from each variable of `VARIABLES1` to each variable of `VARIABLES2`. To each arc corresponds an equality constraint involving the variables associated with the extremities of the arc. We use the graph-property $\text{NSINK} = |VARIABLES2|$ in order to express the fact that each value assigned to a variable of `VARIABLES2` should also be assigned to a variable of `VARIABLES1`. 
CHAPTER 2. DESCRIBING GLOBAL CONSTRAINTS

• **NSINK\_NSOURCE**: sum over the different connected components of $G_f$ of the minimum of the number of sinks and the number of sources of a connected component.

**EXAMPLE**: The `soft_same_var`($C, \text{VARIABLES1, VARIABLES2}$) constraint enforces $C$ to be the minimum number of values to change in the \text{VARIABLES1} and the \text{VARIABLES2} collections of variables\(^1\), so that the variables of \text{VARIABLES2} correspond to the variables of \text{VARIABLES1} according to a permutation.

A connected component $C_{val}$ of the final graph $G_f$ corresponds to all variables that are assigned to the same value $val$: the sources and the sinks of $C_{val}$ respectively correspond to the variables of \text{VARIABLES1} and to the variables of \text{VARIABLES2} that are assigned to $val$. For a connected component, the minimum of the number of sources and sinks expresses the number of variables for which we do not need to make any change. Therefore we use the graph-property $\text{NSINK\_NSOURCE} = |\text{VARIABLES1}| - C$ for encoding the meaning of the `soft_same_var` constraint.

\(^1\)Both collections have the same number of variables.

• **NSOURCE**: number of vertices of $G_f$ that do not have any predecessor.

**EXAMPLE**: The `same`($\text{VARIABLES1, VARIABLES2}$) enforces that the variables of the \text{VARIABLES1} collection correspond to the variables of the \text{VARIABLES2} collection according to a permutation.

We first create an arc from each variable of \text{VARIABLES1} to each variable of \text{VARIABLES2}. To each arc corresponds an equality constraint involving the variables associated with the extremities of the arc. We use the graph-property $\text{NSOURCE} = |\text{VARIABLES1}|$ in order to express the fact that each value assigned to a variable of \text{VARIABLES1} should also be assigned to a variable of \text{VARIABLES2}.

• **NTREE**: number of vertices of $G_f$ that do not belong to any circuit and for which at least one successor belongs to a circuit. Such vertices can be interpreted as root nodes of a tree.

**EXAMPLE**: The `cycle`($\text{NCYCLE, NODES}$) enforces that $\text{NCYCLE}$ equal the number of circuits for covering an initial graph in such a way that each vertex belongs to one single circuit.

The graph-property $\text{NTREE} = 0$ enforces that all vertices of the final graph belong to a circuit.
• **NVERTEX**: cardinality of the set $V(G_f)$.

**EXAMPLE**: The $\text{cutset(\text{SIZE\_CUTSET, NODES})}$ constraint considers a digraph with $n$ vertices described by the NODES collection. It enforces that the subset of kept vertices of cardinality $n - \text{SIZE\_CUTSET}$ and their corresponding arcs form a graph without a circuit. It uses the graph-property $\text{NVERTEX} = n - \text{SIZE\_CUTSET}$ for enforcing that the final graph $G_f$ contain the required number of vertices.

• **RANGE\_DRG**: difference between the largest distance between sources and sinks in the reduced graph associated with $G_f$ and the smallest distance between sources and sinks in the reduced graph associated with $G_f$.

**EXAMPLE**: The $\text{tree\_range}$ constraint enforces to cover a digraph in such a way that each vertex belongs to a distinct tree. In addition it forces the difference between the longest and the shortest paths of $G_f$ to be equal to the variable $R$. For this purpose it uses the graph-property $\text{RANGE\_DRG} = R$.

• **RANGE\_NCC**: difference between the number of vertices of the largest connected component of $G_f$ and the number of vertices of the smallest connected component of $G_f$.

**EXAMPLE**: We do not provide any example since $\text{RANGE\_NCC}$ is currently not used by any constraint.

• **RANGE\_NSCC**: difference between the number of vertices of the largest strongly connected component of $G_f$ and the number of vertices of the smallest strongly connected component of $G_f$.

**EXAMPLE**: The $\text{balance(\text{BALANCE, VARIABLES})}$ constraint forces $\text{BALANCE}$ to be equal to the difference between the number of occurrences of the value that occurs the most and the value that occurs the least within the collection of variables $\text{VARIABLES}$. Each strongly connected component of $G_f$ corresponds to the variables that are assigned to the same value. The graph property $\text{RANGE\_NSCC} = \text{BALANCE}$ allows for expressing this definition.

• **ORDER(rank, default, attr)**
  
  – rank is an integer or an argument of type integer of the global constraint,
  – default is an integer,
– attr is an attribute corresponding to an integer or to a domain variable that occurs in all the collections that were used for generating the vertices of the initial graph.

We explain what is the value associated with \texttt{ORDER}(\texttt{rank}, \texttt{default}, attr). Let \(\mathcal{V}\) denote the vertices of rank \(\texttt{rank}\) of \(G_f\) from which we remove any loops.

– When \(\mathcal{V}\) is not empty, it corresponds to the values of attribute \(\texttt{attr}\) of the items associated with the vertices of \(\mathcal{V}\),

– Otherwise, when \(\mathcal{V}\) is empty, it corresponds to the default value \texttt{default}.

\textbf{EXAMPLE:} The \texttt{minimum}(\texttt{MIN}, \texttt{VARIABLES}) forces \texttt{MIN} to be the minimum value of the collection of domain variables \texttt{VARIABLES}. There is an arc from a variable \texttt{var}_1 to a variable \texttt{var}_2 if and only if \texttt{var}_1 < \texttt{var}_2. The graph-property \texttt{ORDER}(0, \texttt{MAXINT}, \texttt{var}) = \texttt{MIN} expresses the fact that \texttt{MIN} is equal to the value of the source of \(G_f\) (since \texttt{rank} = 0).

\textbullet \hspace{1em} \textbf{PATH\_FROM\_TO} (attr, from, to)

– \* attr is an attribute corresponding to an integer that occurs in all the collections that were used for generating the vertices of the initial graph,

\* from is an integer or an argument of type integer of the global constraint,

\* to is an integer or an argument of type integer of the global constraint.

Let \(\mathcal{F}\) (respectively \(\mathcal{T}\)) denote the vertices of \(G_f\) such that \texttt{attr} is equal to \texttt{from} (respectively \texttt{to}). \texttt{PATH\_FROM\_TO} (attr, from, to) is equal to 1 if there exists a path between each vertex of \(\mathcal{F}\) and each vertex of \(\mathcal{T}\), and 0 if there exists no path between a vertex of \(\mathcal{F}\) and a vertex of \(\mathcal{T}\).

– \* attr is an attribute corresponding to an integer that occurs in all the collections that were used for generating the vertices of the initial graph,

\* from is an attribute corresponding to an integer or to a set of integers that occurs in all the collections that were used for generating the vertices of the initial graph,

\* to is an attribute corresponding to an integer or to a set of integers that occurs in all the collections that were used for generating the vertices of the initial graph.

For each vertex \(v\) of \(G_f\) let:

\* \(\mathcal{F}_v\) the set of vertices for which the value of the attribute \texttt{attr} is equal to the \texttt{from} attribute (or is included within the \texttt{from} attribute when it corresponds to a set of integers).
2.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES

- $\mathcal{T}_v$: the set of vertices for which the value of the attribute $\text{attr}$ is equal to the $\text{to}$ attribute (or is included within the $\text{to}$ attribute when it corresponds to a set of integers).

$\text{PATH\_FROM\_TO}(\text{attr, from, to})$ is equal to

- $1$ if for each vertex of $G_f$ there exists a path between each vertex of $\mathcal{F}_v$ and each vertex of $\mathcal{T}_v$.
- $0$ if for a vertex of $G_f$ there is no path between a vertex of $\mathcal{F}_v$ and a vertex of $\mathcal{T}_v$.

**EXAMPLE:** The constraints $\text{lex\_lesseq}$ and $\text{stable\_compatibility}$ use the $\text{PATH\_FROM\_TO}$ graph-property.

- **PROD**(col, attr)
  
  - col is a collection that was used for generating the vertices of the initial graph,
  
  - attr is an attribute corresponding to an integer or to a domain variable of the collection col.

Let $\mathcal{V}$ be the set of vertices of $G_f$ that were generated from the items of the collection col.

- If $\mathcal{V}$ is not empty, $\text{PROD}(\text{col, attr})$ corresponds to the product of the values of attribute attr associated with the vertices of $\mathcal{V}$,
  
  - Otherwise, if $\mathcal{V}$ is empty, $\text{PROD}(\text{col, attr})$ is equal to $1$.

**EXAMPLE:** The constraint $\text{product\_ctr}(\text{VARIABLES, CTR, VAR})$ forces the product of the variables of the VARIABLES collection to be equal, less than or equal, ... to a given domain variable VAR.

To each variable of VARIABLES corresponds a vertex of the initial graph. Since we want to keep all the vertices of the initial graph we use the $\text{SELF}$ arc generator together with the $\text{TRUE}$ arc constraint. Finally, $\text{PROD}(\text{VARIABLES, var}) \text{ CTR VAR}$ expresses the required condition. In this expression var and CTR respectively corresponds to the attribute of the collection VARIABLES (a domain variable) and to the condition we want to enforce. Since the final graph $G_f$ contains all the vertices of the initial graph, the expression $\text{PROD}(\text{VARIABLES, var})$ corresponds to the product of the variables of the VARIABLES collection.

- **RANGE**(col, attr)
  
  - col is a collection that was used for generating the vertices of the initial graph,
CHAPTER 2. DESCRIBING GLOBAL CONSTRAINTS

- attr is an attribute corresponding to an integer or to a domain variable of the collection col.

Let \( \mathcal{V} \) be the set of vertices of \( G_f \) that were generated from the items of the collection col.

- If \( \mathcal{V} \) is not empty, \( \text{RANGE}(\text{col}, \text{attr}) \) corresponds to the difference between the maximum and the minimum values of attribute attr associated with the vertices of \( \mathcal{V} \).
- Otherwise, if \( \mathcal{V} \) is empty, \( \text{RANGE}(\text{col}, \text{attr}) \) is equal to 0.

**EXAMPLE:** The constraint \( \text{range}_\text{ctr}(\text{VARIABLES}, \text{CTR}, \text{VAR}) \) forces the difference between the maximum value and the minimum value of the variables of the VARIABLES collection to be equal, less than or equal, ... to a given domain variable VAR.

To each variable of VARIABLES corresponds a vertex of the initial graph. Since we want to keep all the vertices of the initial graph we use the SELF arc generator together with the TRUE arc constraint. Finally, \( \text{RANGE}(\text{VARIABLES}, \text{var}) \) \( \text{CTR} \) \( \text{VAR} \) expresses the required condition. In this expression var and CTR respectively corresponds to the attribute of the collection VARIABLES (a domain variable) and to the condition we want to enforce. Since the final graph \( G_f \) contains all the vertices of the initial graph, the expression \( \text{RANGE}(\text{VARIABLES}, \text{var}) \) corresponds to the difference between the maximum value and the minimum value of the variables of the VARIABLES collection.

- \( \text{SUM}(\text{col}, \text{attr}) \)
  - col is a collection that was used for generating the vertices of the initial graph,
  - attr is an attribute corresponding to an integer or to a domain variable of the collection col.

Let \( \mathcal{V} \) be the set of vertices of \( G_f \) that were generated from the items of the collection col.

- If \( \mathcal{V} \) is not empty, \( \text{SUM}(\text{col}, \text{attr}) \) corresponds to the sum of the values of attribute attr associated with the vertices of \( \mathcal{V} \).
- Otherwise, if \( \mathcal{V} \) is empty, \( \text{SUM}(\text{col}, \text{attr}) \) is equal to 0.

**EXAMPLE:** The constraint \( \text{sum}_\text{ctr}(\text{VARIABLES}, \text{CTR}, \text{VAR}) \) forces the sum of the variables of the VARIABLES collection to be equal, less than or equal, ... to a given domain variable VAR.

To each variable of VARIABLES corresponds a vertex of the initial graph. Since we want to keep all the vertices of the initial graph we use the SELF arc generator together with the TRUE arc constraint. Finally, \( \text{SUM}(\text{VARIABLES}, \text{var}) \) \( \text{CTR} \) \( \text{VAR} \) expresses the required condition. In this expression var and CTR respectively correspond to the attribute of the collection VARIABLES (a domain variable) and to the condition we want to enforce. Since the final graph \( G_f \) contains all the vertices of the initial graph, the expression \( \text{SUM}(\text{VARIABLES}, \text{var}) \) corresponds to the sum of the variables of the VARIABLES collection.
2.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES

- **SUM_WEIGHT_ARC**(Expr) Expr is an arithmetic expression. For each arc \( a \) of \( E(G_f) \), let \( f(a) \) denote the value of Expr. **SUM_WEIGHT_ARC**(Expr) is equal to \( \sum_{a \in E(G_f)} f(a) \). The value of Expr usually depends on the attributes of the items located at the extremities of an arc.

**EXAMPLE:** The constraint **global_cardinality_with_costs**(VARIABLES, VALUES, MATRIX, COST) enforces that each value VALUES\([i].val\) be assigned to exactly VALUES\([i].noccurrence\) variables of the VARIABLES collection. In addition the COST of an assignment is equal to the sum of the elementary costs associated with the fact that we assign the \( i^{th} \) variable of the VARIABLES collection to the \( j^{th} \) value of the VALUES collection. These elementary costs are given by the MATRIX collection. The graph-property **SUM_WEIGHT_ARC**(MATRIX\([(\text{variables.key}-1)\times|\text{VALUES}|+\text{values.key}.c]) = \text{COST} \) expresses that the COST variable is equal to the sum of the elementary costs associated with each variable-value assignment. All these elementary costs are recorded in the MATRIX collection. More precisely, the cost \( c_{ij} \) is recorded in the attribute \( c \) of the \((i - 1) \times |\text{VALUES}| + j\)th entry of the MATRIX collection.

A last graph parameter, **DISTANCE**, is computed on two final graphs \( G_1 \) and \( G_2 \) that have the same set \( V \) of vertices and the sets \( E(G_1) \) and \( E(G_2) \) of arcs. This graph parameter is the cardinality of the set \((E(G_1) - E(G_2)) \cup (E(G_2) - E(G_1))\). This corresponds to the number of arcs that belong to \( E(G_1) \) but not to \( E(G_2) \), plus the number of arcs that are in \( E(G_2) \) but not in \( E(G_1) \).

**Graph class** For a given global constraint, a graph class specifies a general property that holds on its final digraph. We list the different graph classes and, for each of them, we point to some global constraints that fit in that class. Finding all the global constraints corresponding to a given graph class can be done by looking into the list of keywords (see Section 3.7 on page 147).

- **ACYCLIC** : the final graph doesn’t have any circuit.

- **BIPARTITE** : the final graph is bipartite.

- **CONSECUTIVE_LOOPS_ARE_CONNECTED** : denotes that the graph constraint of a global constraint uses only the PATH and the LOOP arc generators and that the final graph does not contain consecutive vertices that have a loop and that are not connected together by an arc.

- **EQUIVALENCE** : the final graph is reflexive, symmetric and transitive.
• **NO_LOOP**: the final graph doesn’t have any loop.

• **ONE_SUCC**: the vertices of the initial graph belong to the final graph and all vertices of the final graph have exactly one successor.

• **SYMMETRIC**: the final graph is symmetric. A digraph is symmetric if and only if, if there is an arc from a vertex $u$ to a vertex $v$, there is also an arc from $v$ to $u$.

### 2.2.3 Graph constraint

A global constraint can be defined as a conjunction of several *simple* or *dynamic graph constraints* that all share the same name, the same arguments and the same argument restrictions. This section first describes *simple graph constraints* and then *dynamic graph constraints*, which are an extension of *simple graph constraints*.

#### Simple graph constraint

To a *simple graph constraint* correspond several initial graphs, usually one, where all the initial graphs have the same vertices and arcs. Specifying more than one initial graph is usually achieved by using the `FOR ALL ITEMS OF` iterator (e.g., see for instance the definition of the `global_cardinality` constraint), which takes a collection $C$ and generates an initial graph $G_i(t)$ for each item $t$ of $C$. In this context, the arc constraints and/or graph properties of an initial graph may depend of the attributes of the item $t$ of $C$ from which they were generated. All arc constraints attached to a given arc have to be pairwise mutually incompatible.

The graphs of a *simple graph constraint* are defined by the following slots:

• An **Arc input(s)** slot, which consists of:
  
  – Either a sequence of collections $C_1, C_2, \ldots, C_d$ ($d \geq 1$). To each item of these collections corresponds a vertex of the initial graph (i.e., in this context we generate one single initial graph).
  
  – Either a list of sequences of collections. To each item of the collections of a given sequence corresponds a vertex of one of the initial graphs (i.e., in this context we generate one initial graph for each sequence).

---

20 For an example of global constraint that is defined by more than one graph constraint see for instance the `sort` constraint and its two graph constraints.

21 The arguments and the argument restrictions were described in Section 2.1.4 on page 17.

22 An other way of generating several initial graphs will be explained later on in the **Arc input(s)** slot.

23 As we previously said, even if we have more than one initial graph, all vertices and arcs of the different initial graphs are identical.

24 Two arc constraints $\text{constraint}_1(X_1, X_2, \ldots, X_n)$ and $\text{constraint}_2(X_1, X_2, \ldots, X_n)$ are *incompatible* if there does not exist any tuple of values $(v_1, v_2, \ldots, v_n)$ such that both $\text{constraint}_1(X_1, X_2, \ldots, X_n)$ and $\text{constraint}_2(X_1, X_2, \ldots, X_n)$ hold.

25 This is for instance the case for the `distance_between` constraint.
• An Arc generator slot, which can be one or several expressions\(^{26}\) of the following forms:

\(\text{ARC GENERATOR} \rightarrow \text{collection(item}_1, \text{item}_2, \ldots, \text{item}_a)\), where \(\text{ARC GENERATOR}\) is one of the arc generators with a fixed arity\(^{27}\) defined in Section 2.2.2 on page 52, and \(\text{item}_i (1 \leq i \leq a)\) denotes the \(i^{th}\) item associated with the \(i^{th}\) vertex of an arc. These items correspond to formal parameters\(^{28}\) which can be used within an arc constraint. When the Arc input(s) slot consists of one single collection \((d = 1)\), \(\text{item}_i (1 \leq i \leq a)\) represents an item of the collection \(C_1\). Otherwise, when \(d > 1\), we must have \(a = d\) and, in this context, \(\text{item}_i (1 \leq i \leq a)\) represents an item of \(C_i\).

**EXAMPLE:** The \text{alldifferent}(\text{VARIABLES}) constraint has the following Arc input(s) and Arc generator slots:

* Its Arc input(s) slot refers only to the collection \text{VARIABLES} (i.e., \(d = 1\)).
* Its Arc generator slot consists of
  \(\text{CLIQUE} \rightarrow \text{collection(\text{variables}_1, \text{variables}_2)}\) (i.e., \(a = 2\)).

In this context, where \(d = 1\), both \text{variables}_1 and \text{variables}_1 are items of the \text{VARIABLES} collection.

**EXAMPLE:** The \text{name}(\text{VARIABLES1, VARIABLES2}) constraint has the following Arc input(s) and Arc generator slots:

* Its Arc input(s) slot refers to the collections \text{VARIABLES1} and \text{VARIABLES2} (i.e., \(d = 2\)).
* Its Arc generator slot consists of
  \(\text{PRODUCT} \rightarrow \text{collection(\text{variables}_1, \text{variables}_2)}\) (i.e., \(a = 2\)).

In this context, where \(d > 1\), \text{variables}_1 and \text{variables}_1 respectively correspond to items of the \text{VARIABLES1} and the \text{VARIABLES2} collections.

\(\text{ARC GENERATOR} \rightarrow \text{collection}\), where \(\text{ARC GENERATOR}\) is one of the arc generators \text{PATH}_1 or \text{PATH}_N. In this context, collection denotes a collection of items corresponding to the vertices of an arc of the initial graph. An arc constraint enforces a restriction on the items of this collection.

\(^{26}\) Usually one single expression.

\(^{27}\) Any arc generator different from \text{PATH}_1 and \text{PATH}_N.

\(^{28}\) See the description of simple arithmetic expressions page 48.
EXAMPLE:
The \texttt{size_max_seq_alldifferent} (\texttt{SIZE}, \texttt{VARIABLES}) constraint has the following \textbf{Arc input(s)} and \textbf{Arc generator} slots:

\begin{itemize}
  \item Its \textbf{Arc input(s)} slot refers to the \texttt{VARIABLES} collection.
  \item Its \textbf{Arc generator} slot consists of \texttt{PRODUCT} \mapsto \texttt{collection}.
\end{itemize}

In this context, \texttt{collection} is a collection of the same type as the \texttt{VARIABLES} collection. It corresponds to the variables associated with an arc of the initial graph.

When the \textbf{Arc generator} slot consists of $n$ ($n > 1$) expressions then these expressions have the form:

\begin{align*}
  &ARC\_GENERATOR_1 \mapsto \texttt{collection}(\text{item}_1, \text{item}_2, \ldots, \text{item}_a) \\
  &ARC\_GENERATOR_2 \mapsto \texttt{collection}(\text{item}_1, \text{item}_2, \ldots, \text{item}_a) \\
  \vdots &
  \vdots \hspace{1cm} \vdots \\
  &ARC\_GENERATOR_n \mapsto \texttt{collection}(\text{item}_1, \text{item}_2, \ldots, \text{item}_a)
\end{align*}

All leftmost part of the expressions must be the same since they will be involved in one single \textbf{Arc constraint(s)} slot. The \texttt{global_contiguity} constraint is an example of global constraint where more than one arc generator is used.

- An \textbf{Arc arity} slot, which corresponds to the number of vertices $a$ of each arc of the initial graph. $a$ is either a strictly positive integer, an argument of the global constraint of type \texttt{int}, or the character \texttt{*}. In this last case, this is used for denoting that all the arc constraints do not involve the same number of vertices. This is for instance the case when we use the arc generators \texttt{PATH\_1} or \texttt{PATH\_N} as in the \texttt{arith_sliding} or the \texttt{size_max_seq_alldifferent} constraints.

- An \textbf{Arc constraint(s)} slot, which corresponds to a conjunction of \textit{arc constraints}\footnote{Usually this conjunction consists of one single \textit{arc constraint}.} that were introduced in Section 2.2.2 on page 48.

- A \textbf{Graph property(ies)} slot, which corresponds to one or several \textit{graph properties} (see Section 2.2.2 on page 57) to be satisfied on the final graphs associated with an instantiated solution of the global constraint. To each initial graph corresponds one final graph obtained by removing all arcs for which the corresponding arc constraints do not hold as well as all vertices that do not have any arc.

We now give several examples of descriptions of \textit{simple graph constraints}, starting from the \texttt{nvalue} constraint, which was introduced as a first example of global constraint that can be modelled by a graph property in Section 2.2.1 on page 39.
2.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES

**EXAMPLE:** The constraint \texttt{nvalue}(\texttt{NVAL}, \texttt{VARIABLES}) restricts \texttt{NVAL} to be the number of distinct values taken by the variables of the collection \texttt{VARIABLES}. Its meaning is described by a simple graph constraint corresponding to the following items:

- **Arc input(s):** \texttt{VARIABLES}
- **Arc generator:** \texttt{CLIQUE} $\mapsto$ \texttt{collection}(\texttt{variables1}, \texttt{variables2})
- **Arc arity:** 2
- **Arc constraint(s):** variables1.var = variables2.var
- **Graph property(ies):** \texttt{NSCC} = \texttt{NVAL}

Since this description does not use the FOR ALL ITEMS OF iterator we generate one single initial graph. Each vertex of this graph corresponds to one item of the \texttt{VARIABLES} collection. Since we use the \texttt{CLIQUE} arc generator we have an arc between each pair of vertices. An arc constraint corresponds to an equality constraint between the two variables that are associated with the extremities of the arc. Finally, the Graph property(ies) slot forces the final graph to have \texttt{NVAL} strongly connected components.

**EXAMPLE:** The constraint \texttt{global.contiguity}(\texttt{VARIABLES}) forces all variables of the \texttt{VARIABLES} collection to be assigned to 0 or 1. In addition, all variables assigned to value 1 appear contiguously. Its meaning is described by a simple graph constraint corresponding to the following items:

- **Arc input(s):** \texttt{VARIABLES}
- **Arc generator:** \texttt{PATH} $\mapsto$ \texttt{collection}(\texttt{variables1}, \texttt{variables2})
  \texttt{LOOP} $\mapsto$ \texttt{collection}(\texttt{variables1}, \texttt{variables2})
- **Arc arity:** 2
- **Arc constraint(s):** variables1.var = variables2.var
  variables1.var = 1
- **Graph property(ies):** \texttt{NCC} $\leq$ 1

Since this description does not use the FOR ALL ITEMS OF iterator we generate one single initial graph. Each vertex of this graph corresponds to one item of the \texttt{VARIABLES} collection. Since we use the \texttt{PATH} arc generator we generate an arc from item \texttt{VARIABLES}[i] to item \texttt{VARIABLES}[i + 1] (1 $\leq$ i $<$ |\texttt{VARIABLES}|). In addition, since we use the \texttt{LOOP} arc generator, we generate also an arc from each item of the \texttt{VARIABLES} collection to itself.\footnote{We use the \texttt{LOOP} arc generator in order to keep in the final graph those isolated variables assigned to 1. This is because isolated vertices with no arcs are always removed from the final graph.} The effect of the arc constraint is to keep in the final graph those vertices for which the corresponding variable is assigned to 1. Adjacent variables assigned to 1 form a connected component of the final graph and the graph property \texttt{NCC} $\leq$ 1 enforces to have at most one such group of adjacent variables assigned to 1.
CHAPTER 2. DESCRIBING GLOBAL CONSTRAINTS

EXAMPLE:
The \texttt{global\_cardinality(VARIABLES, VALUES)} constraint enforces that each value VALUES[i].\texttt{val} (1 \leq i \leq \text{\texttt{|VALUES|}}) be taken by exactly VALUES[i].\texttt{noccurrence} variables of the \texttt{VARIABLES} collection. Its meaning is described by a \textit{simple graph constraint} corresponding to the following items:

- **For all items of VALUES:**
  - Arc input(s) : VARIABLES
  - Arc generator : \texttt{SELF} $\mapsto$ collection(variables)
  - Arc arity : 1
  - Arc constraint(s) : variables.var = VALUES.val

- **Graph property(ies):** NVERTEX = VALUES.noccurrence

Since this description uses the \texttt{For all items of VALUES} iterator on the \texttt{VALUES} collection we generate an initial graph for each item of the \texttt{VALUES} collection (i.e., one graph for each value). Each vertex of an initial graph corresponds to one item of the \texttt{VARIABLES} collection. Since we use the \texttt{SELF} arc generator we have an arc for each vertex. For an initial graph associated with a value \texttt{val} an arc constraint on a vertex \texttt{v} corresponds to an equality constraint between the variable associated with \texttt{v} and the value \texttt{val}. Finally, the **Graph property(ies) slot** forces the final graph to have a given number of vertices (i.e., associated with the attribute \texttt{val}).

Dynamic graph constraint

The purpose of a \textit{dynamic graph constraint} is to enforce a condition on different subsets of variables, not known in advance. This situation occurs frequently in practice and is hard to express since one cannot use a classical constraint for which it is required to provide all variables right from the beginning. One good example of such global constraint is the \texttt{cumulative} constraint where one wants to force the sum of some variables to be less than or equal to a given limit. In the context of the \texttt{cumulative} constraint, each set of variables is defined by the height of the different tasks that overlap a given instant \texttt{i}. Since the origins of the tasks are not initially fixed, we do not know in advance which task will overlap a given instant and so, we cannot state any sum constraint initially.

A \textit{dynamic graph constraint} is defined in exactly the same way as a \textit{simple graph constraint}, except that we may omit the **Graph property(ies) slot**, and that we have to provide the two following additional slots:

- The **Set** slot denotes a generator of sets of vertices. Such a generator takes as argument a final graph and produces different sets of vertices. In order to have something tractable, we force the total number of generated sets to be polynomial in the number of vertices.

  In practice each set of vertices is represented by a collection of items. The type of this collection corresponds either to the type of the items associated with the vertices, or to the type of a new derived collection. This is achieved by providing an expression of the form \texttt{name} or \texttt{name-\texttt{derived}\_\texttt{collection}}, where \texttt{name}
represents a formal parameter, and derived_collection a declaration of a new derived collection (as specified in Section 2.2.2 on page 42).

- The **Constraint(s) on sets** slot provides a global constraint defined in the catalogue that has to hold for each set created by the previous generator.

We now describe the different generators of sets of vertices currently available:

- **ALL_VERTICES** generates one single set containing all the vertices of the final graph. It is specified by a declaration of the form
  \[ \text{ALL\_VERTICES} >> \{\text{vertices}\} \]
  where vertices represents all the vertices of the final graph.

- **CC** generates one set of vertices for each connected component of the final graph. These sets correspond to all the vertices of a given connected component. It is specified by a declaration of the form
  \[ \text{CC} >> \{\text{connected}\_\text{component}\} \]
  where connected\_component represents the vertices of a connected component of the final graph.

- **PATH\_LENGTH\(_L\)** generates all elementary paths\(^{30}\) of \(L\) vertices of the final graph such that, discarding loops, all vertices of a path (except the last one) have no more than one successor in the final graph. It is specified by a declaration of the form
  \[ \text{PATH\_LENGTH}\(_L\) >> \{\text{path}\} \]
  where path represents the vertices of an elementary path, ordered according to their occurrence in the path.

- **PRED** generates the non-empty sets corresponding to the predecessors of each vertex of the final graph. It is specified by a declaration of the form
  \[ \text{PRED} >> \{\text{predecessor,} \text{successor}\} \]
  where destination represents a vertex of the final graph and predecessor its predecessors.

- **SUCC** generates the non-empty sets corresponding to the successors of each vertex of the final graph. It is specified by a declaration of the form
  \[ \text{SUCC} >> \{\text{source,} \text{successor}\} \]
  where source represents a vertex of the final graph and successor its successors.

As an illustrative example of dynamic graph constraint we now consider the cumulative constraint.

\(^{30}\)A path where all vertices are distinct is called an elementary path.
EXAMPLE: The cumulative(TASKS, LIMIT) constraint, where TASKS is a collection of the form collection(origin − dvar, duration − dvar, end − dvar, height − dvar), and where LIMIT is a non-negative integer, holds if, for any point the cumulated height of the set of tasks that overlap that point, does not exceed LIMIT.

The first graph constraint of cumulative enforces for each task of the TASKS collection the equality origin + duration = end. We focus on the second graph constraint, which uses a dynamic graph constraint described by the following items:

- **Arc input(s)**: TASKS TASKS
- **Arc generator**: PRODUCT ↦ collection(tasks1, tasks2)
- **Arc arity**: 2
- **Arc constraint(s)**:
  - tasks1.duration > 0
  - tasks2.origin ≤ tasks1.origin
  - tasks1.origin ≤ tasks2.end
- **Sets**:
  - SUCC
    - source,
    - variables − col(VARIABLES − collection(var − dvar),
      - item(var − TASKS.height))]
- **Constraint(s) on sets**: sum_ctr(variables, ≤, LIMIT)

The second graph constraint is defined by:

- To each item of the TASKS collection correspond two vertices of the initial graph.
- The arity of the arc constraint is 2.
- The arcs of the initial graph are constructed with the PRODUCT arc generator between the TASKS collection and the TASKS collection. Therefore, each vertex associated with a task is linked to all the vertices related to the different tasks.
- The arc constraint that is associated with an arc between a task tasks1 and a task tasks2 is an overlapping constraint that holds if both, the duration of tasks1 is strictly greater than zero, and if the origin of tasks1 is overlapped by task tasks2.
- The set generator is SUCC. The final graph will consist of those tasks for which the origin is covered by at least one task and of those corresponding tasks.
- The dynamic constraint on a set forces the sum of the heights of the tasks that belong to a successor set to not exceed LIMIT.
2.2. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF GRAPH PROPERTIES

Figure 2.6: Initial and final graph of an instance of the cumulative constraint

Parts (A) and (B) of Figure 2.6 respectively show the initial and the final graph corresponding to the following instance:

\[
\text{cumulative}((\text{origin} - 1 \text{ duration} - 3 \text{ height} - 1, \\
\text{origin} - 2 \text{ duration} - 9 \text{ height} - 2, \\
\text{origin} - 3 \text{ duration} - 10 \text{ height} - 1, \\
\text{origin} - 6 \text{ duration} - 6 \text{ height} - 1, \\
\text{origin} - 7 \text{ duration} - 2 \text{ height} - 3), 8).
\]

We label the vertices of the initial and final graph by giving the key of the corresponding task. On both graphs the edges are oriented from left to right. On the final graph we consider the sets that consist of the successors of the different vertices; those are the sets of tasks \{1\}, \{1, 2\}, \{1, 2, 3\}, \{2, 3, 4\} and \{2, 3, 4, 5\}. Since the \textsc{Succ} set generator uses a derived collection that only considers the height attribute of a task, these sets respectively correspond to the following collection of items:

- \{\text{var} - 1\},
- \{\text{var} - 1, \text{var} - 2\},
- \{\text{var} - 1, \text{var} - 2, \text{var} - 1\},
- \{\text{var} - 2, \text{var} - 1, \text{var} - 1\},
- \{\text{var} - 2, \text{var} - 1, \text{var} - 1, \text{var} - 3\}.

The cumulative constraint holds since, for each successors set, the corresponding constraint holds:

- \text{sum}_\text{ctr}((\text{var} - 1), \leq, 8),
- \text{sum}_\text{ctr}((\text{var} - 1, \text{var} - 2), \leq, 8),
- \text{sum}_\text{ctr}((\text{var} - 1, \text{var} - 2, \text{var} - 1), \leq, 8),
- \text{sum}_\text{ctr}((\text{var} - 2, \text{var} - 1, \text{var} - 1), \leq, 8),
- \text{sum}_\text{ctr}((\text{var} - 2, \text{var} - 1, \text{var} - 1, \text{var} - 3), \leq, 8).

The \text{sum}_\text{ctr}(\text{VARIABLES}, \text{CTR}, \text{VAR}) constraint holds if the sum \(S\) of the variables of the \text{VARIABLES} collection satisfies \(S \text{ CT} \text{R} \text{ VARIABLES}\), where \text{CTR} is a comparison operator.

\footnote{\text{key} is an implicit attribute corresponding to the position of an item within a collection that was introduced in Section 2.1.2 on page 8.}
2.3 Describing global constraints in terms of automata

This section is based on the article describing global constraint in terms of automata \[34\]. The main difference with the original article is the introduction of array of counters within the description of an automaton. We consider global constraints for which any ground instance can be checked in linear time by scanning once through their variables without using any data structure, except counters or arrays of counters. In order to concretely illustrate this point we first select a set of global constraints and write down a checker for each of them. Finally, we give for each checker a sketch of the corresponding automaton. Based on these observations, we define the type of automaton we use in the catalogue.

2.3.1 Selecting an appropriate description

As we previously said, we focus on those global constraints that can be checked by scanning once through their variables. This is for instance the case of:

- \texttt{element} \[393\],
- \texttt{minimum} \[26\],
- \texttt{pattern} \[78\],
- \texttt{global_contiguity} \[252\],
- \texttt{lex.lesseq} \[160\],
- \texttt{among} \[39\],
- \texttt{inflexion} \[24\],
- \texttt{alldifferent} \[320\].

Since they illustrate key points needed for characterising the set of solutions associated with a global constraint, our discussion will be based on the last five constraints for which we now recall the definition:

- The \texttt{global_contiguity}(vars) constraint forces the sequence of 0-1 variables \texttt{vars} to have at most one group of consecutive 1. For instance, the constraint \texttt{global_contiguity}((0, 1, 1, 0)) holds since we have only one group of consecutive 1.

- The lexicographic ordering constraint \( \vec{x} \leq_{\text{lex}} \vec{y} \) (see \texttt{lex.lesseq}) over two vectors of variables \( \vec{x} = \langle x_0, \ldots, x_{n-1} \rangle \) and \( \vec{y} = \langle y_0, \ldots, y_{n-1} \rangle \) holds if and only if \( n = 0 \) or \( x_0 < y_0 \) or \( x_0 = y_0 \) and \( \langle x_1, \ldots, x_{n-1} \rangle \leq_{\text{lex}} \langle y_1, \ldots, y_{n-1} \rangle \).

- The \texttt{among}(nvar, vars, values) constraint restricts the number of variables of the sequence of variables \texttt{vars} that take their value in a given set \texttt{values} to be equal to the variable \texttt{nvar}. For instance, \texttt{among}(3, \langle 4, 5, 5, 4, 1 \rangle, \langle 1, 5, 8 \rangle) holds since exactly 3 values of the sequence 45541 are located in the set of values \{1, 5, 8\}.

- The \texttt{inflexion}(ninf, vars) constraint forces the number of inflexions of the sequence of variables \texttt{vars} to be equal to the variable \texttt{ninf}. An \texttt{inflexion} is described by one of the following patterns: a strict increase followed by a strict decrease or, conversely, a strict decrease followed by a strict increase. For instance,
inflexion \((4, (3, 3, 1, 4, 5, 5, 6, 5, 6, 3))\) holds since we can extract from the sequence \(3314556563\) the four subsequences \(314, 565, 6556\) and \(563\), which all follow one of these two patterns.

- The **alldifferent** \((\text{vars})\) constraint forces all pairs of distinct variables of the collection \(\text{vars}\) to take distinct values. For instance **alldifferent** \((6, 1, 5, 9)\) holds since we have four distinct values.

```plaintext
global_contiguity(\text{vars}[0..n-1]):\text{BOOLEAN}
1 BEGIN
2 i=0;
3 WHILE i<n AND \text{vars}[i]=0 DO i++;
4 RETURN (i=n);
7 END.

lex_lesseq(x[0..n-1],y[0..n-1]):\text{BOOLEAN}
1 BEGIN
2 i=0;
3 WHILE i<n AND x[i]=y[i] DO i++;
4 RETURN (i=n OR x[i]<y[i]);
5 END.

among(nvar,\text{vars}[0..n-1],\text{values}):\text{BOOLEAN}
1 BEGIN
2 i=0; c=0;
3 WHILE i<n DO
4 IF \text{vars}[i] \in \text{values} THEN c++;
5 i++;
6 RETURN (nvar=c);
7 END.

inflection(ninf,\text{vars}[0..n-1]):\text{BOOLEAN}
1 BEGIN
2 i=0; c=0;
3 WHILE i<n−1 AND \text{vars}[i]=\text{vars}[i+1] DO i++;
4 IF i<n−1 THEN less=(\text{vars}[i]<\text{vars}[i+1]);
5 WHILE i<n−1 DO
6 IF less THEN
7 IF \text{vars}[i] > \text{vars}[i+1] THEN c++; less=FALSE;
8 ELSE
9 IF \text{vars}[i] < \text{vars}[i+1] THEN c++; less=TRUE;
10 i++;
11 RETURN (ninf=c);
12 END.

alldifferent(\text{vars}[0..n-1]):\text{BOOLEAN}
1 BEGIN
2 u=\text{vars}[0]; v=\text{vars}[0]; i=1;
3 WHILE i<n DO
4 IF \text{vars}[i]+u THEN \text{vars} \leftarrow \text{vars}[i];
5 IF \text{vars}[i]+v THEN \text{vars} \leftarrow \text{vars}[i];
6 i++;
7 FOR i=u TO v DO c[i]=0;
8 FOR i=0 TO n−1 DO c[\text{vars}[i]]+c[\text{vars}[i]+1];
9 IF i=\text{vars}[i] THEN RETURN FALSE;
10 RETURN TRUE;
11 END.
```

Figure 2.7: Five checkers and their corresponding automata

Parts (A1), (B1), (C1), (D1) and (E1) of Figure 2.7 depict the five checkers respectively associated with **global_contiguity**, with **lex_lesseq**, with **among**, with
inflexion and with alldifferent. For each checker we observe the following facts:

- Within the checker depicted by part (A1) of Figure 2.7, the values of the sequence \( \text{vars}[0], \ldots, \text{vars}[n-1] \) are successively compared against 0 and 1 in order to check that we have at most one group of consecutive 1. This can be translated to the automaton depicted by part (A2) of Figure 2.7. The automaton takes as input the sequence \( \text{vars}[0], \ldots, \text{vars}[n-1] \), and triggers successively a transition for each term of this sequence. Transitions labelled by 0, 1 and $ are respectively associated with the conditions \( \text{vars}[i] = 0, \text{vars}[i] = 1 \) and \( i = n \). Transitions leading to failure are systematically skipped. This is why no transition labelled with a 1 starts from state \( z \).

- Within the checker given by part (B1) of Figure 2.7, the components of vectors \( \text{x} \) and \( \text{y} \) are scanned in parallel. We first skip all the components that are equal and then perform a final check. This is represented by the automaton depicted by part (B2) of Figure 2.7. The automaton takes as input the sequence \( \langle x[0], y[0] \rangle, \ldots, \langle x[n-1], y[n-1] \rangle \) and triggers a transition for each term of this sequence. Unlike the global contiguity constraint, some transitions now correspond to a condition (e.g., \( x[i] = y[i], x[i] < y[i] \)) between two variables of the lex_lesseq constraint.

- Note that the among(nvar, vars, values) constraint involves a variable \( nvar \) whose value is computed from a given collection of variables \( \text{vars} \). The checker depicted by part (C1) of Figure 2.7 counts the number of variables of \( \text{vars}[0], \ldots, \text{vars}[n-1] \) that take their value in \( \text{values} \). For this purpose it uses a counter \( c \), which is eventually tested against the value of \( nvar \). This convinced us to allow the use of counters in an automaton. Each counter has an initial value, which can be updated while triggering certain transitions. The final state of an automaton can force a variable of the constraint to be equal to a given counter. Part (C2) of Figure 2.7 describes the automaton corresponding to the code given in part (C1) of the same figure. The automaton uses the counter variable \( c \) initially set to 0 and takes as input the sequence \( \text{vars}[0], \ldots, \text{vars}[n-1] \). It triggers a transition for each variable of this sequence and increments \( c \) when the corresponding variable takes its value in \( \text{values} \). The final state returns a success when the value of \( c \) is equal to \( nvar \). At this point we want to stress the following fact: it would have been possible to use an automaton that avoids the use of counters. However, this automaton would depend on the effective value of the argument \( nvar \). In addition, it would require more states than the automaton of part (C2) of Figure 2.7. This is typically a problem if we want to have a fixed number of states in order to save memory as well as time.

- As the among constraint, the inflexion(ninf, vars) constraint involves a variable \( ninf \) whose value is computed from a given sequence of variables \( \text{vars}[0], \ldots, \text{vars}[n-1] \). Therefore, the checker depicted in part (D1) of Figure 2.7 uses also a counter \( c \) for counting the number of inflexions, and compares its final value to the \( ninf \) argument. The automaton depicted by part (D2) of Figure 2.7 represents this program. It takes as input the sequence of pairs
2.3. DESCRIBING GLOBAL CONSTRAINTS IN TERMS OF AUTOMATA

\[ \langle \text{vars}[0], \text{vars}[1] \rangle, \langle \text{vars}[1], \text{vars}[2] \rangle, \ldots, \langle \text{vars}[n - 2], \text{vars}[n - 1] \rangle \]

and triggers a transition for each pair. Note that a given variable may occur in more than one pair. Each transition compares the respective values of two consecutive variables of \( \text{vars}[0..n - 1] \) and increments the counter \( c \) when a new inflexion is detected. The final state returns a success when the value of \( c \) is equal to \( \text{ninf} \).

- The checker associated with \textit{alldifferent} is depicted by part (E1) of Figure 2.7. It first initialises an array of counters to 0. The entries of the array correspond to the potential values of the sequence \( \text{vars}[0], \ldots, \text{vars}[n - 1] \). In a second phase the checker computes for each potential value its number of occurrences in the sequence \( \text{vars}[0], \ldots, \text{vars}[n - 1] \). This is done by scanning this sequence. Finally in a third phase the checker verifies that no value is used more than once. These three phases are represented by the automaton depicted by part (E2) of Figure 2.7. The automaton depicted by part (E2) takes as input the sequence \( \text{vars}[0], \ldots, \text{vars}[n - 1] \). Its initial state initialises an array of counters to 0. Then it triggers successively a transition for each element \( \text{vars}[i] \) of the input sequence and increments by 1 the entry corresponding to \( \text{vars}[i] \). The final state checks that all entries of the array of counters are strictly less than 2, which means that no value occurs more than once in the sequence \( \text{vars}[0], \ldots, \text{vars}[n - 1] \).

Synthesising all the observations we got from these examples leads to the following remarks and definitions for a given global constraint \( C \):

- For a given state, no transition can be triggered indicates that the constraint \( C \) does not hold.

- Since all transitions starting from a given state are mutually incompatible all automata are deterministic. Let \( M \) denote the set of mutually incompatible conditions associated with the different transitions of an automaton.

- Let \( S_0, \ldots, S_{m - 1} \) denote the sequence of subsets of variables of \( C \) on which the transitions are successively triggered. All these subsets contain the same number of elements and refer to some variables of \( C \). Since these subsets typically depend on the constraint, we leave the computation of \( S_0, \ldots, S_{m - 1} \) outside the automaton. To each subset \( S_i \) of this sequence corresponds a variable \( S_i \) with an initial domain ranging over \( \lbrack \text{min}, \text{min} + |M| - 1 \rbrack \), where \( \text{min} \) is a fixed integer. To each integer of this range corresponds one of the mutually incompatible conditions of \( M \). The sequences \( S_0, \ldots, S_{m - 1} \) and \( S_0, \ldots, S_{m - 1} \) are respectively called the \textit{signature} and the \textit{signature argument} of the constraint. The constraint between \( S_i \) and the variables of \( S_i \) is called the \textit{signature constraint} and is denoted by \( \Psi_C(S_i, S_i) \).

- From a pragmatic point view, the task of writing a constraint checker is naturally done by writing down an imperative program where local variables, arrays, assignment statements and control structures are used. This suggested us to consider deterministic finite automata augmented with local variables and assignment statements on these variables. Regarding control structures, we did not
introduce any extra feature since the deterministic choice of which transition to trigger next seemed to be good enough.

• Many global constraints involve a variable whose value is computed from a given collection of variables. This convinced us to allow the final state of an automaton to optionally return a result. In practice, this result corresponds to the value of a local variable of the automaton in the final state.

### 2.3.2 Defining an automaton

An automaton $A$ of a global constraint $C$ is defined by

$$\langle \text{Signature}, \text{SignatureDomain}, \text{SignatureArg}, \text{SignatureArgPattern}, \text{Counters}, \text{Arrays}, \text{States}, \text{Transitions} \rangle$$

where:

• **Signature** is the sequence of variables $S_0, \ldots, S_{m-1}$ corresponding to the signature of the constraint $C$.

• **SignatureDomain** is an interval that defines the range of possible values of the variables of **Signature**.

• **SignatureArg** is the signature argument $S_0, \ldots, S_{m-1}$ of the constraint $C$. The link between the variables of $S_i$ and the variable $S_i$ $(0 \leq i < m)$ is done by writing down the signature constraint $\Psi_C(S_i, S_i)$.

• When used, **SignatureArgPattern** defines a symbolic name for each term of **SignatureArg**. These names can be used within the description of a transition for expressing an additional condition for triggering the corresponding transition.

• **Counters** is the, possibly empty, list of all counters used in the automaton $A$. Each counter is described by a term $t(\text{Counter}, \text{InitialValue}, \text{FinalVariable})$ where **Counter** is a symbolic name representing the counter, **InitialValue** is an integer giving the value of the counter in the initial state of $A$, and **FinalVariable** gives the variable that should be unified with the value of the counter in the final state of $A$.

• **Arrays** is the, possibly empty, list of all arrays used in the automaton $A$. Each array is described by a term $t(\text{Array}, \text{InitialValue}, \text{FinalConstraint})$ where **Array** is a symbolic name representing the array, **InitialValue** is an integer giving the value of all the entries of the array in the initial state of $A$, **FinalConstraint** denotes an existing constraint of the catalogue that should hold in the final state of $A$. Arguments of this constraint correspond to collections of variables that are bound to array of counters, or to variables that are bound to counters declared in **Counters**. For an array of counters we only consider those entries that are located between the first and the last entries that were modified while triggering a transition of $A$. 
2.4. REFORMULATING GLOBAL CONSTRAINTS AS A CONJUNCTION

- **States** is the list of states of \( A \), where each state has the form \( \text{source}(id) \), \( \text{sink}(id) \) or \( \text{node}(id) \). \( id \) is a unique identifier associated with each state. Finally, \( \text{source}(id) \) and \( \text{sink}(id) \) respectively denote the initial and the final state of \( A \).

- **Transitions** is the list of transitions of \( A \). Each transition \( t \) has the form \( \text{arc}(id_1, label, id_2) \) or \( \text{arc}(id_1, label, id_2, counters) \). \( id_1 \) and \( id_2 \) respectively correspond to the state just before and just after \( t \), while \( label \) denotes the value that the signature variable should have in order to trigger \( t \). When used, \( counters \) gives for each counter of \( Counts \) its value after firing the corresponding transition. This value is specified by an arithmetic expression involving counters, constants, as well as usual arithmetic functions, such as +, -, min, or max. The order used in the \( counters \) list is identical to the order used in \( Counts \).

**EXAMPLE:** As an illustrative example we give the description of the automaton associated with the \( \text{inflexion}(\text{ninf}, \text{vars}) \) constraint. We have:

- **Signature = \( S_0, S_1, \ldots, S_{n-2} \).**
- **SignatureDomain = 0..2.**
- **SignatureArg = \( \{\text{vars}[0], \text{vars}[1]\}, \ldots, \{\text{vars}[n-2], \text{vars}[n-1]\} \).**
- **SignatureArgPattern is not used,**
- **Counts = \( t(c, 0, \text{ninf}) \),**
- **States = \( [\text{source}(s), \text{node}(i), \text{node}(j), \text{sink}(t)] \),**
- **Transitions = \( \{\text{arc}(s, 1, s), \text{arc}(s, 2, i), \text{arc}(s, 0, j), \text{arc}(s, \$, t), \text{arc}(i, 1, i), \text{arc}(i, 2, i), \text{arc}(i, 0, j, [c + 1]), \text{arc}(i, \$, t), \text{arc}(j, 1, j), \text{arc}(j, 0, j), \text{arc}(j, 2, i, [c + 1]), \text{arc}(j, \$, t)\} \).**

The signature constraint relating each pair of variables \( \{\text{vars}[i], \text{vars}[i+1]\} \) to the signature variable \( S_i \) is defined as follows: \( \Psi_{\text{inflexion}}(S_i, \text{vars}[i], \text{vars}[i+1]) \equiv \text{vars}[i] > \text{vars}[i+1] \Leftrightarrow S_i = 0 \land \text{vars}[i] = \text{vars}[i+1] \Leftrightarrow S_i = 1 \land \text{vars}[i] < \text{vars}[i+1] \Leftrightarrow S_i = 2 \). The sequence of transitions triggered on the ground instance \( \text{inflexion}(4, [3, 3, 1, 4, 5, 5, 5, 6, 3]) \) is \( \frac{\text{arc}(s, 1, s)}{i} \frac{\text{arc}(s, 2, i)}{c=1} \frac{\text{arc}(s, 0, j)}{c=3} \frac{\text{arc}(s, \$, t)}{c=4} \frac{\text{arc}(i, 1, i)}{c=1} \frac{\text{arc}(i, 2, i)}{c=4} \frac{\text{arc}(i, 0, j, [c + 1])}{c=0} \frac{\text{arc}(i, \$, t)}{c=2} \frac{\text{arc}(j, 1, j)}{c=2} \frac{\text{arc}(j, 0, j)}{c=4} \frac{\text{arc}(j, 2, i, [c + 1])}{c=0} \frac{\text{arc}(j, \$, t)}{c=2} \). Each transition gives the corresponding condition and, possibly, the value of the counter \( c \) just after firing that transition.

### 2.4 Reformulating global constraints as a conjunction

Many global constraints can be reformulated as a conjunction of global or reified constraints. The slot **Reformulation** provides for some global constraints such reformulations (see for instance the reformulation slots respectively associated with the **coloured_cumulative** or the **tree** constraints). When it exists, the corresponding code is available in the “.pl file” attached to a constraint. The initial concrete motivation for providing reformulations was triggered by the fact that it is usually an easy way to have a first implementation of a constraint, which is a feature we want to have
in the context of the catalogue. However, many reformulations (e.g., `alldifferent`, `nvalue`, `tree`) involve a quadratic (or even more) number of variables and/or constraints, which does not scale in practice when one wants to handle constraints with thousands of variables. This is why many filtering algorithms compute again and again common quantities that would require too much memory if stored explicitly.

### 2.5 Semantic links between global constraints

For each global constraint entry of the catalogue, the slot `See also` provides links to other global constraints. Rather than just pointing to a set of constraints, we prefer to explicitly indicate the reason why we point to a given constraint. A link `link(C_{entry}, C_{also})` from a constraint `C_{entry}` (i.e., the constraint associated with a catalogue entry) to another constraint `C_{also}` (i.e., the constraint of the `See also` slot located in the catalogue entry of constraint `C_{entry}`) has a given semantics and this section describes the kind of **semantic links** that are currently used. Before introducing each semantic link and its meaning, let us first quote that some of them are related by one of the following relations:

- A link `link` is symmetric if and only if `link(C_1, C_2) ⇔ link(C_2, C_1)`.
- A link `link` is asymmetric if and only if `link(C_1, C_2) ⇒ ¬link(C_2, C_1)` (¬`link(C_2, C_1)` is a shortcut for denoting that the link `link(C_2, C_1)` does not occur in the catalogue).
- A link `link_j` is the converse of a link `link_i` if and only if `link_i(C_1, C_2) ⇔ link_j(C_2, C_1)`.

Table 2.1 lists each semantic link and the relation it has.\(^{31}\) Then one section describes the meaning of each semantic link.

#### 2.5.1 Assignment dimension added

Constraint `C_{also}` corresponds to constraint `C_{entry}` where an **assignment dimension** is added to `C_{entry}`.

**EXAMPLE**: As an example, constraint `C_{also} = cumulatives` corresponds to constraint `C_{entry} = cumulative` where an assignment dimension corresponding to the machine attribute is added (i.e., the constraint `cumulatives` enforces a `cumulative` constraint for each maximum set of tasks that are assigned the same machine).

#### 2.5.2 Assignment dimension removed

Constraint `C_{also}` corresponds to constraint `C_{entry}` where an **assignment dimension** is removed from `C_{entry}`.

\(^{31}\) All links are automatically checked with respect to their relation each time the catalogue is generated.
2.5. SEMANTIC LINKS BETWEEN GLOBAL CONSTRAINTS

<table>
<thead>
<tr>
<th>semantic links</th>
<th>relation between semantic links</th>
</tr>
</thead>
<tbody>
<tr>
<td>assignment dimension added</td>
<td>converse: assignment dimension removed</td>
</tr>
<tr>
<td>assignment dimension removed</td>
<td>converse: assignment dimension added</td>
</tr>
<tr>
<td>attached to cost variant</td>
<td>converse: cost variant</td>
</tr>
<tr>
<td>common keyword</td>
<td>symmetric</td>
</tr>
<tr>
<td>comparison swapped</td>
<td>symmetric</td>
</tr>
<tr>
<td>cost variant</td>
<td>converse: attached to cost variant</td>
</tr>
<tr>
<td>generalisation</td>
<td>converse: specialisation</td>
</tr>
<tr>
<td>hard version</td>
<td>converse: soft variant</td>
</tr>
<tr>
<td>implied by</td>
<td>converse: implies</td>
</tr>
<tr>
<td>implies</td>
<td>converse: implied by</td>
</tr>
<tr>
<td>implies (if swap arguments)</td>
<td>symmetric</td>
</tr>
<tr>
<td>implies (items to collection)</td>
<td>asymmetric</td>
</tr>
<tr>
<td>negation</td>
<td>symmetric</td>
</tr>
<tr>
<td>part of system of constraints</td>
<td>converse: system of constraints</td>
</tr>
<tr>
<td>related</td>
<td>symmetric</td>
</tr>
<tr>
<td>related to a common problem</td>
<td>symmetric</td>
</tr>
<tr>
<td>root concept</td>
<td>converse: shift of concept</td>
</tr>
<tr>
<td>shift of concept</td>
<td>converse: root concept</td>
</tr>
<tr>
<td>soft variant</td>
<td>converse: hard version</td>
</tr>
<tr>
<td>specialisation</td>
<td>converse: generalisation</td>
</tr>
<tr>
<td>system of constraints</td>
<td>converse: part of system of constraints</td>
</tr>
<tr>
<td>used in graph description</td>
<td>asymmetric</td>
</tr>
<tr>
<td>used in reformulation</td>
<td>converse: uses in its reformulation</td>
</tr>
<tr>
<td>uses in its reformulation</td>
<td>converse: used in reformulation</td>
</tr>
</tbody>
</table>

Table 2.1: Available semantic links between constraints

**EXAMPLE:** As an example, constraint \( C_{also} = \text{among}_{low,up} \) corresponds to constraint \( C_{entry} = \text{interval} \text{and} \text{count} \) where an assignment dimension corresponding to the origin attribute is removed from \( C_{entry} = \text{interval} \text{and} \text{count} \) (i.e., the constraint \( \text{interval} \text{and} \text{count} \) enforces a \( \text{among}_{low,up} \) constraint for each maximum set of tasks for which the origin is assigned the same interval \( [k \cdot \text{SIZE} \text{INTERVAL}, k \cdot \text{SIZE} \text{INTERVAL} + \text{SIZE} \text{INTERVAL} - 1] \) (\( \text{SIZE} \text{INTERVAL} \) is the last argument of \( \text{interval} \text{and} \text{count} \)).

2.5.3 Attached to cost variant

Constraint \( C_{also} \) is the original version attached to the cost variant constraint \( C_{entry} \).

**EXAMPLE:** As an example, constraint \( C_{also} = \text{alldifferent} \) is the original version attached to the cost variant constraint \( C_{entry} = \text{minimum}\_\text{weight}\_\text{alldifferent} \), where the total cost of a solution is the sum of the costs associated with the fact that we assign a given value to a specific variable.
2.5.4 Common keyword

Constraints $C_{entry}$ and $C_{also}$ share one or more common keywords with a strong semantic connotation.

**EXAMPLE:** As an example, constraints $C_{entry} = \text{tree}$ and $C_{also} = \text{cycle}$ are both graph partitioning constraints (i.e., constraints that partition the vertices of a given initial digraph so that each partition corresponds to a specific pattern, a tree and a circuit in this example).

2.5.5 Comparison swapped

Constraint $C_{also}$ corresponds to constraint $C_{entry}$ where one of the following conditions holds:

- The comparison operator $\geq$ is swapped to $\leq$ or, conversely, $\leq$ is swapped to $\geq$.
- The comparison operator $>$ is swapped to $<$ or, conversely, $<$ is swapped to $>$.  

**EXAMPLE:** Constraint $C_{also} = \text{atmost}$ corresponds to constraint $C_{entry} = \text{atleast}$ where the comparison $\leq N$ for expressing that we should not exceed a given threshold (i.e., restricts the maximum number of occurrences for a given value) is replaced by $\geq N$ for expressing that we should reach a given threshold (i.e., enforces a minimum number of occurrences for a given value).

2.5.6 Cost variant

Constraint $C_{also}$ is a cost variant of constraint $C_{entry}$.

**EXAMPLE:** As an example, constraint $C_{also} = \text{sum of weights of distinct values}$ is the cost variant of constraint $C_{entry} = n\text{value}$, where we introduce a weight for each value and we replace the number of distinct values by the sum of weights associated with distinct values.

2.5.7 Generalisation

Denotes that constraint $C_{also}$ is a generalisation of constraint $C_{entry}$.

**EXAMPLE:** As an example, constraint $C_{also} = \text{all min dist}$ is a generalisation of constraint $C_{entry} = \text{alldifferent}$ where we replace a disequality between two variables by the fact that two line-segments of same length do not overlap.

2.5.8 Hard version

Constraint $C_{also}$ is a hard version of constraint $C_{entry}$ (i.e., constraint $C_{entry}$ is a soft variant of constraint $C_{also}$).

**EXAMPLE:** As an example, constraint $C_{also} = \text{alldifferent}$ is a hard version of constraint $C_{entry} = \text{soft alldifferent}$, which restricts the minimum number of variables that should be unassigned in order that all variables take a distinct value.
2.5.9 **Implied by**

If constraint \( C_{\text{also}} \) holds and if all restrictions of constraint \( C_{\text{entry}} \) hold then constraint \( C_{\text{entry}} \) also holds. Note that we try to restrict ourselves to the transitive reduction of the implication graph between constraints.

**EXAMPLE:** As an example, constraint \( C_{\text{entry}} = \text{minimum} \) is implied by constraint \( C_{\text{also}} = \text{and} \).

2.5.10 **Implies**

If constraint \( C_{\text{entry}} \) holds and if all restrictions of constraint \( C_{\text{also}} \) hold then constraint \( C_{\text{also}} \) also holds. Note that we also consider all the implications depicted in the implication graphs mentioned in the tables associated with the normalised signature tree of global constraints arguments. For an example of such table see Table 3.1.

**EXAMPLE:** As an example, constraint \( C_{\text{entry}} = \text{alldifferent} \) implies constraint \( C_{\text{also}} = \text{not\_all\_equal} \). Note that the case of an \( \text{alldifferent} \) constraint with one single variable does not imply a \( \text{not\_all\_equal} \) constraint since its restriction (i.e., the number of variables of a \( \text{not\_all\_equal} \) constraint should be strictly greater than one) does not hold.

2.5.11 **Implies (if swap arguments)**

Given two constraints \( C_{\text{entry}} \) and \( C_{\text{also}} \) that both have two arguments, if constraint \( C_{\text{entry}}(\text{arg}_1, \text{arg}_2) \) holds then constraint \( C_{\text{also}}(\text{arg}_2, \text{arg}_1) \) also holds.

**EXAMPLE:** As an example, we can go from constraint \( C_{\text{entry}} = \text{lex\_lesseq} \) to constraint \( \text{lex\_greatereq} \) if we swap the two arguments of constraint \( \text{lex\_lesseq} \).

2.5.12 **Implies (items to collection)**

Given two constraints \( C_{\text{entry}} \) and \( C_{\text{also}} \) where:

- \( C_{\text{entry}} \) has one single argument \( \text{arg}_1 \) corresponding to a collection of \( k \) items, each attribute of type \text{int} or \text{dvar}.

- \( C_{\text{also}} \) has one single argument \( \text{arg}_2 \) corresponding to a collection of collections of \text{dvar}, each of them having the same number of items \( k \).

If constraint \( C_{\text{entry}}(\text{arg}_1) \) holds then constraint \( C_{\text{also}}(\text{arg}_2) \) also holds.

**EXAMPLE:** As an example, we can go from constraint \( C_{\text{entry}} = \text{circuit} \) to constraint \( \text{lex\_alldifferent} \) if we create for each item “index \( i \) succ \( s \)” of the \text{circuit} constraint a collection \( \langle \text{var} - i, \text{var} - s \rangle \).
CHAPTER 2. DESCRIBING GLOBAL CONSTRAINTS

2.5.13 Negation

If constraint $C_{\text{entry}}$ holds then constraint $C_{\text{also}}$ does not hold. Reciprocally, if constraint $C_{\text{also}}$ holds then constraint $C_{\text{entry}}$ does not hold. Note that constraints $C_{\text{entry}}$ and $C_{\text{also}}$ must also have exactly the same parameters, but not necessarily the same parameters restrictions.

**EXAMPLE:** As an example, the constraint $C_{\text{also}} = \text{not all equal}$ (i.e., prevent all variables to be assigned the same value) is the negation of constraint $C_{\text{entry}} = \text{all equal}$ (i.e., enforce all variables to be assigned the same value).

Note that negation is also directly available for constraints which are defined by:

- One single counter free automaton, see keyword `automaton without counters`.
- One single automaton with counter, see keyword `automaton with counters`.
- A set of functional dependencies, see keyword `pure functional dependency`.

2.5.14 Part of system of constraints

Denotes that a constraint $C_{\text{entry}}$ is a conjunction of constraints $C_{\text{also}}$ (i.e., see the keyword `system of constraints`).

**EXAMPLE:** As an example, the constraint $C_{\text{also}} = \text{neq}$ (i.e., prevent two variables to be assigned the same value) can be used to reformulate the constraint $C_{\text{entry}} = \text{alldifferent}$ (i.e., enforce a set of variables to take distinct values) as a conjunction of $\text{neq}$ constraints.

2.5.15 Related

Denotes that a constraint $C_{\text{entry}}$ and a constraint $C_{\text{also}}$ are related by a specific reason that is not covered by an existing link.

**EXAMPLE:** As an example, the constraint $C_{\text{also}} = \text{tree range}$ (i.e., given a digraph, partition it so that each vertex belongs to one tree for which the difference between the longest and the shortest paths – from a leaf to the root – is restricted) is related to the constraint $C_{\text{entry}} = \text{balance}$ (i.e., given a set of variables, restrict the difference between the number of occurrence of the value that occurs the most and the value that occurs the least) by the fact that, on the one hand the constraint $\text{tree range}$ can express a balanced tree, on the other side the constraint $\text{balance}$ can express a balanced assignment.

2.5.16 Related to a common problem

Denotes that a constraint $C_{\text{entry}}$ and a constraint $C_{\text{also}}$ are related to a same problem (i.e., they can both be used for modelling that problem).

**EXAMPLE:** As an example, the constraints $C_{\text{entry}} = \text{colored matrix}$ and $C_{\text{also}} = \text{same}$ can both be used for modelling the matrix reconstruction problem.
2.5. SEMANTIC LINKS BETWEEN GLOBAL CONSTRAINTS

2.5.17 Root concept
Constraint $C_{entry}$ is derived from constraint $C_{also}$.

**EXAMPLE:** As an example, the constraint $C_{entry} = \text{tree\_resource}$ is derived from the constraint $C_{also} = \text{tree}$. Given a digraph $G$, the \text{tree} constraint enforces a partitioning of $G$ by a set of trees in such a way that each vertex of $G$ belongs to one distinct tree. In addition, the \text{tree\_resource} constraint distinguishes \text{resource} and \text{task} vertices, and enforces each tree to contain exactly one resource vertex.

2.5.18 Shift of concept
Constraint $C_{also}$ is derived from constraint $C_{entry}$.

**EXAMPLE:** As an example, constraint $C_{also} = \text{global\_cardinality\_no\_loop}(N\text{LOOP}, \text{VARIABLES}, \text{VALUES})$ is derived from constraint $C_{entry} = \text{global\_cardinality}(\text{VARIABLES}, \text{VALUES})$ (i.e., each value $\text{VALUES}[i].\text{val}$ should be taken by exactly $\text{VALUES}[i].\text{val}$ variables of the $\text{VARIABLES}$ collection) by discarding all variables such that $\text{VARIABLES}[i].\text{var} = i$.

2.5.19 Soft variant
Constraint $C_{also}$ is a soft variant of constraint $C_{entry}$. Note that, from an academic point of view, a soft constraint $C_{also} =$ is usually defined with a cost variable that quantifies how much the constraint $C_{entry} =$ is violated. We exceptionally breaks this rule when it seems to make sense from an application point of view. For instance, within the \text{alldifferent} constraint, we reference the \text{alldifferent\_except\_0} since it can be seen as a kind of relaxation of the \text{alldifferent} constraint where we allow to use value 0 several times.

**EXAMPLE:** As an example, one of the possible soft variant of constraint $C_{entry} = \text{alldifferent}$ (i.e., the \text{alldifferent} constraint enforces all variables of a collection to take distinct values) is the constraint $C_{also} = \text{soft\_alldifferent\_var}$, where the cost is the minimum number of variables that need to be unassigned to satisfy the \text{alldifferent} constraint.

2.5.20 Specialisation
Denotes that constraint $C_{also}$ is a specialisation of constraint $C_{entry}$.

**EXAMPLE:** As an example, constraint $C_{also} = \text{path}$ is a specialisation of constraint $C_{entry} = \text{tree}$. Given a digraph $G$, the \text{tree} constraint enforces a partitioning of $G$ by a set of trees in such a way that each vertex of $G$ belongs to one distinct tree. If, in addition, we restrict each vertex to have at most one child we get the \text{path} constraint.

2.5.21 System of constraints
Denotes that a constraint $C_{also}$ is a conjunction of constraints $C_{entry}$ (see the keyword system of constraints).
EXAMPLE: As an example, the constraint $C_{also} = \text{colored\_matrix}$ corresponds to a conjunction of constraints of the form $C_{entry} = \text{global\_cardinality}$. Given a matrix $\mathcal{M}$ of variables, the \text{colored\_matrix} constraint enforces a \text{global\_cardinality} on each row and each column of $\mathcal{M}$.

### 2.5.22 Used in graph description

Constraint $C_{also}$ is used within a graph based description of constraint $C_{entry}$.

EXAMPLE: As an example, the constraint $C_{also} = \text{two\_orth\_do\_not\_overlap}$, a constraint enforcing two orthotopes to not overlap, is used in the graph based description of the constraint $C_{entry} = \text{diffn}$. Given a collection of orthotopes, the \text{diffn} constraint enforces for each pair of orthotopes $(O_1, O_2)$ that $O_1$ and $O_2$ do not overlap.

\*An orthotope corresponds to the generalisation of a segment, a rectangle and a box to the $n$-dimensional case.

### 2.5.23 Used in reformulation

Constraint $C_{also}$ is used within a reformulation of constraint $C_{entry}$. Since it is already handled by the link part of system of constraints, we do not consider the case where constraint $C_{entry}$ can be expressed as a conjunction of constraints $C_{also}$.

EXAMPLE: As an example, the constraint $C_{also} = \text{open\_minimum}$ is used within the reformulation slot of the constraint $C_{entry} = \text{tree\_range}$.

### 2.5.24 Uses in its reformulation

Constraint $C_{also}$ uses constraint $C_{entry}$ in its reformulation. Since it is already handled by the link system of constraints, we do not consider the case where constraint $C_{also}$ can be expressed as a conjunction of constraints $C_{entry}$.

EXAMPLE: As an example, the reformulation slot of constraint $C_{also} = \text{tree\_range}$ uses the constraint $C_{entry} = \text{open\_minimum}$.
Chapter 3

Description of the Catalogue

Contents

3.1 Which global constraints are included? ......................... 98
3.2 Which global constraints are missing? ......................... 100
3.3 Searching in the catalogue ................................. 100
   3.3.1 How to see if a global constraint is in the catalogue? . . 100
   3.3.2 How to search for all global constraints sharing the same structure . . . 101
   Searching from a graph property perspective .......................... 101
   Searching from an automaton perspective .......................... 101
   Searching from a first order logic perspective ..................... 102
   3.3.3 Searching all places where a global constraint is referenced ... 102
   3.3.4 Searching the mapping with a constraint of a concrete system . . 103
3.4 Figures of the catalogue ..................................... 103
3.5 Constraints argument patterns ............................. 105
   3.5.1 Constraints with 1 argument .............................. 108
   3.5.2 Constraints with 2 arguments .............................. 112
   3.5.3 Constraints with 3 arguments .............................. 123
   3.5.4 Constraints with 4 arguments .............................. 131
   3.5.5 Constraints with 5 arguments .............................. 135
   3.5.6 Constraints with 6 arguments .............................. 136
   3.5.7 Constraints with 8 arguments .............................. 137
   3.5.8 Constraints with 10 arguments .............................. 137
3.6 Meta-keywords attached to the keywords .................. 138
   3.6.1 Application area ........................................... 138
   3.6.2 Characteristic of a constraint .................................. 138
   3.6.3 Combinatorial object ........................................... 139
   3.6.4 Complexity .................................................... 139
   3.6.5 Constraint network structure .................................. 139
   3.6.6 Constraint type ................................................. 140
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6.7</td>
<td>Constraint arguments</td>
<td>140</td>
</tr>
<tr>
<td>3.6.8</td>
<td>Filtering</td>
<td>140</td>
</tr>
<tr>
<td>3.6.9</td>
<td>Final graph structure</td>
<td>141</td>
</tr>
<tr>
<td>3.6.10</td>
<td>Geometry</td>
<td>141</td>
</tr>
<tr>
<td>3.6.11</td>
<td>Heuristics</td>
<td>142</td>
</tr>
<tr>
<td>3.6.12</td>
<td>Miscellaneous</td>
<td>142</td>
</tr>
<tr>
<td>3.6.13</td>
<td>Modelling</td>
<td>142</td>
</tr>
<tr>
<td>3.6.14</td>
<td>Modelling exercises</td>
<td>144</td>
</tr>
<tr>
<td>3.6.15</td>
<td>Problems</td>
<td>145</td>
</tr>
<tr>
<td>3.6.16</td>
<td>Puzzles</td>
<td>145</td>
</tr>
<tr>
<td>3.6.17</td>
<td>Symmetry</td>
<td>146</td>
</tr>
<tr>
<td>3.7</td>
<td>Keywords attached to the global constraints</td>
<td>147</td>
</tr>
<tr>
<td>3.7.1</td>
<td>3-dimensional-matching</td>
<td>147</td>
</tr>
<tr>
<td>3.7.2</td>
<td>3-SAT</td>
<td>147</td>
</tr>
<tr>
<td>3.7.3</td>
<td>Abstract interpretation</td>
<td>147</td>
</tr>
<tr>
<td>3.7.4</td>
<td>Acyclic</td>
<td>148</td>
</tr>
<tr>
<td>3.7.5</td>
<td>Aggregate</td>
<td>148</td>
</tr>
<tr>
<td>3.7.6</td>
<td>Air traffic management</td>
<td>150</td>
</tr>
<tr>
<td>3.7.7</td>
<td>Alignment</td>
<td>150</td>
</tr>
<tr>
<td>3.7.8</td>
<td>All different</td>
<td>151</td>
</tr>
<tr>
<td>3.7.9</td>
<td>Alpha-acyclic constraint network(2)</td>
<td>151</td>
</tr>
<tr>
<td>3.7.10</td>
<td>Alpha-acyclic constraint network(3)</td>
<td>152</td>
</tr>
<tr>
<td>3.7.11</td>
<td>Apartition</td>
<td>152</td>
</tr>
<tr>
<td>3.7.12</td>
<td>Arc-consistency</td>
<td>153</td>
</tr>
<tr>
<td>3.7.13</td>
<td>Arithmetic constraint</td>
<td>155</td>
</tr>
<tr>
<td>3.7.14</td>
<td>Array constraint</td>
<td>155</td>
</tr>
<tr>
<td>3.7.15</td>
<td>Assigning and scheduling tasks that run in parallel</td>
<td>156</td>
</tr>
<tr>
<td>3.7.16</td>
<td>Assignment</td>
<td>159</td>
</tr>
<tr>
<td>3.7.17</td>
<td>Assignment dimension</td>
<td>160</td>
</tr>
<tr>
<td>3.7.18</td>
<td>Assignment to the same set of values</td>
<td>163</td>
</tr>
<tr>
<td>3.7.19</td>
<td>At least</td>
<td>168</td>
</tr>
<tr>
<td>3.7.20</td>
<td>At most</td>
<td>168</td>
</tr>
<tr>
<td>3.7.21</td>
<td>Automaton</td>
<td>168</td>
</tr>
<tr>
<td>3.7.22</td>
<td>Automaton with array of counters</td>
<td>171</td>
</tr>
<tr>
<td>3.7.23</td>
<td>Automaton with counters</td>
<td>171</td>
</tr>
<tr>
<td>3.7.24</td>
<td>Automaton without counters</td>
<td>172</td>
</tr>
<tr>
<td>3.7.25</td>
<td>Autoref</td>
<td>173</td>
</tr>
<tr>
<td>3.7.26</td>
<td>Balanced assignment</td>
<td>173</td>
</tr>
<tr>
<td>3.7.27</td>
<td>Balanced tree</td>
<td>174</td>
</tr>
<tr>
<td>3.7.28</td>
<td>Berge-acyclic constraint network</td>
<td>174</td>
</tr>
<tr>
<td>3.7.29</td>
<td>Binary constraint</td>
<td>177</td>
</tr>
<tr>
<td>3.7.30</td>
<td>Bioinformatics</td>
<td>177</td>
</tr>
<tr>
<td>3.7.31</td>
<td>Bipartite</td>
<td>178</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>3.7.32</td>
<td>Bipartite matching</td>
<td></td>
</tr>
<tr>
<td>3.7.33</td>
<td>Bipartite matching in convex bipartite graphs</td>
<td></td>
</tr>
<tr>
<td>3.7.34</td>
<td>Boolean channel</td>
<td></td>
</tr>
<tr>
<td>3.7.35</td>
<td>Boolean constraint</td>
<td></td>
</tr>
<tr>
<td>3.7.36</td>
<td>Border</td>
<td></td>
</tr>
<tr>
<td>3.7.37</td>
<td>Bound-consistency</td>
<td></td>
</tr>
<tr>
<td>3.7.38</td>
<td>Business rules</td>
<td></td>
</tr>
<tr>
<td>3.7.39</td>
<td>Centered cyclic(1) constraint network(1)</td>
<td></td>
</tr>
<tr>
<td>3.7.40</td>
<td>Centered cyclic(2) constraint network(1)</td>
<td></td>
</tr>
<tr>
<td>3.7.41</td>
<td>Centered cyclic(3) constraint network(1)</td>
<td></td>
</tr>
<tr>
<td>3.7.42</td>
<td>Channel routing</td>
<td></td>
</tr>
<tr>
<td>3.7.43</td>
<td>Channelling constraint</td>
<td></td>
</tr>
<tr>
<td>3.7.44</td>
<td>Circuit</td>
<td></td>
</tr>
<tr>
<td>3.7.45</td>
<td>Circular sliding cyclic(1) constraint network(2)</td>
<td></td>
</tr>
<tr>
<td>3.7.46</td>
<td>Cluster</td>
<td></td>
</tr>
<tr>
<td>3.7.47</td>
<td>Coloured</td>
<td></td>
</tr>
<tr>
<td>3.7.48</td>
<td>Compulsory part</td>
<td></td>
</tr>
<tr>
<td>3.7.49</td>
<td>Conditional constraint</td>
<td></td>
</tr>
<tr>
<td>3.7.50</td>
<td>Configuration problem</td>
<td></td>
</tr>
<tr>
<td>3.7.51</td>
<td>Connected component</td>
<td></td>
</tr>
<tr>
<td>3.7.52</td>
<td>Consecutive loops are connected</td>
<td></td>
</tr>
<tr>
<td>3.7.53</td>
<td>Consecutive values</td>
<td></td>
</tr>
<tr>
<td>3.7.54</td>
<td>Constraint between two collections of variables</td>
<td></td>
</tr>
<tr>
<td>3.7.55</td>
<td>Constraint between three collections of variables</td>
<td></td>
</tr>
<tr>
<td>3.7.56</td>
<td>Constraint involving set variables</td>
<td></td>
</tr>
<tr>
<td>3.7.57</td>
<td>Constraint on the intersection</td>
<td></td>
</tr>
<tr>
<td>3.7.58</td>
<td>Constructive disjunction</td>
<td></td>
</tr>
<tr>
<td>3.7.59</td>
<td>Contact</td>
<td></td>
</tr>
<tr>
<td>3.7.60</td>
<td>Contractible</td>
<td></td>
</tr>
<tr>
<td>3.7.61</td>
<td>Convex</td>
<td></td>
</tr>
<tr>
<td>3.7.62</td>
<td>Convex bipartite graph</td>
<td></td>
</tr>
<tr>
<td>3.7.63</td>
<td>Convex hull relaxation</td>
<td></td>
</tr>
<tr>
<td>3.7.64</td>
<td>Conway packing problem</td>
<td></td>
</tr>
<tr>
<td>3.7.65</td>
<td>Core</td>
<td></td>
</tr>
<tr>
<td>3.7.66</td>
<td>Costas arrays</td>
<td></td>
</tr>
<tr>
<td>3.7.67</td>
<td>Cost filtering constraint</td>
<td></td>
</tr>
<tr>
<td>3.7.68</td>
<td>Cost matrix</td>
<td></td>
</tr>
<tr>
<td>3.7.69</td>
<td>Counting constraint</td>
<td></td>
</tr>
<tr>
<td>3.7.70</td>
<td>Cumulative longest hole problems</td>
<td></td>
</tr>
<tr>
<td>3.7.71</td>
<td>Cycle</td>
<td></td>
</tr>
<tr>
<td>3.7.72</td>
<td>Cyclic</td>
<td></td>
</tr>
<tr>
<td>3.7.73</td>
<td>Data constraint</td>
<td></td>
</tr>
<tr>
<td>3.7.74</td>
<td>Deadlock breaking</td>
<td></td>
</tr>
</tbody>
</table>
### 3.7.75 Decomposition

Page 209

### 3.7.76 Decomposition-based violation measure

Page 210

### 3.7.77 DFS-bottleneck

Page 210

### 3.7.78 Demand profile

Page 210

### 3.7.79 Degree of diversity of a set of solutions

Page 211

### 3.7.80 Derived collection

Page 214

### 3.7.81 Difference

Page 214

### 3.7.82 Difference between pairs of variables

Page 214

### 3.7.83 Directed acyclic graph

Page 215

### 3.7.84 Disequality

Page 215

### 3.7.85 Disjunction

Page 216

### 3.7.86 Domain channel

Page 216

### 3.7.87 Domain definition

Page 216

### 3.7.88 Dominating queens

Page 216

### 3.7.89 Domination

Page 217

### 3.7.90 Dual model

Page 217

### 3.7.91 Duplicated variables

Page 218

### 3.7.92 Dynamic programming

Page 218

### 3.7.93 Empty intersection

Page 218

### 3.7.94 Entailment

Page 219

### 3.7.95 Equality

Page 219

### 3.7.96 Equality between multisets

Page 220

### 3.7.97 Equivalence

Page 220

### 3.7.98 Euler knight

Page 220

### 3.7.99 Excluded

Page 221

### 3.7.100 Extensible

Page 221

### 3.7.101 Extension

Page 224

### 3.7.102 Facilities location problem

Page 224

### 3.7.103 Floor planning problem

Page 224

### 3.7.104 Flow

Page 227

Flow models for \texttt{alldifferent} and \texttt{open\_alldifferent}

Page 227

Flow models for the \texttt{gcc\_low\_up} and the \texttt{gcc\_low\_up\_no\_loop} constraints

Page 228

Flow models for the \texttt{used\_by} and the \texttt{same} constraints

Page 230

Flow model for the \texttt{same\_and\_global\_cardinality\_low\_up\_constraint}

Page 232

### 3.7.105 Frequency allocation problem

Page 233

### 3.7.106 Functional dependency

Page 233

### 3.7.107 Geometrical constraint

Page 236

### 3.7.108 Golomb ruler

Page 237

### 3.7.109 Graph colouring

Page 237

### 3.7.110 Graph constraint

Page 237

### 3.7.111 Graph partitioning constraint

Page 238

### 3.7.112 Guillotine cut

Page 238

### 3.7.113 Hall interval

Page 238

---

**CHAPTER 3. DESCRIPTION OF THE CATALOGUE**
3.7.114 Hamiltonian ........................................... 239
3.7.115 Heuristics ........................................... 239
3.7.116 Heuristics and Berge-acyclic constraint network .... 239
3.7.117 Heuristics and lexicographical ordering .......... 241
3.7.118 Heuristics for two-dimensional rectangle placement problems . . . . 241
Dual strategy for rectangle placement problems with no slack ..... 242
Strategy that gradually creates a compulsory part ........... 242
3.7.119 Hungarian method for the assignment problem ...... 243
3.7.120 Hybrid-consistency ................................ 243
3.7.121 Hypergraph ........................................ 244
3.7.122 Included ........................................... 244
3.7.123 Inclusion ........................................... 244
3.7.124 Incompatible pairs of values ......................... 245
3.7.125 Indistinguishable values ............................ 245
3.7.126 Interval ............................................. 245
3.7.127 Joker value ........................................ 246
3.7.128 Klee's measure problem ............................ 246
3.7.129 Labelling by increasing cost ......................... 246
3.7.130 Latin square ....................................... 249
3.7.131 Lexicographic order ................................ 249
3.7.132 Limited discrepancy search ......................... 250
3.7.133 Linear programming ................................ 250
3.7.134 Line-segments intersection ......................... 252
3.7.135 Logic ............................................... 252
3.7.136 Logigraph ........................................... 252
3.7.137 Magic hexagon .................................... 254
3.7.138 Magic series ....................................... 255
3.7.139 Magic square ...................................... 255
3.7.140 Matching ........................................... 255
3.7.141 Matrix ............................................. 256
3.7.142 Matrix model ...................................... 256
3.7.143 Matrix symmetry .................................. 256
3.7.144 Maximum ........................................... 257
3.7.145 Maximum clique ................................... 257
3.7.146 Maximum number of occurrences ................. 257
3.7.147 maxint .............................................. 257
3.7.148 Metro .............................................. 258
3.7.149 Minimum ........................................... 260
3.7.150 Minimum cost flow ................................ 261
3.7.151 Minimum feedback vertex set ..................... 262
3.7.152 Minimum hitting set cardinality ................. 262
3.7.153 Minimum number of occurrences ................. 262
3.7.154 Modulo ............................................. 262
3.7.155 Multi-site employee scheduling with calendar constraints . . . . 263
3.7.156 Multiset ............................................. 265
3.7.157 Multiset ordering ................................ 265
3.7.158 No cycle ............................................. 265
3.7.159 No loop .............................................. 265
3.7.160 n-Amazon ........................................... 266
3.7.161 n-queen ............................................. 269
3.7.162 Non-deterministic automaton .................... 269
3.7.163 Non-overlapping .................................... 269
3.7.164 Number of changes ................................. 270
3.7.165 Number of distinct equivalence classes ............. 270
3.7.166 Number of distinct values .......................... 270
3.7.167 Obverse .............................................. 271
3.7.168 One succ ............................................. 271
3.7.169 Open automaton constraint ....................... 272
3.7.170 Open constraint ...................................... 273
3.7.171 Order constraint .................................... 274
3.7.172 Orthotope ............................................. 275
3.7.173 Overlapping alldifferent ............................ 275
3.7.174 Pair ................................................... 275
3.7.175 Packing almost squares ............................. 276
3.7.176 Pallet loading ........................................ 276
3.7.177 Partition .............................................. 277
3.7.178 Path .................................................. 277
3.7.179 Partridge ............................................. 277
3.7.180 Pattern sequencing ................................ 278
3.7.181 Pentomino ........................................... 279
3.7.182 Periodic .............................................. 279
3.7.183 Permutation ......................................... 279
3.7.184 Permutation channel ............................... 280
3.7.185 Phi-tree .............................................. 280
3.7.186 Phylogeny ............................................ 282
3.7.187 Pick-up delivery ..................................... 282
3.7.188 Planarity test ........................................ 282
3.7.189 Polygon .............................................. 282
3.7.190 Positioning constraint ............................. 282
3.7.191 Predefined constraint .............................. 283
3.7.192 Preferences ......................................... 284
3.7.193 Producer-consumer ................................ 284
3.7.194 Product ............................................... 285
3.7.195 Program verification ................................ 285
3.7.196 Proximity constraint ............................... 285
3.7.197 Pure functional dependency ....................... 286
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.7.198</td>
<td>Quadtree</td>
<td>287</td>
</tr>
<tr>
<td>3.7.199</td>
<td>Range</td>
<td>288</td>
</tr>
<tr>
<td>3.7.200</td>
<td>Rank</td>
<td>288</td>
</tr>
<tr>
<td>3.7.201</td>
<td>RCC8</td>
<td>288</td>
</tr>
<tr>
<td>3.7.202</td>
<td>Rectangle clique partition</td>
<td>289</td>
</tr>
<tr>
<td>3.7.203</td>
<td>Regret based heuristics</td>
<td>289</td>
</tr>
<tr>
<td>3.7.204</td>
<td>Regret based heuristics in matrix problems</td>
<td>290</td>
</tr>
<tr>
<td>3.7.205</td>
<td>Reified automaton constraint</td>
<td>290</td>
</tr>
<tr>
<td>3.7.206</td>
<td>Reified constraint</td>
<td>292</td>
</tr>
<tr>
<td>3.7.207</td>
<td>Relation</td>
<td>293</td>
</tr>
<tr>
<td>3.7.208</td>
<td>Relaxation</td>
<td>293</td>
</tr>
<tr>
<td>3.7.209</td>
<td>Relaxation dimension</td>
<td>294</td>
</tr>
<tr>
<td>3.7.210</td>
<td>Resource constraint</td>
<td>295</td>
</tr>
<tr>
<td>3.7.211</td>
<td>Run of a permutation</td>
<td>296</td>
</tr>
<tr>
<td>3.7.212</td>
<td>SAT</td>
<td>296</td>
</tr>
<tr>
<td>3.7.213</td>
<td>Scalar product</td>
<td>297</td>
</tr>
<tr>
<td>3.7.214</td>
<td>Sequence</td>
<td>297</td>
</tr>
<tr>
<td>3.7.215</td>
<td>Sequence dependent set-up</td>
<td>298</td>
</tr>
<tr>
<td>3.7.216</td>
<td>Sequencing with release times and deadlines</td>
<td>299</td>
</tr>
<tr>
<td>3.7.217</td>
<td>Set channel</td>
<td>299</td>
</tr>
<tr>
<td>3.7.218</td>
<td>Set packing</td>
<td>300</td>
</tr>
<tr>
<td>3.7.219</td>
<td>Shikaku</td>
<td>300</td>
</tr>
<tr>
<td>3.7.220</td>
<td>Scheduling constraint</td>
<td>301</td>
</tr>
<tr>
<td>3.7.221</td>
<td>Scheduling with machine choice, calendars and preemption</td>
<td>301</td>
</tr>
<tr>
<td>3.7.222</td>
<td>Shared table</td>
<td>305</td>
</tr>
<tr>
<td>3.7.223</td>
<td>Schur number</td>
<td>306</td>
</tr>
<tr>
<td>3.7.224</td>
<td>SLAM problem</td>
<td>306</td>
</tr>
<tr>
<td>3.7.225</td>
<td>Sliding cyclic(1) constraint network(1)</td>
<td>306</td>
</tr>
<tr>
<td>3.7.226</td>
<td>Sliding cyclic(1) constraint network(2)</td>
<td>307</td>
</tr>
<tr>
<td>3.7.227</td>
<td>Sliding cyclic(1) constraint network(3)</td>
<td>307</td>
</tr>
<tr>
<td>3.7.228</td>
<td>Sliding cyclic(2) constraint network(2)</td>
<td>308</td>
</tr>
<tr>
<td>3.7.229</td>
<td>Sliding sequence constraint</td>
<td>308</td>
</tr>
<tr>
<td>3.7.230</td>
<td>Smallest square for packing consecutive dominoes</td>
<td>309</td>
</tr>
<tr>
<td>3.7.231</td>
<td>Smallest rectangle area</td>
<td>310</td>
</tr>
<tr>
<td>3.7.232</td>
<td>Smallest square for packing rectangles with distinct sizes</td>
<td>312</td>
</tr>
<tr>
<td>3.7.233</td>
<td>Soft constraint</td>
<td>314</td>
</tr>
<tr>
<td>3.7.234</td>
<td>Sort</td>
<td>314</td>
</tr>
<tr>
<td>3.7.235</td>
<td>Sort based reformulation</td>
<td>314</td>
</tr>
<tr>
<td>3.7.236</td>
<td>Sparse functional dependency</td>
<td>315</td>
</tr>
<tr>
<td>3.7.237</td>
<td>Sparse table</td>
<td>315</td>
</tr>
<tr>
<td>3.7.238</td>
<td>Sport timetabling</td>
<td>315</td>
</tr>
<tr>
<td>3.7.239</td>
<td>Squared squares</td>
<td>315</td>
</tr>
<tr>
<td>3.7.240</td>
<td>Statistics</td>
<td>320</td>
</tr>
</tbody>
</table>
3.1 Which global constraints are included?

The global constraints of this catalogue come from the following sources:

- Existing constraint systems like:
  - ALICE [238],
3.1. WHICH GLOBAL CONSTRAINTS ARE INCLUDED?

- CHARME in C [278],
- CHIP [130] in Prolog, C and C++ ([http://www.cosytec.com](http://www.cosytec.com)),
- ECLAIR [380] in Claire,
- FaCile in OCaml ([http://www.recherche.enac.fr/opti/facile/](http://www.recherche.enac.fr/opti/facile/)),
- Ilog Solver [303] in C++ and later in Java ([http://www.ilog.com](http://www.ilog.com)),
- Koalog in Java,
- SICStus [95] in Prolog ([http://www.sics.se/sicstus/](http://www.sics.se/sicstus/)).

When available, the Systems slot of a global constraint entry of the catalogue provides the name of the corresponding global constraint in the context of the Choco, Gecode, JaCoP, MiniZinc, and SICStus systems.

- Constraint programming articles mostly from conferences like:
  - The Principles and Practice of Constraint Programming (CP) ([http://www.informatik.uni-trier.de/~ley/db/conf/cp/index.html](http://www.informatik.uni-trier.de/~ley/db/conf/cp/index.html)),
  - The International Joint Conference on Artificial Intelligence (IJCAI) ([http://www.informatik.uni-trier.de/~ley/db/conf/ijcai/index.html](http://www.informatik.uni-trier.de/~ley/db/conf/ijcai/index.html)),
  - The National Conference on Artificial Intelligence (AAAI) ([http://www.informatik.uni-trier.de/~ley/db/conf/aaai/index.html](http://www.informatik.uni-trier.de/~ley/db/conf/aaai/index.html)),
  - The International Conference on Logic Programming (ICLP) ([http://www.informatik.uni-trier.de/~ley/db/conf/iclp/index.html](http://www.informatik.uni-trier.de/~ley/db/conf/iclp/index.html)),
  - The International Conference of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CPAIOR) ([http://www.informatik.uni-trier.de/~ley/db/conf/cpaior/](http://www.informatik.uni-trier.de/~ley/db/conf/cpaior/)).

- Graph constraints from the CP(Graph) computation domain [131].

- New constraints inspired by variations of existing constraints, practical applications, combinatorial problems, puzzles or discussions with colleagues.
3.2 Which global constraints are missing?

Constraints with too many arguments (like for instance the original cycle [117] constraint with 16 arguments), which are in fact a combination of several constraints, were not directly put into the catalogue. Constraints that have complex arguments were also omitted. Besides this, the following constraints should be added in some future version of the catalogue: alldifferent_on_multisets [315] [316], case [94, 89], [107, 108], choquet [202], cost_regular [125], cumulative_trapeze [300, 51], deviation [350, 348], inequality_sum [334, 335], minimum_spanning_tree [132, 329], no_cycle [99], range [59, 61], regular [286] [118], soft_gcc_val [399, 400, 420, 351], soft_gcc_var [399, 400, 420], soft_regular [399], spread [287, 349], multicost_regular [263], pref_alldifferent_var (i.e., variable-based relaxation of alldifferent with preferences) [265, page 100], pref_alldifferent_ctr (i.e., decomposition-based relaxation of alldifferent with preferences) [265, page 103], pref_global_cardinality_low_up_var (i.e., variable-based relaxation of global_cardinality_low_up with preferences) [265, page 123], pref_global_cardinality_low_up_ctr (i.e., decomposition-based relaxation of global_cardinality_low_up with preferences) [265, page 126]. Finally we only consider a restricted number of constraints involving set variables since this is a relatively new area, which is currently growing rapidly since 2003.

3.3 Searching in the catalogue

3.3.1 How to see if a global constraint is in the catalogue?

Searching a given global constraint through the catalogue can be achieved in the following ways:

- If you have an idea of the name of the global constraint you are looking for, then put all its letters in lower case, separate distinct words by an underscore and search the resulting name in the index. Within the pdf document, the entry of the catalogue where the constraint is defined is shown in **bold**. Common abbreviations, synonyms and usual names found in articles have also been put in the index in **bold and italic**.

- If you do not know the name of the global constraint you are looking for, but you know the types of its arguments then Section 3.5 lists the different argument patterns and the corresponding global constraints.

- You can also search a global constraint through the list of keywords that is attached to each global constraint. All available keywords are listed alphabetically in Section 3.7 on page 147. For each keyword we give the list of global constraints using the corresponding keyword as well as the definition of the keyword.
In order to make it possible to search for all keywords related to a specific area, we have also attached to each keyword one, or exceptionally two, meta-keywords. For instance, if you are searching for global constraints that are mentioning puzzles, you first look to the meta-keyword *Puzzles* where you find the keywords corresponding to puzzles (i.e., *Autoref, Conway packing problem, ..., Sudoku, Zebra puzzle*). Then as previously described, for each keyword you can access to the corresponding global constraints. All available meta-keywords are listed alphabetically in Section 3.6 on page 138. For each meta-keyword it first gives the list of keywords using the corresponding meta-keyword and then defines the meta-keyword.

### 3.3.2 How to search for all global constraints sharing the same structure

Since we have three ways of defining global constraints (e.g., searching for a graph with specific properties, coming up with an automaton that only recognises the solutions associated with the global constraint or using a first order logic formula) we can look to the global constraints from these three perspectives.

**Searching from a graph property perspective**

The index contains all the arc generators as well as all the graph properties and the pages where they are mentioned.\(^1\) This allows for finding all global constraints that use a given arc generator or a given graph property in their definition. You can further restrict your search to those global constraints using a specific combination of arc generators and graph properties. All these combinations are listed at the “signature” entry of the index. Within these combinations, a graph property with an underline means that the constraint should be evaluated each time the minimum of this graph property increases. Similarly a graph property with an overline indicates that the constraint should be evaluated each time the maximum of this graph property decreases. For instance if we look for those constraints that both use the *CLIQUE* arc generator as well as the *NARC* graph-property we find the *inverse* and *place in pyramid* constraints. Since *NARC* is underlined and overlined these constraints will have to be woken each time the minimum or the maximum of *NARC* changes. The signature associated with a global constraint is also shown in the header of the even pages corresponding to the description of the global constraint.

**Searching from an automaton perspective**

We have created the following list of keywords, which allow for finding all global constraints defined by a specific type of automaton that recognises its solutions\(^2\):

- *Automaton* indicates that the catalogue provides a deterministic automaton,

---

\(^1\)Arc generators and graph properties are introduced in the section “Describing Explicitly Global Constraints”.

\(^2\)Automata that recognise the solutions of a global constraint were introduced in the section “Describing Explicitly Global Constraints”.

• **Automaton without counters** indicates that the catalogue provides a deterministic automaton without counters as well as without array of counters,

• **Automaton with counters** indicates that the catalogue provides a deterministic automaton with counters but without array of counters,

• **Automaton with array of counters** indicates that the catalogue provides a deterministic automaton with array of counters and possibly with counters.

In addition, we also provide a list of keywords that characterise the structure of the hypergraph associated with the decomposition of the automaton of a global constraint (i.e., see the meta-keyword *constraint network structure*). Note that, when a global constraint is defined by several graph properties it is also defined by several automata (usually one automata for each graph property). This is for instance the case of the `change_continuity` constraint. Currently we have these keywords:

• Berge-acyclic constraint network,

• Alpha-acyclic constraint network(2),

• Alpha-acyclic constraint network(3),

• Sliding cyclic(1) constraint network(1),

• Sliding cyclic(1) constraint network(2),

• Sliding cyclic(1) constraint network(3),

• Sliding cyclic(2) constraint network(2),

• Circular sliding cyclic(1) constraint network(2),

• Centered cyclic(1) constraint network(1),

• Centered cyclic(2) constraint network(1),

• Centered cyclic(3) constraint network(1),

When a global constraint is only defined by one or several automaton its signature is set to the keyword **AUTOMATON**.

**Searching from a first order logic perspective**

The keyword `logic` provides the list of constraints that are described within the catalogue in term of a first order logic formula where predicates are replaced by arithmetic constraints.

### 3.3.3 Searching all places where a global constraint is referenced

Beside the page where a global constraint is defined (in bold), the index also gives all the pages where a global constraint is referenced.

Last, since a global constraint can also be used for defining another global constraint the slot **Used in** of the description of a global constraint provides this information.
3.4. FIGURES OF THE CATALOGUE

3.3.4 Searching the mapping with a constraint of a concrete system

Two distinct ways are provided for making the correspondence between a constraint of the catalogue and a constraint of a concrete existing system:

1. Appendix C provides, when it exists, the direct correspondence between the constraints of the catalogue and the constraints of a given concrete system. For the time being we have considered, with the help on their respective authors, the following systems:
   - Choco in Java [227] (http://choco.emn.fr/),
   - Gecode in C++ [353] (http://www.gecode.org/),
   - JaCoP in Java (http://www.jacop.eu/),
   - MiniZinc (http://www.g12.cs.mu.oz.au/minizinc/),
   - SICStus [95] in Prolog (http://www.sics.se/sicstus/).

Since not all constraints of a given system always have their counterpart in the current version of the catalogue, and since systems are always enriched, this is the reason why this mapping is not complete.

2. Within the entry of the catalogue the slot Systems provides the correspondence between the constraint associated with that entry and the name of the constraint in a given concrete system or modelling language. For instance, the Systems slot of the entry of the catalogue corresponding to the element constraint indicates that element is called nth in Choco and element in Gecode, JaCoP MiniZinc and SICStus.

3.4 Figures of the catalogue

The catalogue contains the following types of figures:

- Figures that give the normalised signature tree of the arguments of a global constraint. These figures are located in Section 3.5.

- Figures that provide the implication graph between global constraints that have the same normalised signature tree for their arguments (e.g., see the figure embedded in the lower part of Table 3.1).

- Figures that illustrate a global constraint or a keyword (e.g., see Figure 3.31 that illustrates the keyword limited discrepancy search).

- Figures that depict the initial as well as the final graphs associated with a global constraint (e.g., see Figure 5.96 that provides the initial and final graphs of the change constraint).

---

3. We do not consider that a given constraint of the catalogue can be reformulated in terms of a conjunction of constraints of a given concrete system.
CHAPTER 3. DESCRIPTION OF THE CATALOGUE

- Figures that provide an automaton that only recognises the solutions associated with a given global constraint (e.g., see Figure 5.283 that gives the automaton of the global_contiguity constraint).

- Figures that give the hypergraph associated with the decomposition of an automaton in terms of signature and transition constraints (e.g., see Figure 5.284 that gives the hypergraph of the automaton-based reformulation of the global_contiguity constraint).

- Figures for the graph structure of the XML schema of the parameters of a global constraint. They are only available in the on-line version of the catalogue.

- Figures for visualising different views (i.e., compulsory part and cumulative profile) of two-dimensional placement of constraints. These figures are only available in the on-line version of the catalogue. They are accessible from the table containing the squared_squares problem instances.

Most of the graph figures that depict the initial and final graph of a global constraint of this catalogue as well as the graph structure of the XML schema of the parameters of a global constraint were automatically generated by using the open source graph drawing software Graphviz [168] available from AT&T. Within the web version, figures for visualising two-dimensional placement constraints were also automatically produced by generating PSTricks [411] code.
3.5 Constraints argument patterns

If you do not know the name of the global constraint you are looking for, but you know the types of its arguments this section allows to find out all global constraints which have similar arguments. For this purpose we associate to each global constraint of the catalogue a unique normalised signature tree derived from the types of its arguments. The purpose of this normalised signature tree is to get a concise normal form of the arguments of a global constraint that does not depend of the order in which these arguments are defined.

The normalisation takes as input the slots Type(s) and Argument(s) of the description of a global constraint and computes the normalised signature tree in four steps:

1. The first step converts all types related to variables to their corresponding ground counterpart: the types dvar, svar, mvar and rvar are respectively transformed to int, sint, mint and real.

2. The second step builds a tree of types $\mathcal{T}$ by exploring the slot Argument(s) and by developing the compound data types eventually used. The root of this tree is the type atom and represents the name of the global constraint.

3. The third step normalises the tree of types $\mathcal{T}$ by first normalising each subtree of $\mathcal{T}$ and then by sorting the children of $\mathcal{T}$. We assume the following ordering on the different types: atom $\prec$ int $\prec$ sint $\prec$ mint $\prec$ real $\prec$ list $\prec$ collection. Let $\mathcal{T}_n$ denote the normalised tree obtained at this third step.

![](image.png)

Figure 3.1: Illustrating steps (2), (3) and (4) for computing the normalised signature tree

An informal rule used in the catalogue about the order of the arguments of a constraint is that we usually first mention a domain variable which represents a result computed from one or several collections that occur just after. Finally, eventual parameters are put as the last arguments of the constraint.

See Section 2.1.4 for the description of these slots.
4. Finally the last step tries to reduce the size of the normalised tree $T_n$ by identifying $k (k > 1)$ children of a vertex $v$ of $T_n$ for which the $k$ subtrees are identical. When such a configuration is identified the $k$ subtrees of $v$ are replaced by one single subtree and the integer $k$ is put as an exponent of $v$.

<table>
<thead>
<tr>
<th>atom</th>
<th>col^2</th>
<th>int</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. alldifferent_on_intersection
2. consecutive_groups_of_ones
3. disjoint
4. incomparable
5. int_value_precede_chain
6. inverse_within_range
7. lex_different
8. lex_equal
9. lex_greater
10. lex_greatereq
11. lex_less
12. lex_leseq
13. lex_leseq_allperm
14. same
15. same_intersection
16. sort
17. used_by
18. uses
19. vec_eq_tuple

Table 3.1: Example of information associated with a normalised signature tree
The three rows of Figure 3.1 illustrate respectively the second, third and fourth steps for computing the normalised signature tree associated with the arguments of the constraints \texttt{alldifferent}, \texttt{change}, \texttt{count}, \texttt{cumulative}, \texttt{diffn}, \texttt{minimum} and \texttt{same}.

The next sections provide for each possible constraints arity all existing normalised signature trees together with the corresponding list of global constraints of the catalogue. The leftmost part of an entry corresponds to a normalised signature tree, while the rightmost upper part gives the corresponding list of global constraints. Finally the rightmost lower part describes the dependency between the constraints of the list: there is an edge from a constraint $\text{ctr}_1$ to a constraint $\text{ctr}_2$ if and only if the fact that $\text{ctr}_1$ holds implies that $\text{ctr}_2$ also holds. For instance, consider the constraints associated with the normalised signature tree corresponding to two collections of integers depicted by Table 3.1. There is an edge from 16 (i.e., \texttt{sort}) to 14 (i.e., \texttt{same}) since the fact that a \texttt{sort} constraint holds implies that a \texttt{same} constraint also holds.
### 3.5.1 Constraints with 1 argument

<table>
<thead>
<tr>
<th>atom</th>
<th>sint</th>
<th>1. <code>sum_free</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>atom</th>
<th>col</th>
<th>int</th>
<th>1. <code>alldifferent</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>2. <code>alldifferent_consecutive_values</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3. <code>alldifferent_except_0</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4. <code>all_equal</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5. <code>consecutive_values</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6. <code>decreasing</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7. <code>global_contiguity</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8. <code>golomb</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9. <code>increasing</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10. <code>multi_global_contiguity</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11. <code>no_peak</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12. <code>no_valley</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>13. <code>not_all_equal</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>14. <code>permutation</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>15. <code>some_equal</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>16. <code>strictly_decreasing</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>17. <code>strictly_increasing</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>atom</th>
<th>col</th>
<th>sint</th>
<th>1. <code>alldifferent_between_sets</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

A diagram illustrating the constraints' relationships is shown alongside the constraints list.
### 3.5. CONSTRAINTS ARGUMENT PATTERNS

<table>
<thead>
<tr>
<th>atom</th>
<th>col</th>
<th>col</th>
<th>int</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. allperm  
2. k_alldifferent  
3. k_disjoint  
4. k_same  
5. k_used_by  
6. lex2  
7. lex_alldifferent  
8. lex_chain_less  
9. lex_chain_leqseq  
10. strict_lex2

\[ 9 \quad 7 \]
\[ 6 \quad 8 \quad 5 \]
\[ 10 \quad 4 \]

<table>
<thead>
<tr>
<th>atom</th>
<th>col</th>
<th>col</th>
<th>int²</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. diffn  
2. orths_are_connected

\[ 1 \]
\[ 2 \]
## Chapter 3. Description of the Catalogue

<table>
<thead>
<tr>
<th>atom</th>
<th>col</th>
<th>int²</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. alldifferent_cst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. circuit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. derangement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. disjunctive</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. disjunctive_or_same_end</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. disjunctive_or_same_start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. precedence</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. sequence_folding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9. symmetric_alldifferent</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10. symmetric_alldifferent_except_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11. twin</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>atom</th>
<th>col</th>
<th>int</th>
<th>sint</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. atmost1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. bipartite</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. connected</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. dag</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. strongly_connected</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. symmetric</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. tour</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>atom</th>
<th>col</th>
<th>int³</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. inverse</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. orth_link_ori_siz_end</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>atom</th>
<th>col</th>
<th>int³</th>
<th>sint</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. disj</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.5. CONSTRAINTS ARGUMENT PATTERNS

\begin{center}
\begin{tabular}{|c|c|}
\hline
\begin{itemize}
\item atom
\item col
\item int\textsuperscript{2}
\item sint\textsuperscript{2}
\end{itemize} & 1. stable\textunderscore compatibility \\
\hline
\begin{itemize}
\item atom
\item col
\item col
\item int\textsuperscript{5}
\end{itemize} & 1. polyomino \\
\hline
\end{tabular}
\end{center}
3.5.2 Constraints with 2 arguments

- abs\_value
- divisible
- divisible\_or
- eq
- geq
- gt
- leq
- lt
- neq
- opposite\_sign
- same\_sign
- sign\_of

\hspace{1cm}

\hspace{1cm}

- in\_set
### 3.5. CONSTRAINTS ARGUMENT PATTERNS

<table>
<thead>
<tr>
<th>Atom</th>
<th>Argument Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td><code>all_min_dist</code></td>
</tr>
<tr>
<td>col</td>
<td><code>alldifferent_interval</code></td>
</tr>
<tr>
<td>int</td>
<td><code>alldifferent_modulo</code></td>
</tr>
<tr>
<td></td>
<td><code>among_diff_0</code></td>
</tr>
<tr>
<td></td>
<td><code>and</code></td>
</tr>
<tr>
<td></td>
<td><code>atleast_nvalue</code></td>
</tr>
<tr>
<td></td>
<td><code>atmost_nvalue</code></td>
</tr>
<tr>
<td></td>
<td><code>balance</code></td>
</tr>
<tr>
<td></td>
<td><code>between_min_max</code></td>
</tr>
<tr>
<td></td>
<td><code>deepest_valley</code></td>
</tr>
<tr>
<td></td>
<td><code>equivalent</code></td>
</tr>
<tr>
<td></td>
<td><code>highest_peak</code></td>
</tr>
<tr>
<td></td>
<td><code>in</code></td>
</tr>
<tr>
<td></td>
<td><code>increasing_nvalue</code></td>
</tr>
<tr>
<td></td>
<td><code>increasing_sum</code></td>
</tr>
<tr>
<td></td>
<td><code>inflexion</code></td>
</tr>
<tr>
<td></td>
<td><code>imply</code></td>
</tr>
<tr>
<td></td>
<td><code>length_first_sequence</code></td>
</tr>
<tr>
<td></td>
<td><code>length_last_sequence</code></td>
</tr>
<tr>
<td></td>
<td><code>max_nvalue</code></td>
</tr>
<tr>
<td></td>
<td><code>max_size_set_of_consecutive_var</code></td>
</tr>
<tr>
<td></td>
<td><code>maximum</code></td>
</tr>
<tr>
<td></td>
<td><code>min_nvalue</code></td>
</tr>
<tr>
<td></td>
<td><code>min_size_set_of_consecutive_var</code></td>
</tr>
<tr>
<td></td>
<td><code>minimum</code></td>
</tr>
<tr>
<td></td>
<td><code>nand</code></td>
</tr>
<tr>
<td></td>
<td><code>nor</code></td>
</tr>
<tr>
<td></td>
<td><code>not_in</code></td>
</tr>
<tr>
<td></td>
<td><code>nset_of_consecutive_values</code></td>
</tr>
</tbody>
</table>

*continuation ➔*
30. nvalue
31. nvisible_from_end
32. nvisible_from_start
33. or
34. peak
35. size_max_seq_alldifferent
36. size_max_starting_seq_alldifferent
37. soft_alldifferent_ctr
38. soft_alldifferent_var
39. soft_all_equal_max_var
40. soft_all_equal_min_ctr
41. soft_all_equal_min_var
42. sum_of_increments
43. valley
44. xor
3.5. CONSTRAINTS ARGUMENT PATTERNS

1. all_differ_from_at_least_k_pos
2. nvector
3. atleast_nvector
4. atmost_nvector
5. k_same_interval
6. k_same_modulo
7. k_used_by_interval
8. k_used_by_modulo
9. ordered_atleast_nvector
10. ordered_atmost_nvector
11. ordered_nvector

1. diffn_column
2. diffn_include
3. place_in_pyramid

........................................

........................................
### 1. balance_cycle
2. balance_path
3. balance_tree
4. bin_packing
5. binary_tree
6. cycle
7. domain_constraint
8. in_intervals
9. increasing_nvalue_chain
10. max_index
11. min_index
12. npair
13. open_maximum
14. open_minimum
15. ordered_global_cardinality
16. path
17. tree

```
17
16
15
14
13
12
11
10
  9
    8
      7
        6
          5
            int
```

### 1. clique
2. discrepancy
3. k_cut
4. proper_forest

```
  atom
    int
    col
      int
        int^2
```

### Additional Notes
- The table above lists various terms related to the catalogues, including `balance_cycle`, `balance_path`, `balance_tree`, `bin_packing`, `binary_tree`, `cycle`, `domain_constraint`, `in_intervals`, `increasing_nvalue_chain`, `max_index`, `min_index`, `npair`, `open_maximum`, `open_minimum`, `ordered_global_cardinality`, `path`, and `tree`.
- The diagram on the right illustrates the structure of some terms, showing how they are interconnected with nodes and edges.
3.5. CONSTRAINTS ARGUMENT PATTERNS

1. cumulative.convex

2. circuit.cluster
   2. graph.crossing
   3. orchard
   4. orth_on_the_ground
   5. track

1. cutset

1. coloured.cumulative
   2. crossing
   3. cumulative
   4. cumulative.product
   5. temporal.path

1. cumulative_two_d
### CHAPTER 3. DESCRIPTION OF THE CATALOGUE

<table>
<thead>
<tr>
<th>Atom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{atom} ) \quad \text{sint}^2</td>
<td>1. \text{eq.set}</td>
</tr>
<tr>
<td>\text{atom} \quad \text{sint} \quad \text{col} \quad \text{int}</td>
<td>1. \text{open.alldifferent}</td>
</tr>
<tr>
<td>\text{atom} \quad \text{sint} \quad \text{col} \quad \text{int}^2</td>
<td>1. \text{link.set.to.booleans}</td>
</tr>
</tbody>
</table>
3.5. CONSTRAINTS ARGUMENT PATTERNS

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>atom</td>
<td>col^2</td>
<td>int</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. `alldifferent_on_intersection`
2. `consecutive_groups_of_ones`
3. `disjoint`
4. `incomparable`
5. `int_value_precede_chain`
6. `inverse_within_range`
7. `lex_different`
8. `lex_equal`
9. `lex_greater`
10. `lex_greater_eq`
11. `lex_less`
12. `lex_leq`
13. `lex_leq_all_perm`
14. `same`
15. `same_intersection`
16. `sort`
17. `used_by`
18. `uses`
19. `vec_eq_tuple`

....................

18

17

15

12 7 10 14 1

11 9 8 16 3

19
### Description of the Catalogue

<table>
<thead>
<tr>
<th>Atom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>col</td>
</tr>
<tr>
<td>int</td>
<td>col</td>
</tr>
<tr>
<td></td>
<td>int</td>
</tr>
</tbody>
</table>

1. `alldifferent_partition`
2. `in_relation`
3. `pattern`

<table>
<thead>
<tr>
<th>Atom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>col</td>
</tr>
<tr>
<td>int</td>
<td>int</td>
</tr>
</tbody>
</table>

1. `global_cardinality`

<table>
<thead>
<tr>
<th>Atom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>col</td>
</tr>
<tr>
<td>int</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>int</td>
</tr>
</tbody>
</table>

1. `increasing_global_cardinality`
2. `global_cardinality_low_up`
3. `stretch_circuit`
4. `stretch_path`

<table>
<thead>
<tr>
<th>Atom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>col</td>
</tr>
<tr>
<td>int</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>int</td>
</tr>
</tbody>
</table>

1. `stretch_path_partition`

<table>
<thead>
<tr>
<th>Atom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col</td>
<td>col</td>
</tr>
<tr>
<td>int</td>
<td>int</td>
</tr>
</tbody>
</table>

1. `k_same_partition`
2. `k_used_by_partition`
3.5. CONSTRAINTS ARGUMENT PATTERNS

<table>
<thead>
<tr>
<th>atom</th>
<th>col</th>
<th>int</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>col</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. bin_packing_capa
2. elem
3. element_greaterq
4. element_lesseq
5. elements
6. elements_alldifferent
7. indexed_sum

<table>
<thead>
<tr>
<th>atom</th>
<th>col</th>
<th>int</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>col</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. stage_element
2. tree_resource

<table>
<thead>
<tr>
<th>atom</th>
<th>col</th>
<th>sint</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. inverse_set

<table>
<thead>
<tr>
<th>atom</th>
<th>col</th>
<th>col</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. coloured_cumulatives
2. cumulative_with_level_of_priority
3. elem_from_to
CHAPTER 3. DESCRIPTION OF THE CATALOGUE

| atom | 1. cycle_resource
| col² | 2. disjoint_tasks
| int³ | 3. two_orth_are_in_contact
|      | 4. two_orth_do_not_overlap

| atom | 1. symmetric_gcc
| col² | int² | sint

| atom | 1. symmetric_cardinality
| col² | int³ | sint
3.5.3 Constraints with 3 arguments

<table>
<thead>
<tr>
<th>Atom</th>
<th>Int^3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. distance 2. eq_cst 3. gcd 4. geq_cst 5. in_interval 6. leq_cst 7. neq_cst 8. power 9. remainder</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Atom</th>
<th>Int</th>
<th>Col</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Atom</th>
<th>Int</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Atom</th>
<th>Int</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>
### CHAPTER 3. DESCRIPTION OF THE CATALOGUE

<table>
<thead>
<tr>
<th>atom</th>
<th>atom</th>
<th>int</th>
<th>col</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>int(^2)</td>
</tr>
<tr>
<td>1. assign_and_nvalues</td>
<td>2. scalar_product</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>atom</th>
<th>atom</th>
<th>col</th>
<th>col</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>int(^2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>int(^5)</td>
</tr>
<tr>
<td>1. cumulatives</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>atom</th>
<th>atom</th>
<th>int</th>
<th>col</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>col</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>int</td>
</tr>
<tr>
<td>1. n_vectors</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>atom</th>
<th>int</th>
<th>col</th>
<th>col</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>atom</td>
<td>col</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>1. change_vectors</td>
<td>2. period_vectors</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.5. CONSTRAINTS ARGUMENT PATTERNS

1. atleast
2. atmost
3. balance_interval
4. balance_modulo
5. domain
6. element
7. exactly
8. int_value_precede
9. ith_pos_different_from_0
10. max_n
11. maximum_modulo
12. min_n
13. minimum_except_0
14. minimum_greater_than
15. minimum_modulo
16. multi_inter_distance
17. nequivalence
18. next_greater_element
19. ninterval
20. smooth

set
value
precede

1. set_value_precede
CHAPTER 3. DESCRIPTION OF THE CATALOGUE

1. \textit{in\_same\_partition}

2. \textit{interval\_and\_sum}
3. \textit{map}
4. \textit{path\_from\_to}
5. \textit{tree\_range}

6. \textit{inverse\_offset}
7. \textit{shift}
8. \textit{sliding\_time\_window}

9. \textit{cycle\_or\_accessibility}
10. \textit{sliding\_time\_window\_sum}
### 3.5. CONSTRAINTS ARGUMENT PATTERNS

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>alldifferent_same_value</code></td>
<td>1.</td>
</tr>
<tr>
<td><code>among</code></td>
<td>2.</td>
</tr>
<tr>
<td><code>among_var</code></td>
<td>3.</td>
</tr>
<tr>
<td><code>cardinality_atleast</code></td>
<td>4.</td>
</tr>
<tr>
<td><code>cardinality_atmost</code></td>
<td>5.</td>
</tr>
<tr>
<td><code>clause_and</code></td>
<td>6.</td>
</tr>
<tr>
<td><code>clause_or</code></td>
<td>7.</td>
</tr>
<tr>
<td><code>differ_from_at_least_k_pos</code></td>
<td>8.</td>
</tr>
<tr>
<td><code>elementn</code></td>
<td>9.</td>
</tr>
<tr>
<td><code>nvalue_on_intersection</code></td>
<td>10.</td>
</tr>
<tr>
<td><code>same_interval</code></td>
<td>11.</td>
</tr>
<tr>
<td><code>same_modulo</code></td>
<td>12.</td>
</tr>
<tr>
<td><code>soft_same_var</code></td>
<td>13.</td>
</tr>
<tr>
<td><code>soft_used_by_var</code></td>
<td>14.</td>
</tr>
<tr>
<td><code>used_by_interval</code></td>
<td>15.</td>
</tr>
<tr>
<td><code>used_by_modulo</code></td>
<td>16.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>balance_partition</code></td>
<td>1.</td>
</tr>
<tr>
<td><code>cardinality_atmost_partition</code></td>
<td>2.</td>
</tr>
<tr>
<td><code>change_partition</code></td>
<td>3.</td>
</tr>
<tr>
<td><code>cond_lex_cost</code></td>
<td>4.</td>
</tr>
<tr>
<td><code>nclass</code></td>
<td>5.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>global_cardinality_no_loop</code></td>
<td>1.</td>
</tr>
<tr>
<td><code>sum_of_weights_of_distinct_values</code></td>
<td>2.</td>
</tr>
</tbody>
</table>
CHAPTER 3. DESCRIPTION OF THE CATALOGUE

1. minimum_weight_allDifferent
2. sliding_distribution

1. element_sparse
2. elements_sparse

1. orth_on_top_of_orth
2. two_orth_column
3. two_orth_include

1. geost

1. roots
3.5. CONSTRAINTS ARGUMENT PATTERNS

<table>
<thead>
<tr>
<th></th>
<th>1. open_global_cardinality</th>
</tr>
</thead>
</table>
| \[
\begin{array}{c}
\text{atom} \\
\text{sint} \quad \text{col} \quad \text{col} \\
\quad \text{int} \quad \text{int}^2 \\
\end{array}
\] |

<table>
<thead>
<tr>
<th></th>
<th>1. open_global_cardinality_low_up</th>
</tr>
</thead>
</table>
| \[
\begin{array}{c}
\text{atom} \\
\text{sint} \quad \text{col} \quad \text{col} \\
\quad \text{int} \quad \text{int}^3 \\
\end{array}
\] |

|                           | 1. correspondence  
|---------------------------|------------------|
| \[
\begin{array}{c}
\text{atom} \\
\text{col}^3 \\
\quad \text{int} \\
\end{array}
\] |
|                           | 2. lex_between  
|                           | 3. sort_permutation  

\[
\begin{array}{c}
\text{...} \\
\quad 1 \\
\quad 3 \\
\end{array}
\] 

|                           | 1. subgraph_isomorphism  
|---------------------------|------------------|
| \[
\begin{array}{c}
\text{atom} \\
\text{col} \quad \text{col}^2 \\
\quad \text{int} \quad \text{int} \quad \text{sint} \\
\end{array}
\] |
|                           | 2. graph_isomorphism  

|                           | 1. cond_lex_greater  
|---------------------------|------------------|
| \[
\begin{array}{c}
\text{atom} \\
\text{col}^2 \quad \text{col} \\
\quad \text{int} \quad \text{col} \quad \text{int} \\
\end{array}
\] |
|                           | 2. cond_lex_greatereq  
|                           | 3. cond_lex_less  
|                           | 4. cond_lex_leq  
|                           | 5. same_partition  
|                           | 6. used_by_partition  

\[
\begin{array}{c}
\quad 1 \\
\quad 3 \\
\quad 5 \\
\end{array}
\]
CHAPTER 3. DESCRIPTION OF THE CATALOGUE

1. same_and_global_cardinality

1. same_and_global_cardinality_low_up
### 3.5.4 Constraints with 4 arguments

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>atom atom int col int^2</code></td>
<td>1. <code>change_pair</code></td>
</tr>
</tbody>
</table>
| `atom atom int^2 col int` | 1. `count`  
2. `cyclic_change`  
3. `cyclic_change_joker` |
| `atom atom int sint col int` | 1. `sum_set` |
| `atom atom int col^2 int` | 1. `arith_or`  
2. `counts`  
3. `distance_between`  
4. `distance_change` |
| `atom atom int col col int int^2` | 1. `assign_and_counts` |
| `atom int int^4` | 1. `in_interval_reified` |
1. among\_interval
2. among\_modulo
3. element\_product
4. sliding\_sum

1. next\_element

1. sliding\_time\_window\_from\_start

1. soft\_cumulative

1. open\_atleast
2. open\_atmost
### 3.5. CONSTRAINTS ARGUMENT PATTERNS

<table>
<thead>
<tr>
<th>Diagram</th>
<th>Constraints</th>
</tr>
</thead>
</table>
| ![Diagram 1](image1) | 1. `among_low_up`  
2. `common`  
3. `sliding_card_skip0`  
4. `soft_same_interval_var`  
5. `soft_same_modulo_var`  
6. `soft_used_by_interval_var`  
7. `soft_used_by_modulo_var` |
| ![Diagram 2](image2) | 1. `interval_and_count`  
2. `weighted_partial_alldiff` |
| ![Diagram 3](image3) | 1. `sum` |
| ![Diagram 4](image4) | 1. `global_cardinality_low_up_no_loop` |
| ![Diagram 5](image5) | 1. `open_among` |
CHAPTER 3. DESCRIPTION OF THE CATALOGUE

1. geost_time

1. dom.reachability

1. soft_same_partition_var
2. soft_used_by_partition_var

1. global_cardinality_with_costs

1. two_layer_edge_crossing
3.5.5 Constraints with 5 arguments

1. visible

1. connect.points

1. among_seq
   2. common_interval
   3. common_modulo

1. common_partition
### 3.5.6 Constraints with 6 arguments

<table>
<thead>
<tr>
<th>Constraint Type</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atom</td>
<td>int&lt;sup&gt;5&lt;/sup&gt;, collist, int&lt;sup&gt;6&lt;/sup&gt;</td>
</tr>
<tr>
<td>Atom</td>
<td>int&lt;sup&gt;5&lt;/sup&gt;, collist, int&lt;sup&gt;3&lt;/sup&gt;</td>
</tr>
<tr>
<td>Atom</td>
<td>int&lt;sup&gt;4&lt;/sup&gt;, collist, int&lt;sup&gt;2&lt;/sup&gt;</td>
</tr>
<tr>
<td>Atom</td>
<td>int&lt;sup&gt;4&lt;/sup&gt;, collist, int&lt;sup&gt;3&lt;/sup&gt;</td>
</tr>
<tr>
<td>Atom</td>
<td>int&lt;sup&gt;3&lt;/sup&gt;, collist, int&lt;sup&gt;3&lt;/sup&gt;</td>
</tr>
</tbody>
</table>
3.5. CONSTRAINTS ARGUMENT PATTERNS

3.5.7 Constraints with 8 arguments

3.5.8 Constraints with 10 arguments
3.6 Meta-keywords attached to the keywords

This section explains the meaning of the meta-keywords attached to the keywords of the catalogue. Keywords are usually associated with one single meta-keyword, except some that are linked to the meta-keyword modelling exercises and to one other meta-keyword like modelling or puzzles (e.g., see for instance the keywords magic series or degree of diversity of a set of solutions). For each meta-keyword it first gives the list of keywords using the corresponding meta-keyword and then defines the meta-keyword. At present the following meta-keywords are in use.

3.6.1 Application area

- Air traffic management,
- Assignment,
- Bioinformatics,
- Configuration problem,
- Deadlock breaking,
- Floor planning problem
- Frequency allocation problem,
- Phylogeny,
- Program verification,
- SLAM problem,
- Sport timetabling,
- Workload covering.

Denotes that a keyword is related to an application area.

3.6.2 Characteristic of a constraint

- All different,
- Automaton,
- Automaton with array of counters,
- Automaton with counters,
- Automaton without counters,
- Coloured,
- Consecutive values,
- Convex,
- Convex hull relaxation,
- Core,
- Cyclic,
- Derived collection,
- Difference,
- Disequality,
- Equality,
- Hypergraph,
- Joker value,
- Maximum,
- maxint,
- Minimum,
- Modulo,
- Non-deterministic automaton,
- Pair,
- Partition,
- Product,
- Range,
- Rank,
- Reified automaton constraint,
- Reified constraint,
- Sort,
- Sort based reformulation,
- Sum,
- Time window,
- Tuple,
- Undirected graph,
- Vector.
3.6. **META-KEYWORDS ATTACHED TO THE KEYWORDS**

Denotes that a keyword is related to a characteristic of the description of a constraint.

### 3.6.3 Combinatorial object

- Latin square,
- Matching,
- Multiset,
- Path,
- Pentomino,
- Periodic,
- Permutation,
- Relation,
- Run of a permutation,
- Sequence.

Denotes that a keyword corresponds to a combinatorial object or to a characteristic of a combinatorial object.

### 3.6.4 Complexity

- 3-dimensional-matching,
- 3-SAT,
- Minimum hitting set cardinality,
- Rectangle clique partition,
- Sequencing with release times and deadlines,
- Set packing,
- Subset sum.

Denotes that a keyword corresponds to a problem used to recognise NP-hard problems attached to the feasibility of a constraint.

### 3.6.5 Constraint network structure

- Alpha-acyclic constraint network(2),
- Alpha-acyclic constraint network(3),
- Berge-acyclic constraint network,
- Centered cyclic(1) constraint network(1),
- Centered cyclic(2) constraint network(1),
- Centered cyclic(3) constraint network(1),
- Circular sliding cyclic(1) constraint network(2),
- Sliding cyclic(1) constraint network(1),
- Sliding cyclic(1) constraint network(2),
- Sliding cyclic(1) constraint network(3),
- Sliding cyclic(2) constraint network(2),
- Sliding cyclic(2) constraint network(2),

Denotes that a keyword designates a specific constraint network structure occurring repeatedly in several constraints.
3.6.6 Constraint type

- Arithmetic constraint,
- Boolean constraint,
- Conditional constraint,
- Constraint on the intersection,
- Counting constraint,
- Data constraint,
- Decomposition,
- Decomposition-based violation measure,
- Extension,
- Graph constraint,
- Graph partitioning constraint,
- Logic,
- Open automaton constraint,
- Open constraint,
- Order constraint,
- Overlapping alldifferent,
- Predefined constraint,
- Proximity constraint,
- Relaxation,
- Resource constraint,
- Scheduling constraint,
- Sliding sequence constraint,
- Soft constraint,
- System of constraints,
- Temporal constraint,
- Timetabling constraint,
- Value constraint,
- Value partitioning constraint,
- Variable-based violation measure.

Denotes that a keyword designates a constraint category.

3.6.7 Constraint arguments

- Aggregate,
- Binary constraint,
- Business rules,
- Constraint between three collections of variables,
- Constraint between two collections of variables,
- Constraint involving set variables,
- Contractible,
- Extensible,
- Pure functional dependency,
- Ternary constraint,
- Unary constraint.

Denotes that a keyword provides an information about the arguments of a constraint.

3.6.8 Filtering

- Abstract interpretation,
- Arc-consistency,
- Bipartite matching,
- Bipartite matching in convex bipartite graphs,
- Border,
- Bound-consistency,
3.6. META-KEYWORDS ATTACHED TO THE KEYWORDS

- Compulsory part,
- Constructive disjunction,
- Convex bipartite graph,
- Cost filtering constraint,
- Cumulative longest hole problems,
- DFS-bottleneck,
- Duplicated variables,
- Dynamic programming,
- Entailment,
- Flow,
- Hall interval,
- Hungarian method for the assignment problem,
- Hybrid-consistency,
- Klee measure problem,
- Linear programming,
- Minimum cost flow,
- Phi-tree,
- Planarity test,
- Quadtree,
- SAT,
- Strong articulation point,
- Strong bridge,
- Sweep.

Denotes that a keyword is related to an existing or a potential filtering algorithm of a constraint or to an algorithm checking a ground instance of a constraint.

3.6.9 Final graph structure

- Acyclic,
- Apartition,
- Bipartite,
- Circuit,
- Connected component,
- Consecutive loops are connected,
- Directed acyclic graph,
- Equivalence,
- No cycle,
- No loop,
- One succ,
- Strongly connected component,
- Symmetric,
- Tree,
- Vpartition.

Denotes that a keyword describes the structure of the final graph associated with a constraint.

3.6.10 Geometry

- Alignment,
- Contact,
- Geometrical constraint,
- Guillotine cut,
- Line-segments intersection,
- Non-overlapping,
- Orthotope,
- Polygon,
- Positioning constraint,
- RCC8,
- Touch.

Denotes that a keyword is related to a geometrical constraint or to a geometrical object.
3.6.11 Heuristics

- Heuristics,
- Heuristics and Berge-acyclic constraint network,
- Heuristics and lexicographical ordering,
- Heuristics for two-dimensional rectangle placement problems,
- Labelling by increasing cost,
- Limited discrepancy search,
- Regret based heuristics,
- Regret based heuristics in matrix problems.

Denotes that a keyword is related to a search heuristics.

3.6.12 Miscellaneous

- Obscure.

Denotes that a keyword does not belong to any class.

3.6.13 Modelling

- Array constraint,
- Assigning and scheduling tasks that run in parallel,
- Assignment dimension,
- Assignment to the same set of values,
- At least,
- At most,
- Balanced assignment,
- Balanced tree,
- Boolean channel,
- Channelling constraint,
- Cluster,
- Cost matrix,
- Cycle,
- Degree of diversity of a set of solutions,
- Difference between pairs of variables,
- Disjunction,
3.6. META-KEYWORDS ATTACHED TO THE KEYWORDS

- Domain channel,
- Domain definition,
- Dual model,
- Empty intersection,
- Equality between multisets,
- Excluded,
- Functional dependency,
- Included,
- Inclusion,
- Incompatible pairs of values,
- Interval,
- Matrix,
- Matrix model,
- Maximum number of occurrences,
- Minimum number of occurrences,
- Multi-site employee scheduling with calendar constraints,
- Number of changes,
- Number of distinct equivalence classes,
- Number of distinct values,
- Permutation channel,
- Preferences,
- Relaxation dimension,
- Scalar product,
- Scheduling with machine choice, calendars and preemption,
- Sequence dependent set-up,
- Set channel,
- Shared table,
- Sparse functional dependency,
- Sparse table,
- Statistics,
- Table,
- Variable indexing,
- Variable subscript,
- Zero-duration task.

Denotes that a keyword is related to a modelling issue.
3.6.14 Modelling exercises

- Assigning and scheduling tasks that run in parallel: inspired by a modelling question on the Choco mailing list about an assignment and scheduling problem involving nurses and surgeons, use one geost constraint as well as inequalities for breaking symmetries with respect to groups of identical persons. The keyword relaxation dimension shows how to extend the previous model in order to take into account over-constrained assignment and scheduling problems.

- Assignment to the same set of values: inspired by a presentation of F. Hermenier about a task assignment problem where subtasks have to be assigned a same group of machines, use several element constraints and one single resource constraint that has an assignment dimension (e.g., bin_packing, cumulatives, diffn, geost).

- Degree of diversity of a set of solutions: inspired by a discussion with E. Hebrard, how to find out 9 completely different solutions for the 10-queens problem, use the alldifferent, the soft_alldifferent_ctr and the lex_chain_less constraints.

- Logigraph: inspired by an instance from [297, page 36], use a conjunction of consecutive_groups_of_ones constraints.

- Magic series: a special case of Autoref, use one single global_cardinality constraint.

- Metro: a model from H. Simonis, use only leq_cst constraints and propagation (i.e., no enumeration) for modelling the shortest path problem in a network.

- Multi-site employee scheduling with calendar constraints: a timetabling problem, inspired by H. Simonis, where tasks have to be assigned groups of employees located in different countries subject to different calendars, use resource constraints as well as the calendar constraint.

- n-Amazon: an extension of the n-queen problem, use one alldifferent constraint, two alldifferent_cst constraints and three smooth constraints.

- Relaxation dimension: illustrate how to model over-constrained placement problems by introducing an extra dimension in the context of the diffn and the geost constraints.

- Scheduling with machine choice, calendars and preemption: a scheduling problem with crossable and non-crossable unavailability periods as well as resumable and non-resumable tasks, illustrate the use of two time coordinates systems within the same model, use precedence and resource constraints as well as the calendar constraint.

- Sequence dependent set-up: a classical scheduling problem, use the sum_ctr, element and temporal_path constraints.

- Zebra puzzle: illustrate the duality of choice of what is a variable and what is a value in a constraint model as well as the difficulty of stating the constraints in one of the two models, use the alldifferent, the element – with variables in the table – and the inverse constraints.

Denotes that a keyword describes a constraint modelling exercise.
3.6. META-KEYWORDS ATTACHED TO THE KEYWORDS

3.6.15 Problems

- Channel routing,
- Demand profile,
- Domination,
- Facilities location problem,
- Graph colouring,
- Hamiltonian,
- Maximum clique,
- Minimum feedback vertex set,
- Pallet loading,
- Pattern sequencing,
- Pick-up delivery,
- Producer-consumer,
- Schur number,
- Strip packing,
- Two-dimensional orthogonal packing,
- Weighted assignment.

Denotes that a keyword is related to a problem from Operations Research.

3.6.16 Puzzles

- Autoref,
- Conway packing problem,
- Costas arrays,
- Dominating queens,
- Euler knight,
- Golomb ruler,
- Logigraphe,
- Magic hexagon,
- Magic series,
- Magic square,
- n-Amazon,
- n-queen,
- Packing almost squares,
- Partridge,
- Pentomino,
- Shikaku,
- Smallest rectangle area,
- Smallest square for packing consecutive dominoes,
- Smallest square for packing rectangles with distinct sizes,
- Squared squares,
- Sudoku,
- Zebra puzzle.

Denotes that a keyword is related to a specific puzzle.
3.6.17 Symmetry

- Indistinguishable values,
- Lexicographic order,
- Matrix symmetry,
- Multiset ordering,
- Symmetry,
- Value precedence.

Denotes that a keyword is related to a symmetry breaking technique [113, 156].
3.7 Keywords attached to the global constraints

This section explains the meaning of the keywords attached to the global constraints of the catalogue. For each keyword it first gives the list of global constraints using the corresponding keyword and then defines the keyword. At present the following keywords are in use.

3.7.1 ▼ 3-dimensional-matching ➔ [2 CONS]

- k_same,
- soft_all_equal_min_ctr.

Denotes that, by reduction to 3-dimensional-matching, deciding whether a constraint has a solution or not was shown to be NP-hard. The 3-dimensional-matching problem can be described as follows: given a set \( S \subseteq X \times Y \times Z \), where \( X, Y \) and \( Z \) are disjoint sets having the same number of elements \( m \), does \( S \) contain a subset \( M \) of \( m \) elements such that no two elements of \( M \) agree in any coordinate?

3.7.2 ▼ 3-SAT ➔ [5 CONS]

- atmost_nvalue,
- common,
- global_cardinality,
- nvalue,
- uses.

Denotes that, by reduction to 3-SAT, deciding whether a constraint has a solution or not was shown to be NP-hard. The 3-SAT problem can be described as follows: given a collection \( C \) of clauses involving a set of variables \( V \), where each clause has exactly 3 variables, is there a truth assignment for \( V \) that satisfies all the clauses of \( C \)?

3.7.3 ▼ Abstract interpretation ➔ [2 CONS]

- gcd,
- power.

Denotes that abstract interpretation was used for deriving a filtering algorithm for a constraint \( C \) from a polynomial algorithm describing a checker for a ground instance of \( C \). Abstract interpretation [120] executes an algorithm on abstract values in order to deduce some information about that algorithm.
3.7.4 ▶ Acyclic

- alldifferent_on_intersection,
- allperm,
- among_low_up,
- among_var,
- arith_or,
- assign_and_counts,
- assign_and_values,
- bin_packing,
- cardinality_atleast,
- cardinality_atmost,
- cardinality_atmost_partition,
- change,
- change_continuity,
- change_pair,
- change_partition,
- common,
- common_interval,
- common_modulo,
- common_partition,
- correspondence,
- counts,
- crossing,
- cutset,
- cyclic_change,
- decreasing,
- lex_equal,
- uses.

Denotes that a constraint is defined by one single graph constraint for which the final graph doesn’t have any circuit.

3.7.5 ▶ Aggregate

- among (+, union, union),
- among_diff0 (+, union),
- among_interval (+, union, id, id),
- among_low_up (+, +, union, union),
- among_modulo (+, union, id, id),
- among_var (+, union, union),
- and (∧, union),
- count (id, union, id, +) when RELOP ∈ [{<, ≤, ≥, >}],
- counts (unions, union, id, +) when RELOP ∈ [{<, ≤, ≥, >}],
- discrepancy (union, +),
- exactly (+, union, id),
- int_value_precede (id, id, union),
- int_value_precede_chain (id, union),
- maximum (max, union),
- minimum (min, union),
- minimum_greater_than (min, id, union),
- nand (∨, union),
- nor (∧, union),
- or (∨, union),
- product_ctr (union, id, *) when CTR ∈ [=],
- same (union, union),
- same_interval (union, union, id),
- same_modulo (union, union, id),
- same_partition (union, union, id),
- scalar_product (union, id, +),
- sum_ctr (union, id, +),
- sum_cubes_ctr (union, id, +),
- sum_squares_ctr (union, id, +),
- used_by (union, union),
- used_by_interval (union, union, id),
- used_by_modulo (union, union, id),
- used_by_partition (union, union, id),
- uses (union, union).
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

Denotes that, given two instances of a constraint, we can combine (i.e., aggregate) these two instances in order to obtain a third constraint, which has the same name as the first two constraints. The first two constraints are called the *source* constraints, while the implied constraint is called the *target* constraint. The *ith* argument of the target constraint is obtained by combining the *ith* arguments of the two source constraints. This is specified for each argument by one of the following options.

- **id**: check that the corresponding arguments of the two source constraints are *identical* and take it as the argument of the target constraint; this option if often used for specifying that an argument corresponding to a parameter has to be the same in the two source constraints, as well as in the target constraint (i.e., the source and the target constraints share the same parameter).

- **+**: *add* the corresponding arguments of the two source constraints.

- **∗**: *multiply* the corresponding arguments of the two source constraints.

- **∧**: make an *and* between the corresponding 0-1 arguments of the two source constraints.

- **∨**: make an *or* between the corresponding 0-1 arguments of the two source constraints.

- **min**: take the *minimum* of the corresponding arguments of the two source constraints.

- **max**: take the *maximum* of the corresponding arguments of the two source constraints.

- **union**: take the *union*, without removing duplicates, of the collections items of the corresponding arguments of the two source constraints.

- **sunion**: take the *union*, and remove duplicates, of the collections items of the corresponding arguments of the two source constraints, where collections correspond to collection of ground values (i.e., parameters).

Finally, the aggregation may be conditioned by a list of restrictions, each restriction corresponding to one of the restrictions described in Section 2.1.3. We call this *conditional aggregation*.

Most constraints for which aggregation applies correspond to constraints where one of the arguments is *functionally determined* by the other arguments. This is for instance the case for the *maximum*(MAX, VARIABLES) constraint which enforces MAX to be equal to the maximum value assigned to the variables of VARIABLES. However some constraints, like the *same* constraint, for which aggregation applies, do not have any argument that is functionally determined by the other arguments.

We now present three examples of deductions that can be obtained by aggregating two source constraints.

- **among**(1, \((4, 5, 5, 4, 1), (0, 1)\)) ∧ **among**(3, \((1, 1, 9, 0), (0, 1)\)) ⇒ **among**(4, \((4, 5, 5, 4, 1, 1, 1, 9, 0), (0, 1)\)), where:
1. The first argument of the target constraint, i.e., 4, is equal to the sum of the first arguments of the two source constraints, i.e., 1 + 3.

2. The second argument of the target constraint, \(4, 5, 5, 4, 1, 1, 1, 9, 0\), is equal to the union (without removing duplicates) of the second arguments \((4, 5, 5, 4, 1)\) and \((1, 1, 9, 0)\) of the two source constraints.

3. The third arguments of the two source constraints are identical, i.e., \((0, 1)\), and the third argument of the target constraint.

\[\text{maximum}(5, (3, 0, 5, 2, 5)) \land \text{maximum}(9, (1, 1, 9, 0)) \Rightarrow \text{maximum}(9, (3, 0, 5, 2, 5, 1, 1, 9, 0))\]

where:

1. The first argument of the target constraint, i.e., 9, is equal to the maximum value of the first arguments of the two source constraints, i.e., \(\text{max}(5, 9)\).

2. The second argument of the target constraint, \((3, 0, 5, 2, 5, 1, 1, 9, 0)\), is equal to the union (without removing duplicates) of the second arguments \((3, 0, 5, 2, 5)\) and \((1, 1, 9, 0)\) of the two source constraints.

\[\text{same}((3, 3, 1), (3, 1, 3)) \land \text{same}((1, 9, 1, 5, 5), (5, 5, 1, 1, 9)) \Rightarrow \text{same}((3, 3, 1, 1, 9, 1, 5, 5), (3, 1, 3, 5, 5, 1, 1, 9))\]

where:

1. The first argument of the target constraint, \((3, 3, 1, 1, 9, 1, 5, 5)\), is equal to the union (without removing duplicates) of the first arguments \((3, 3, 1)\) and \((1, 9, 1, 5, 5)\) of the two source constraints.

2. The second argument of the target constraint, \((3, 1, 3, 5, 5, 1, 1, 9)\), is equal to the union (without removing duplicates) of the second arguments \((3, 1, 3)\) and \((5, 5, 1, 1, 9)\) of the two source constraints.

### 3.7.6 Air traffic management ➤ [3 CONS]

- all_min_dist
- k_alldifferent
- multi_inter_distance

Denotes that a constraint was used for solving a problem in the area of air traffic management.

### 3.7.7 Alignment ➤ [1 CONS]

- orchard

Denotes that a constraint enforces the alignment of different sets of points.
3.7.8 **All different**

- alldifferent,
- alldifferent_between_sets,
- alldifferent_cst,
- alldifferent_consecutive_values,
- alldifferent_except_0,
- alldifferent_interval,
- alldifferent_modulo,
- alldifferent_on_intersection,
- alldifferent_partition,
- golomb,
- k_alldifferent,
- open_alldifferent,
- permutation,
- size_max_starting_seq_alldifferent,
- size_max_seq_alldifferent,
- soft_alldifferent_ctr,
- soft_alldifferent_var,
- symmetric_alldifferent,
- weighted_partial_alldiff.

Denotes that we have one or several cliques of disequalities or that a constraint is a variation of the alldifferent constraint. Variations may be related to relaxation (see, e.g., the alldifferent_except_0, soft_alldifferent_ctr, and soft_alldifferent_var constraints), or to specialisation (see, e.g., the symmetric_alldifferent constraint), of the alldifferent constraint. Variations may also result from an extension of the notion of disequality (see, e.g., the alldifferent_interval, alldifferent_modulo, alldifferent_partition and golomb constraints).

3.7.9 **Alpha-acyclic constraint network(2)**

- among,
- among_diff_0,
- among_interval,
- among_low_up,
- among_modulo,
- atleast,
- atmost,
- count,
- counts,
- differ_from_at_least_k_pos,
- exactly,
- group,
- group_skip_isolated_item,
- sliding_card_skip0.
Before defining *alpha-acyclic constraint network*(2) we first need to introduce the following notions:

- The *dual graph* of a constraint network \( \mathcal{N} \) is defined in the following way: to each constraint of \( \mathcal{N} \) corresponds a vertex in the dual graph and if two constraints have a non-empty set \( S \) of shared variables, there is an edge labelled \( S \) between their corresponding vertices in the dual graph.

- An edge in the dual graph of a constraint network is *redundant* if its variables are shared by every edge along an alternative path between the two end points [124].

- If the subgraph resulting from the removal of the redundant edges of the dual graph is a tree the original constraint network is called \( \alpha \)-acyclic [145].

\textit{Alpha-acyclic constraint network}(2) denotes an \( \alpha \)-acyclic constraint network such that, for any pair of constraints, the two sets of involved variables share at most two variables.

**3.7.10 ▼ Alpha-acyclic constraint network(3) ➤**

- \textit{group,}
- \textit{group.skip.isolated.item,}
- \textit{ith.pos.different.from.0.}

\textit{Alpha-acyclic constraint network(3)} denotes an \( \alpha \)-acyclic constraint network (see *alpha-acyclic constraint network*(2)) such that, for any pair of constraints, the two sets of involved variables share at most three variables.

**3.7.11 ▼ Apartition ➤**

- \textit{change.continuity.}

Denotes that a constraint is defined by two graph constraints having the same initial graph, where each arc of the initial graph belongs to one of the final graph (but not to both).
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.12 ▶Arc-consistency ◀

- abs_value,
- alldifferent,
- alldifferent_cst,
- alldifferent_except_0,
- alldifferent_interval,
- alldifferent_modulo,
- alldifferent_partition,
- among,
- among_diff_0,
- among_interval,
- among_low_up,
- among_modulo,
- among_seq,
- and,
- arith,
- arith_or,
- atleast,
- atleast_nvalue,
- atmost,
- cardinality_atleast,
- cardinality_atmost,
- cardinality_atmost_partition,
- clause_and,
- clause_or,
- cond_lex_cost,
- cond_lex_greater,
- cond_lex_greatereq,
- cond_lex_less,
- cond_lex_leseq,
- consecutive_groups_of_ones,
- count,
- counts,
- decreasing,
- derangement,
- discrepancy,
- divisible,
- domain_constraint,
- elem,
- elem_from_to,
- element,
- elementn,
- element_greatereq,
- element_leseq,
- element_matrix,
- element_sparse,
- elements,
- elements_sparse,
- eq,
- eq_cst
- equivalent,
- exactly,
- geq,
- geq_cst,
- global_cardinality_low_up,
- global_contiguity,
- gt,
- imply,
- in,
- in_interval,
- in_interval_reified,
- in_intervals,
- in_relation,
- in_same_partition,
- increasing,
- increasing_global_cardinality,
- increasing_nvalue,
- int_value_precede,
- int_value_precede_chain,
- inverse,
- inverse_offset,
- leq,
- leq_cst,
- lex_alldifferent,
- lex_between,
- lex_chain_less,
- lex_chain_leseq.
Denotes that, for a given constraint involving only domain variables, there is a filtering algorithm that ensures arc-consistency. A constraint \(\text{ctr}\) defined on the distinct domain variables \(V_1, \ldots, V_n\) is \textit{arc-consistent} if and only if for every pair \((V, v)\) such that \(V\) is a domain variable of \(\text{ctr}\) and \(v \in \text{dom}(V)\), there exists at least one solution to \(\text{ctr}\) in which \(V\) is assigned the value \(v\). As quoted by C. Bessière in [55], “a different name has often been used for arc-consistency on non-binary constraints”, like domain consistency, generalized arc-consistency or hyper arc-consistency.

There is also a weaker form of arc-consistency that also try to remove values from the middle of the domain of a variable \(V\) (i.e., unlike bound-consistency which focus on reducing the minimum and maximum value of a variable), called range consistency in [55], that is defined in the following way. A constraint \(\text{ctr}\) defined on the distinct domain variables \(V_1, \ldots, V_n\) is \textit{range-consistent} if and only if, for every pair \((V, v)\) such that \(V\) is a domain variable of \(\text{ctr}\) and \(v \in \text{dom}(V)\), there exists at least a solution to \(\text{ctr}\) in which, (1) \(V\) is assigned the value \(v\), and (2) each variable \(U \in \{V_1, \ldots, V_n\}\) distinct from \(V\) is assigned a value located in its range \([U, U]\).
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.13 Arithmetic constraint

- abs_value,
- arith_sliding,
- distance,
- divisible,
- divisible_or,
- eq,
- eq_cst,
- gcd,
- geq,
- geq_cst,
- gt,
- increasing_sum,
- leq,
- leq_cst,
- lt,
- neq,
- neq_cst,
- opposite_sign,
- power,
- product_ctr,
- range_ctr,
- remainder,
- same_sign,
- sign_of,
- scalar_product,
- sum_ctr,
- sum_set,
- sum_cubes_ctr,
- sum_squares_ctr.

An arithmetic constraint between two or three variables or an arithmetic constraint involving a sum, a product, or a difference between a maximum and a minimum value. The non binary constraints were introduced within the catalogue since they are required for defining a given global constraint. For instance the sum_ctr constraint is used within the definition of the cumulative constraint.

3.7.14 Array constraint

- elem,
- elem_from_to,
- element,
- elements_alldifferent,
- element_leqseq,
- element_greatereq,
- element_matrix,
- element_product,
- element_sparse.

A constraint that allows for expressing simple array equations.
3.7.15 Assigning and scheduling tasks that run in parallel

Given a set of tasks defined by a set of subtasks, where each subtask has the following attributes:

- A *start* telling when the subtask starts.
- A *duration* giving the duration of the subtask.
- A *deadline* requesting the subtask to finish no later than a given date.
- A *person* indicating which person performs the subtask.

Both the start and the person correspond to discrete decision variables, while the duration and the deadline are integers. Since all subtasks of a same task must run in parallel, their start, duration and deadline are identical. Since a person can perform at most one task at each timepoint, persons assigned to the subtasks of a same task must all be distinct. We also assume that a subtask cannot be preempted.

As an instance of this pattern, consider the problem of scheduling surgical operations in an hospital. Each surgery corresponds to a task that requires a number of persons with specific skills; these persons will all work together during the operation (e.g., typically an anaesthetist, a surgeon and one or several nurses). Moreover, each person has its own calendar defining its unavailability. On the one hand, let us assume we have two anaesthetists, two surgeons and four nurses that are labelled from 1 to 8. Each of them has the following unavailability over the time horizon $[0, 24]$:

- The first anaesthetist is not available during the time periods $[0, 1]$, $[5, 6]$, and $[12, 16]$.
- The second anaesthetist is not available during the time periods $[0, 2]$, $[6, 6]$, $[15, 15]$, and $[22, 22]$.
- The first surgeon is not available during the time periods $[0, 1]$, $[8, 9]$, and $[13, 14]$.
- The second surgeon is not available during the time periods $[5, 5]$, and $[20, 21]$.
- The four nurses are all not available during the time periods $[0, 0]$, $[7, 7]$, $[12, 12]$, and $[22, 22]$.

On the other hand, let us suppose we have to schedule five operation tasks, each of them requiring a specific team:
3.7. **KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS**

- Task $t_1$ needs one anaesthetist, one surgeon and two nurses during two consecutive time slots.
- Task $t_2$ needs one anaesthetist, one surgeon and one nurse during four consecutive time slots.
- Task $t_3$ needs one anaesthetist, two surgeons and two nurses during three consecutive time slots.
- Task $t_4$ needs one anaesthetist, one surgeon and three nurses during two consecutive time slots.
- Task $t_5$ needs one anaesthetist, one surgeon and one nurse during six consecutive time slots.

Moreover, tasks $t_1$, $t_2$, $t_3$, $t_4$ and $t_5$ must be respectively completed no later than 12, 15, 24, 24 and 24. The problem is modelled by using a two-dimensional `geost` constraint, where the first and second dimensions respectively correspond to the time and resource axes. For each person required by a task we create a rectangle of length corresponding to the necessary duration and of height 1 (i.e., 1 since it requires one person). The coordinates of the leftmost lower point of the rectangle correspond to the start of the corresponding task as well as to the person that will be assigned to the subtask (i.e., a value between 1 and 2 for an anaesthetist, a value between 3 and 4 for a surgeon, and a value between 5 and 8 for a nurse). Both the start and the person correspond to a domain variable. Each unavailability period of an anaesthetist, a surgeon and a nurse is modelled by introducing a fixed rectangle (i.e., its coordinates are set to the start of the unavailability period and to the person to which the unavailability belongs; its duration is set to the duration of the unavailability period) that prevent tasks overlapping the corresponding time period for a specific person. This leads to the following `geost` constraint,
A deadline constraint for an operation starting at $t$ and of duration $d$ is modelled by a precedence constraint of the form $o + d \leq \text{deadline}$. This leads to the five constraints $o_1 + 2 \leq 12, o_2 + 4 \leq 15, o_3 + 3 \leq 24, o_4 + 2 \leq 24,$ and $o_5 + 6 \leq 24$. Finally, we break symmetry on the assignment variables corresponding to a group of similar
persons. In the example, the four nurses are similar since (1) they all have exactly the same unavailability periods, and since (2) no task requires a specific nurse. For each task using more than one nurse (i.e., tasks $t_1$, $t_3$, and $t_4$) this leads to a chain of strict inequalities, i.e., $n_{11} < n_{12}$, $n_{31} < n_{32}$, and $n_{41} < n_{42} < n_{43}$. Figure 3.2 depicts a solution to the problem corresponding to the assignment $o_1 = 10$, $a_1 = 1$, $s_1 = 3$, $n_{11} = 5$, $n_{12} = 6$, $a_2 = 2$, $s_2 = 4$, $n_2 = 7$, $a_3 = 1$, $s_3 = 3$, $n_{31} = 5$, $n_{32} = 6$, $a_4 = 17$, $a_4 = 1$, $s_4 = 4$, $n_{41} = 5$, $n_{42} = 6$, $n_{43} = 7$, $o_5 = 16$, $a_5 = 2$, $s_5 = 3$, $n_5 = 8$.

The entry corresponding to the keyword relaxation dimension shows how to express relaxation in the context of over-constrained problems where we have too many operations to schedule with respect to the number of anaesthetists, surgeons and nurses and to their unavailability periods.

3.7.16 ▾Assignment ➤

- assign_and_counts,
- assign_and_nvalues,
- balance,
- balance_interval,
- balance_modulo,
- balance_partition,
- bin_packing,
- bin_packing_capa,
- cardinality_atleast,
- cardinality_atmost,
- global_cardinality,
- global_cardinality_low_up,
- global_cardinality_with_costs,
- increasing_global_cardinality,
- indexed_sum,
- interval_and_count,
- interval_and_sum,
- k_alldifferent,
- max_nvalue,
- min_nvalue,
- min_size_set_of_consecutive_var,
- minimum_weight_alldifferent,
- open_global_cardinality,
- open_global_cardinality_low_up,
- ordered_global_cardinality,
- same_and_global_cardinality,
- same_and_global_cardinality_low_up,
- sum_of_weights_of_distinct_values,
- symmetric_cardinality,
- symmetric_gcc,
- weighted_partial_alldiff.

A constraint related to assignment problems (i.e., k_alldifferent), or a constraint putting a restriction on all items that are assigned to the same equivalence class or on all equivalence classes that are effectively used. Usually an equivalence class corresponds to one single value (see, e.g., the balance, bin_packing, global_cardinality, and sum_of_weights_of_distinct_values constraints), to an interval of consecutive values (see, e.g., the balance_interval, interval_and_count, and interval_and_sum constraints) or to all values that are congruent modulo a given number (see, e.g., the balance_modulo constraint). The restriction on all items that are assigned to the same equivalence class can for instance be a constraint on the number of items (see, e.g., the cardinality_atleast, cardinality_atmost, global_cardinality, and global_cardinality_low_up constraints).
constraints) or a constraint on the sum of a specific attribute (see, e.g., the bin
packing, and interval_and_sum constraints).

3.7.17 ▼Assignment dimension ➔ [12 CONS]

• assign_and_counts (attribute bin of ITEMS collection),
• assign_and_nvalues (attribute bin of ITEMS collection),
• bin_packing (attribute bin of ITEMS collection),
• bin_packing_capa (attribute bin of ITEMS collection),
• calendar (attribute machine of INSTANTS collection),
• coloured_cumulatives (attribute machine of TASKS collection),
• cumulatives (attribute machine of TASKS collection),
• diffn (attribute ori of ORTHOTOPE collection for which siz = 1),
• geost (attribute x of OBJECTS collection for which l = 1),
• geost_time (attribute x of OBJECTS collection for which l = 1),
• interval_and_count (attribute origin of TASKS collection),
• interval_and_sum (attribute origin of TASKS collection).

A constraint for handling placement problems in the broad sense involving an
assignment dimension (i.e., one of the attribute of a collection passed as argument indi-
cates the assignment dimension — the attribute is shown in parenthesis for each con-
straint). In order to illustrate the notion of assignment dimension let us first introduce
three typical examples described in Figure 3.3:

• Part (A) of Figure 3.3 considers a scheduling problem where we have both to as-
sign a task to a machine and to fix its start to a time-point, in such a way that two
tasks that overlap in time are not assigned to the same machine. In this context
the different potential machines where tasks can be assigned is called an assign-
ment dimension. This problem can be directly modelled by a cumulatives, a
diffn or a geost constraint. The corresponding three ground instances encod-
ing the example are (attributes related to the assignment dimension are shown in
bold):

  - cumulatives(
    machine − 1 origin − 2 duration − 2 end − 4 height − 1,
    machine − 3 origin − 4 duration − 3 end − 7 height − 1,
    machine − 1 origin − 7 duration − 1 end − 8 height − 1),
    {id − 1 capacity − 1,
     id − 2 capacity − 1,
     id − 3 capacity − 1})

  - diffn(
    {orth − (ori − 2 siz − 2 end − 4, ori − 1 siz − 1 end − 2),
     orth − (ori − 4 siz − 3 end − 7, ori − 3 siz − 1 end − 4),
     orth − (ori − 7 siz − 1 end − 8, ori − 1 siz − 1 end − 2)})
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

- \textbf{geost}(2, (\textit{oid} − 1 \textit{sid} − 1 \textit{x} − \{2,1\},  
\textit{oid} − 2 \textit{sid} − 2 \textit{x} − \{4,3\},  
\textit{oid} − 3 \textit{sid} − 3 \textit{x} − \{7,1\})  
\langle \textit{sid} − 1 \textit{t} − (0,0) 1 − (2,1),  
\textit{sid} − 2 \textit{t} − (0,0) 1 − (3,1),  
\textit{sid} − 3 \textit{t} − (0,0) 1 − (1,1)\rangle)

- \textbf{Part (B)} of Figure 3.3 considers a placement problem where we have both to assign a rectangle to a rectangular piece and to locate it within the selected rectangular piece. In this context the different potential rectangular pieces where rectangles can be placed is also called an assignment dimension. Note that in such placement problems the size of an object in an assignment dimension is always equal to one. This problem can be directly modelled by a \textit{diffn} or a \textit{geost} constraint. The corresponding two ground instances encoding the example are (attributes related to the assignment dimension are shown in bold):

- \textbf{diffn}(\langle \textit{orth} − \langle \textit{ori} − 2 \textit{siz} − 1 \textit{end} − 3,  
\textit{ori} − 2 \textit{siz} − 2 \textit{end} − 4,  
\textit{ori} − 2 \textit{siz} − 2 \textit{end} − 4\rangle,  
\textit{orth} − \langle \textit{ori} − 1 \textit{siz} − 1 \textit{end} − 2,  
\textit{ori} − 3 \textit{siz} − 3 \textit{end} − 6,  
\textit{ori} − 1 \textit{siz} − 2 \textit{end} − 3\rangle,  
\textit{orth} − \langle \textit{ori} − 2 \textit{siz} − 3 \textit{end} − 9,  
\textit{ori} − 6 \textit{siz} − 1 \textit{end} − 7,  
\textit{ori} − 1 \textit{siz} − 3 \textit{end} − 4\rangle)  
- \textbf{geost}(3, (\textit{oid} − 1 \textit{sid} − 1 \textit{x} − \{2,2,2\},  
\textit{oid} − 2 \textit{sid} − 2 \textit{x} − \{1,3,1\},  
\textit{oid} − 3 \textit{sid} − 3 \textit{x} − \{2,6,1\})  
\langle \textit{sid} − 1 \textit{t} − (0,0,0) 1 − (1,2,2),  
\textit{sid} − 2 \textit{t} − (0,0,0) 1 − (1,3,2),  
\textit{sid} − 3 \textit{t} − (0,0,0) 1 − (1,1,3)\rangle)

- \textbf{Part (C)} of Figure 3.3 considers a placement problem where we have both to assign a box to a container and to place it within the selected container. In this context the different potential containers where boxes can be packed is also called an assignment dimension. Note that in such placement problems the size of an object in an assignment dimension is always equal to one. This problem can be directly modelled by a \textit{diffn} or a \textit{geost} constraint. The corresponding two ground instances encoding the example are (attributes related to the assignment dimension are shown in bold):

- \textbf{diffn}(\langle \textit{orth} − \langle \textit{ori} − 1 \textit{siz} − 1 \textit{end} − 2,  
\textit{ori} − 1 \textit{siz} − 1 \textit{end} − 2,  
\textit{ori} − 1 \textit{siz} − 2 \textit{end} − 3,  
\textit{ori} − 1 \textit{siz} − 1 \textit{end} − 2\rangle,  
\textit{orth} − \langle \textit{ori} − 1 \textit{siz} − 1 \textit{end} − 2,  
\textit{ori} − 1 \textit{siz} − 1 \textit{end} − 2\rangle)
In summary, within the context of placement problems that use a constraint like `diffn` or `geost`, the coordinate of an object in the assignment dimension corresponds to the resource to which the object is assigned. Note that the size of an object in the assignment dimension is always set to 1. This stems from the fact that an object is assigned to a single resource.

Using constraints like `coloured_cumulatives`, `cumulatives`, `diffn`, `geost` or `geost_time` allows to model directly with one single global constraint such problems without knowing in advance to which machine, to which rectangular piece, to which container, a task, a rectangle, a box will be assigned. For each object the potential values of its assignment variable provide the machines, the rectangular pieces, the containers to which the object can possibly be assigned. Note that this allows to avoid 0-1 variables for modelling such problems.

Within constraints like `interval_and_count` or `interval_and_sum` the concept of assignment dimension is extended from the fact that a variable is assigned a value to the fact that a variable is assigned an interval (i.e., a value in an interval).
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.18 Assignment to the same set of values

- bin_packing
- bin_packing_capa
- coloured_cumulatives
- cumulatives
- diffn
- elem
- element
- geoסט
- geoност
time.

Given several mutually disjoint finite sets of values $S_1, S_2, \ldots, S_m$ ($m > 1$) such that $S_1 \cup S_2 \cup \cdots \cup S_m = \{1, 2, \ldots, p\}$, as well as a set of variables $V_1, V_2, \ldots, V_n$, the assignment to the same set of values subproblem consists of assigning all variables $V_1, V_2, \ldots, V_n$ values that belong to the same set $S_i$ ($1 \leq i \leq m$). As we will see later on, this subproblem arises naturally in many resource assignment problems where an additional constraint between variables $V_1, V_2, \ldots, V_n$ also has to hold. The subproblem can be modelled as a conjunction of element constraints of the form:

$element(V_1, \{\text{set}_1, \text{set}_2, \ldots, \text{set}_p\}, \text{SET_INDEX}) \land$

$element(V_2, \{\text{set}_1, \text{set}_2, \ldots, \text{set}_p\}, \text{SET_INDEX}) \land$

$\cdots$

$element(V_n, \{\text{set}_1, \text{set}_2, \ldots, \text{set}_p\}, \text{SET_INDEX}),$

where $\text{set}_i = j$ if and only if $i \in S_j$ (i.e., $\text{set}_i$ corresponds to the index of the set that contains value $i$). The $k$-th element constraint expresses that variable $V_k$ is assigned a value in set $S_{\text{SET_INDEX}}$. Since all element constraints share the same third argument this enforces all variables $V_1, V_2, \ldots, V_n$ to be assigned a value within the same set. Note that this conjunction of element constraints corresponds to a Berge-acyclic constraint network. Consequently, one can achieve arc-consistency on this subproblem provided that arc-consistency is enforced on each element constraint.

As an example, consider the four sets of values $S_1 = \{3, 4, 8\}$, $S_2 = \{1, 5\}$, $S_3 = \{6, 7\}$, and $S_4 = \{2, 9\}$, as well as four variables $w, x, y$ and $z$ that all must be assigned values that belong to the same set $S_s$ ($1 \leq s \leq 4$). This leads to the following conjunction of element constraints:

$element(w, \{2, 4, 1, 1, 2, 3, 3, 1, 4\}, s) \land$

$element(x, \{2, 4, 1, 1, 2, 3, 3, 1, 4\}, s) \land$

$element(y, \{2, 4, 1, 1, 2, 3, 3, 1, 4\}, s) \land$

$element(z, \{2, 4, 1, 1, 2, 3, 3, 1, 4\}, s).$

The first entry of the table $(2, 4, 1, 1, 2, 3, 3, 1, 4)$ is set to 2 since value 1 belongs to $S_2$. Similarly, the second entry of the table is set of 4 since value 2 belongs to $S_4$. The same logic is used for building up the other entries of the table.

A generalisation of this subproblem consists in lifting the restriction that the sets of values $S_1, S_2, \ldots, S_m$ are mutually disjoint. The only change to adapt the previous model is to replace within each element constraint each value $val_i$ ($1 \leq i \leq p$) by a value variable $Val_i$ (i.e., each value of a value variable represents a set containing $i$), where $j \in \text{dom}(Val_i)$ if and only if $i \in S_j$. Distinct element constraints will get distinct value variables. As an example, consider the previous four sets of values
where we add value 2 to \( S_1 \) and value 5 to \( S_3 \). We now have the sets \( S_1 = \{2, 3, 4, 8\} \), \( S_2 = \{1, 5\} \), \( S_3 = \{5, 6, 7\} \), and \( S_4 = \{2, 9\} \) where value 2 occurs both in \( S_1 \) and \( S_4 \), and value 5 appears both in \( S_2 \) and \( S_3 \). This leads to the following conjunction of constraints:

\[
\begin{align*}
& \text{in}(a_1, (1, 4)) \land \text{in}(b_1, (2, 3)) \land \text{element}(w, (2, a_1, 1, 1, b_1, 3, 3, 1, 4), s) \land \\
& \text{in}(a_2, (1, 4)) \land \text{in}(b_2, (2, 3)) \land \text{element}(x, (2, a_2, 1, 1, b_2, 3, 3, 1, 4), s) \land \\
& \text{in}(a_3, (1, 4)) \land \text{in}(b_3, (2, 3)) \land \text{element}(y, (2, a_3, 1, 1, b_3, 3, 3, 1, 4), s) \land \\
& \text{in}(a_4, (1, 4)) \land \text{in}(b_4, (2, 3)) \land \text{element}(z, (2, a_4, 1, 1, b_4, 3, 3, 1, 4), s).
\end{align*}
\]

The domain of the variables \( a_i \) (\( 1 \leq i \leq 4 \)) associated with the second entry of the table of the \text{element} constraint is set to 1 and 4 since value 2 belongs to \( S_1 \) and to \( S_4 \). Similarly, the domain of variables \( b_i \) (\( 1 \leq i \leq 4 \)) associated with the fifth entry is set to 2 and 3 since value 5 belongs to \( S_2 \) and \( S_3 \). Note that, since variables \( a_1, a_2, a_3, b_1, b_2, b_3, b_4 \) are distinct, the corresponding constraint network is still \text{Berge-acyclic}. We now provide an alternative model where the \( i^{th} \) entry of the table of the \( k^{th} \) (\( 1 \leq k \leq n \)) \text{element} constraint corresponds to a variable \( S_{k,i} \) for which the initial domain is the set of values that belong to \( S_i \) (\( 1 \leq i \leq m \)). We have a conjunction of \text{element} constraints of the form:

\[
\begin{align*}
& \text{element}(\text{SET\_INDEX}, (S_{11}, S_{12}, \ldots, S_{1m}), V_1) \land \\
& \text{element}(\text{SET\_INDEX}, (S_{21}, S_{22}, \ldots, S_{2m}), V_2) \land \\
& \cdots \\
& \text{element}(\text{SET\_INDEX}, (S_{n1}, S_{n2}, \ldots, S_{nm}), V_n),
\end{align*}
\]

where \( \text{SET\_INDEX} \) is a variable ranging from 1 to \( m \) designating the selected set. This model perhaps seems more natural. However unlike the first model, when the sets \( S_1, S_2, \ldots, S_m \) are mutually disjoint, it enforces using variables instead of integers in the table of each \text{element} constraint. Like the first model, it is \text{Berge-acyclic}.

Now that we have presented two dual models for the assignment to the same set of values subproblem, we introduce the \text{resource assignment with groups} pattern, which uses several instances of the subproblem. We consider a set of tasks \( t_1, t_2, \ldots, t_q \) (\( q \geq 1 \)) tasks, where each task \( t_i \) (\( 1 \leq i \leq q \)) is decomposed into \( s_i \) subtasks \( t_{ij} \) (\( 1 \leq j \leq s_i \)). All subtasks that belong to one and the same task should be assigned the same group, where groups are defined by the finite sets of values \( S_1, S_2, \ldots, S_m \) (\( m > 1 \)) introduced early on. For this purpose an \text{assignment variable} and a \text{group variable} are respectively associated with each subtask and each task. In addition, we also have a resource constraint involving all subtasks. This resource constraint has an \text{assignment dimension} corresponding to the different resources where subtasks can potentially be assigned. To each resource corresponds a value of \( S_1 \cup S_2 \cup \cdots \cup S_m = \{1, 2, \ldots, p\} \). Depending on the kind of resource constraint we have (e.g., \text{bin\_packing}, \text{cumulatives}, \text{diffn}, \text{geost})

Each subtask has additional attributes that characterise it. For instance, if we have a \text{bin\_packing} constraint then, in addition to the assignment dimension that corresponds to the bin where a subtask will be assigned, we also have a weight attribute that describes how much space a subtask uses in a bin. Then the \text{bin\_packing} constraint expresses that the total weight of the subtasks in each bin does not exceed a given fixed capacity.

\footnote{The table corresponds to the second argument of the \text{element} constraint.}
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

Figure 3.4: Illustration of the constraint network associated with the resource assignment with groups pattern.

Figure 3.4 illustrates the constraint network associated with the resource assignment with groups pattern. Lower circles represent the group variables associated with the different tasks (three tasks in the example), while all the other circles represent the attributes of the different subtasks (i.e., vertically aligned circles correspond to the attributes of a given subtask). All circles that are associated with the same task are coloured with the same colour. As said before, each subtask has an attribute that gives the resource to which the resource will be assigned (called assignment variables in Figure 3.4) and other attributes that depend of the resource constraint we are considering (called other subtask attributes in the Figure). Each blue rounded box corresponds to a group constraint that enforces all subtasks of a given task to be assigned the same group (i.e., within this blue box, each line-segment represents an element constraint of the assignment to the same set of values subproblem). Finally, the pink rounded box represents the resource constraint that involves all subtasks.

Before illustrating the resource assignment with groups pattern on a particular resource constraint, we first point out a potential weakness that is inherent to this constraint network, no matter what kind of resource constraint we use. When pruning the assignment variables, the resource constraint will ignore the groups (since the resource constraint is not aware of the element constraints) and will therefore miss some filtering. Consequently one may complete the constraint network by some global necessary conditions. When fixing variables it may be a good idea to fix all variables that are attached to one task before considering the next task. While fixing the variables of a task one may first assign its group variable, and second fix the variables of its subtasks;
again we may prefer to fix all variables of a subtask before considering the next subtask. The idea behind this heuristics is to try to avoid the creation of infeasible subproblems during search.

Figure 3.5: Illustration of the resource assignment with groups pattern in the context of a bin_packing resource constraint

Figure 3.5 illustrates the resource assignment with groups pattern when the resource constraint corresponds to a bin_packing constraint. As in Figure 3.4, we have three tasks $t_1$, $t_2$ and $t_3$ such that:

- Three subtasks $t_{11}$, $t_{12}$ and $t_{13}$ are associated with task $t_1$. They have a respective weight of 2, 3 and 2 and are coloured in green in Figure 3.5.
- Two subtasks $t_{21}$ and $t_{22}$ of respective weight 2 and 3 are associated with task $t_2$. They are coloured in yellow.
- Two subtasks $t_{31}$ and $t_{32}$ of respective weight 2 and 1 are associated with task $t_3$. They are coloured in orange.

We consider 9 bins that are partitioned into four groups of bins $S_1 = \{3, 4, 8\}$ (coloured in light blue in Figure 3.5), $S_2 = \{1, 5\}$ (coloured in light green), $S_3 = \{6, 7\}$ (coloured in light brown), and $S_4 = \{2, 9\}$ (coloured in light violet), and enforce that all subtasks that are associated with the same task are assigned the same group of bins. In addition, the sum of the weights of the subtasks that are assigned the same bin should not exceed the capacity of the bins, 5 in our example. Within the solution depicted by Figure 3.5, all constraints are satisfied since:

1. For each task, all its subtasks are assigned the same group of bins (i.e., all subtasks that have the same colour are assigned bins with the same colour).

2. The capacity constraint of each bin is respected (i.e., the overall capacity of five is never exceeded).

The conjunction of constraints corresponding to this solution is:

\[
\text{element}(4, \langle 2, 4, 1, 1, 2, 3, 3, 1, 4 \rangle, 1) \land \\
\text{element}(8, \langle 2, 4, 1, 1, 2, 3, 3, 1, 4 \rangle, 1) \land
\]
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

\[ \text{element}(4, (2, 4, 1, 1, 2, 3, 1, 4), 1) \land \\
\text{element}(2, (2, 4, 1, 1, 2, 3, 1, 4), 4) \land \\
\text{element}(9, (2, 4, 1, 1, 2, 3, 1, 4), 4) \land \\
\text{element}(2, (2, 4, 1, 1, 2, 3, 1, 4), 4) \land \\
\text{bin} \; \text{packing}(5, (\text{bin} - 4 \; \text{weight} - 2, \text{bin} - 8 \; \text{weight} - 3, \text{bin} - 4 \; \text{weight} - 2, \\
\text{bin} - 2 \; \text{weight} - 2, \text{bin} - 9 \; \text{weight} - 3, \\
\text{bin} - 2 \; \text{weight} - 2, \text{bin} - 9 \; \text{weight} - 1)). \]

For each subtask we have one \textit{element} constraint expressing that all subtasks of a given task are assigned the same group of bins. Finally we have one \textit{bin} \; \text{packing} constraint expressing the capacity condition.

We now quote two concrete examples of the resource assignment with groups pattern:

- Given, (1) a set of jobs where each job is decomposed into a set of tasks, each of them requiring an amount of memory for its execution, as well as (2) a set of potential machines, each of them having a given available memory, organised into clusters, the problem is to:
  - Assign all tasks to machines in such a way that tasks from the same job are assigned the same cluster.
  - Fulfil the available memory constraint of each machine (i.e., the sum of the required memory of all tasks that are assigned a given machine does not exceed the machine available memory).

This concrete problem corresponds to the example presented in Figure 3.5.

- Given, (1) a set of maintenance activities where each maintenance activity is decomposed into a set of subactivities, each of them requiring a specific skill and a given duration, as well as (2) a set of technicians, each of them having its own home base location and its own working time window, the problem is to:
  - Assign all maintenance subactivities to technicians in such a way that subactivities from the same activity are assigned technicians that have the same home base location (i.e., each subactivity should be assigned one single technician).
  - Fulfil both the working time window of each technician, and the fact that subactivities that are assigned the same technician should not overlap (i.e., subactivities must be assigned a starting time and preemption is not allowed).

In this problem we replace the \textit{bin} \; \text{packing} constraint by a \textit{cumulatives}(\text{TASKS}, \text{MACHINES}, \leq) constraint. To each item of the \text{TASKS} collection corresponds a subactivity, such that:

  - Its \textit{machine} attribute designates the potential technicians that can take care of this subactivity.
– Its origin attribute corresponds to the timepoint where the subactivity will actually start.
– Its duration attribute is set to the duration of the corresponding subactivity.
– Its end attribute is equal to origin + duration.
– Its height attribute is set to one.

In addition to the subactivities, we also introduce for each technician two fixed dummy tasks for preventing assigning subactivities outside its time window. To each item of the MACHINES collection corresponds a technician, such that:

– Its id attribute is a fixed integer that uniquely identifies the technician.
– Its capacity attribute is set to one since it cannot perform more than one subactivity at any timepoint.

### 3.7.19 ▼ At least ➔ [3 CONS]

- atleast.
- cardinality_atleast.
- open_atleast.

A constraint enforcing that one or several values occur a minimum number of time within a given collection of domain variables.

### 3.7.20 ▼ At most ➔ [5 CONS]

- atmost.
- cardinality_atmost.
- cardinality_atmost_partition.
- multi_inter_distance.
- open_atmost.

A constraint enforcing that one or several values occur a maximum number of time within a given collection of domain variables.

### 3.7.21 ▼ Automaton ➔ [122 CONS]

- alldifferent.
- alldifferent_except_0.
- alldifferent_interval.
- alldifferent_modulo.
- alldifferent_on_intersection.
- alldifferent_same_value.
- among.
- among_diff_0.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

- among_interval,
- among_low_up,
- among_modulo,
- and,
- arith,
- arith_or,
- arith_sliding,
- assign_and_counts,
- atleast,
- atmost,
- balance,
- balance_interval,
- balance_modulo,
- between_min_max,
- bin_packing,
- cardinality_atleast,
- cardinality_atmost,
- change,
- change_continuity,
- change_pair,
- change_vectors,
- circular_change,
- clause_and,
- clause_or,
- cond_lex_cost,
- cond_lex_greater,
- cond_lex_greatereq,
- cond_lex_less,
- cond_lex_lesseq,
- consecutive_groups_of_ones,
- count,
- counts,
- cumulative,
- cyclic_change,
- cyclic_change_joker,
- decreasing,
- deepest_valley,
- differ_from_at_least_k_pos,
- disjoint,
- distance_change,
- domain_constraint,
- elem,
- elem_from_to,
- element,
- elementn,
- element_greatereq,
- element_leq,
- element_matrix,
- element_sparse,
- equivalent,
- exactly,
- global_cardinality,
- global_contiguity,
- group,
- group_skip_isolated_item,
- highest_peak,
- imply,
- in,
- in_interval,
- in_same_partition,
- increasing,
- increasing_global_cardinality,
- increasing_value,
- inflexion,
- int_value_precede,
- int_value_precede_chain,
- interval_and_count,
- interval_and_sum,
- inverse,
- ith_pos_different_from_0,
- length_first_sequence,
- length_last_sequence,
- lex_between,
- lex_differential,
- lex_equal,
- lex_greater,
- lex_greatereq,
- lex_less,
- lex_leq,
- longest_change,
- max_value,
- maximum,
- min_n,
- min_value,
- minimum,
A constraint for which the catalogue provides a deterministic automaton for the ground case. This automaton can usually be used for deriving mechanically a filtering algorithm for the general case. We have the following three types of deterministic automata:

- Deterministic automata without counters and without array of counters,
- Deterministic automata with counters but without array of counters,
- Deterministic automata with array of counters and possibly with counters.

Figure 3.6 shows three automata respectively associated with the global_contiguity, the exactly and the alldifferent constraints. These automata correspond to the three types we described above.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.22 ▶ Automaton with array of counters ➲

- alldifferent,
- alldifferent_except_0,
- alldifferent_interval,
- alldifferent_modulo,
- alldifferent_on_intersection,
- alldifferent_same_value,
- assign_and_counts,
- balance,
- balance_interval,
- balance_modulo,
- bin_packing,
- cardinality_atleast,
- cardinality_atmost,
- cumulative,
- disjoint,
- global_cardinality,
- interval_count,
- interval_and_sum,
- inverse,
- max_nvalue,
- min_n,
- min_nvalue,
- nvalue,
- same,
- used_by.

A constraint for which the catalogue provides a deterministic automaton with array of counters and possibly with counters.

3.7.23 ▶ Automaton with counters ➲

- among,
- among_diff_0,
- among_interval,
- among_low_up,
- among_modulo,
- arith_sliding,
- atleast,
- atmost,
- change,
- change_continguity,
- change_pair,
- change_vectors,
- circular_change,
- count,
- counts,
- cyclic_change,
- cyclic_change_joker,
- deepest_valley,
- differ_from_at_least_k_pos,
- distance_change,
- exactly,
- group,
- group_skip_isolated_item,
- highest_peak,
- inflexion,
- ith_pos_diffrent_from_0,
- length_first_sequence,
- length_last_sequence,
- longest_change,
- peak,
- sliding_card_skip0,
- smooth,
- valley.
A constraint for which the catalogue provides a deterministic automaton with counters but without array of counters.

3.7.24 Automaton without counters

- and
- arith
- arith_or
- between_min_max
- clause_and
- clause_or
- cond_lex_cost
- consecutive_groups_of_ones
- decreasing
- domain_constraint
- elem
- elem_from_to
- element
- element_greatereq
- element_leq
- element_matrix
- element_sparse
- elementn
- equivalent
- global_contiguity
- imply
- in
- in_interval
- in_same_partition
- increasing
- increasing_global_cardinality
- increasing_value
- int_value_precede
- int_value_precede_chain
- lex_between
- lex_different
- lex_equal
- lex_greater
- lex_greatereq
- lex_less
- lex_leq
- maximum
- minimum
- minimum_except_0
- minimum_greater_than
- nand
- next_element
- no_peak
- no_valley
- nor
- not_all_equal
- not_in
- open_max
- open_min
- or
- pattern
- sequence_folding
- stage_element
- stretch_path
- stretch_path_partition
- strictly_decreasing
- strictly_increasing
- two_orth_are_in_contact
- two_orth_do_not_overlap
- xor

A constraint for which the catalogue provides a deterministic automaton without counters and without array of counters. Note that the filtering algorithm [286] and the reformulation [34] that were initially done in the context of deterministic automata can
also be used for non-deterministic automata. All these constraints are also annotated with the keyword reified automaton constraint.

3.7.25 ▼Autoref ➔

- global_cardinality.

A constraint that allows for modelling the autoref problem with one single constraint. The autoref problem is a generalisation of the problem of finding a magic series and can be defined in the following way. Given an integer \( n > 0 \) and an integer \( m \geq 0 \), the problem is to find a non-empty finite series \( S = (s_0, s_1, \ldots, s_n, s_{n+1}) \) such that (1) there are \( s_i \) occurrences of \( i \) in \( S \) for each integer \( i \) ranging from 0 to \( n \), and (2) \( s_{n+1} = m \). This leads to the following model:

\[
\text{global_cardinality} \begin{cases} 
\langle \text{var} - s_0, \text{var} - s_1, \ldots, \text{var} - s_n, \text{var} - m \rangle, \\
\text{val} - 0 & \text{noccurrence} - s_0, \\
\text{val} - 1 & \text{noccurrence} - s_1, \\
\vdots & \\
\text{val} - n & \text{noccurrence} - s_n 
\end{cases}
\]

23, 2, 2, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 and 23, 3, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 5 are the two unique solutions for \( n = 27 \) and \( m = 5 \).

3.7.26 ▼Balanced assignment ➔

- balance, balance_interval, balance_modulo, balance_partition, deviation, maximum, spread.

A constraint to obtain a balanced assignment over a set of domain variables. Given a set of domain variables \( \{x_1, x_2, \ldots, x_n\} \), some classical balance criteria reported in [347] are:

- The maximum value, i.e., the maximum value over \( x_i (i \in [1, n]) \) can be modelled with a maximum constraint.

- The maximum deviation, i.e., the maximum value over \( x_i - \frac{\sum_{j \in [1,n]} x_j}{n} \) \( (i \in [1, n]) \).

- The total deviation, i.e., \( \sum_{i \in [1,n]} \left| x_i - \frac{\sum_{j \in [1,n]} x_j}{n} \right| \) can be modelled with a deviation constraint [350, 348].
The total quadratic deviation, i.e., $\sum_{i \in [1,n]} \left( x_i - \frac{\sum_{j \in [1,n]} x_j}{n} \right)^2$ can be modelled with a spread constraint [287, 349].

3.7.27 ▼ Balanced tree ➤

- `tree_range`.

A constraint that allows for expressing that we want to cover a digraph by one (or more) balanced tree. A balanced tree is a tree where no leaf is much farther away than a given threshold from the root than any other leaf. The distance between a leaf and the root of a tree is the number of vertices on the path from the root to the leaf.

3.7.28 ▼ Berge-acyclic constraint network ➤

- `among`,
- `and`,
- `arith`,
- `arith_or`,
- `change`,
- `change_vectors`,
- `clause_and`,
- `clause_or`,
- `cond_lex_cost`,
- `cond_lex_greater`,
- `cond_lex_greatereq`,
- `cond_lex_less`,
- `cond_lex_leq`,
- `consecutive_groups_of_ones`,
- `elementn`,
- `equivalent`,
- `global_contiguity`,
- `imply`,
- `in_interval`,
- `increasing_global_cardinality`,
- `increasing_nvalue`,
- `int_value_precede`,
- `int_value_precede_chain`,
- `lex_between`,
- `lex_different`,
- `lex_equal`,
- `lex_greater`,
- `lex_greatereq`,
- `lex_less`,
- `lex_leq`,
- `nand`,
- `nor`,
- `or`,
- `pattern`,
- `smooth`,
- `stretch_path`,
- `stretch_path_partition`,
- `two_orth_are_in_contact`,
- `two_orth_do_not_overlap`,
- `xor`.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

A constraint for which the decomposition associated with its usually counter-free deterministic automaton is Berge-acyclic. Arc-consistency for a Berge-acyclic constraint network is achieved by making each constraint of the corresponding network arc-consistent [22]. A constraint network for which the corresponding intersection graph does not contain any cycle and such that, for any pair of constraints, the two sets of involved variables share at most one variable is Berge-acyclic, where Berge-acyclic is defined by the following two conditions:

1. There is no more than one shared variable between any pair of constraints,

2. The hypergraph corresponding to the constraint network does not contain any cycle. Within [54, page 150] a cycle of an hypergraph is defined as “Let \( H \) be an hypergraph on a finite set \( X \). A cycle of length \( k (k \geq 2) \) is a sequence \( (x_1, E_1, x_2, E_2, x_3, \ldots, E_k, x_1) \) such that (1) \( E_1, E_2, \ldots, E_k \) are distinct edges of \( H \), (2) \( x_1, x_2, \ldots, x_k \) are distinct vertices of \( H \), (3) \( x_i, x_{i+1} \in E_i \) \((i = 1, 2, \ldots, k - 1)\), (4) \( x_k, x_1 \in E_k\).”

The intersection graph of a constraint network is built in the following way: to each vertex corresponds a constraint and there is an edge between two vertices if and only if the sets of variables involved in the two corresponding constraints intersect.

Figure 3.7: Illustration of Berge-acyclic constraint network

Parts (A), (B), (C) and (D) of Figure 3.7 provide four examples of constraint networks, while parts (E), (F), (G) and (F) give their corresponding intersection graph.

1. The constraint network corresponding to part (A) is Berge-acyclic since its corresponding intersection graph (E) does not contain any cycle and since there is no more than one shared variable between any pair of constraints.

---

8 All the above constraints, except among, change, and smooth have a deterministic counter-free automaton. The among constraint has an automaton involving one counter and one single state, see Figure 5.24, while the change and the smooth constraints have a counter-free non deterministic automaton, see Figures 5.99 and 5.548.
2. The constraint network corresponding to part (B) is not Berge-acyclic since its hypergraph (B) contains a cycle.

3. The constraint network corresponding to (C) is also not Berge-acyclic since its third and fourth constraints share more than one variable.

4. Finally, the constraint network corresponding to (D) is Berge acyclic, even if its intersection graph (H) has a cycle, since its hypergraph (D) does not contain any cycle and since there is no more than one shared variable between any pair of constraints.

If we execute the filtering algorithm of each constraint of a Berge-acyclic constraint network $\mathcal{N}$ in an appropriate order then each constraint needs only to be waken twice in order to reach the fix-point. A static ordering for waking the constraints of $\mathcal{N}$ can be determined as follows:

- Consider the intersection graph $G_{\mathcal{N}}$ associated with the constraint network $\mathcal{N}$. We perform a topological sort on $G_{\mathcal{N}}$, which always first selects in the remaining part of $G_{\mathcal{N}}$ a vertex (i.e., a constraint) which has only one single neighbour. Let $C_1, C_2, \ldots, C_n$ be the constraints successively removed by the topological sort.

- Then, the static ordering for reaching a fix-point is given by the sequence $C_1, C_2, \ldots, C_{n-1}, C_n, C_{n-1}, \ldots, C_2, C_1$, where each constraint is woken at most twice. This can be done by using the notion of propagator group \[229\]. This facility allows the user of a solver controlling the order of execution of a group of constraints. Propagator groups are useful, both to guaranty the theoretical worst case complexity of a decomposition, and for accelerating convergence to the fix-point in practice.

If we consider the Berge-acyclic constraint network given by Part (D) of Figure 3.7 an appropriate order for waking the constraints could for instance be CTR$_1$, CTR$_4$, CTR$_2$, CTR$_3$, CTR$_2$, CTR$_4$, CTR$_1$.

For heuristics that try creating a Berge-acyclic constraint network see also the keyword heuristics and Berge-acyclic constraint network.
3.7. **KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS**

### 3.7.29 ▶ Binary constraint ➤

- abs_value,
- divisible,
- divisible_or,
- element_greatereq,
- element_leseq,
- element_sparse,
- eq,
- eq_cst,
- eq_set,
- geq,
- geq_cst,
- gt,
- in_same_partition,
- leq,
- leq_cst,
- lt,
- neq,
- neq_cst,
- opposite_sign,
- in_interval_reified,
- remainder,
- same_sign,
- sign_of,
- stage_element,
- sum_set.

A constraint involving only two variables.

### 3.7.30 ▶ Bioinformatics ➤

- all_differ_from_at_least_k_pos,
- sequence_folding,
- stable_compatibility.

Denotes that, for a given constraint, either there is a reference to its uses in Bioinformatics, or it was inspired by a problem from the area of Bioinformatics.
3.7.31 Bipartite

- alldifferent_on_intersection,
- allperm,
- among_low_up,
- among_var,
- arith_or,
- assign_and_counts,
- assign_and_nvalues,
- bin_packing,
- bipartite,
- cardinality_atleast,
- cardinality_atmost,
- cardinality_atmost_partition,
- change,
- change_continuity,
- change_pair,
- change_partition,
- common,
- common_interval,
- common_modulo,
- common_partition,
- correspondence,
- counts,
- cyclic_change,
- cyclic_change_joker,
- decreasing,
- inverse_within_range,
- lex_equal,
- two_orth_do_not_overlap,
- uses.

Denotes that a constraint is defined by one graph constraint for which the final graph is bipartite.

3.7.32 Bipartite matching

- alldifferent,
- alldifferent_between_sets,
- alldifferent_cst,
- atleast_nvalue,
- disjoint,
- lex_alldifferent.

Figure 3.8: A bipartite graph and one of its bipartite matching

Denotes that, for a given constraint, a bipartite matching algorithm can be used within its filtering algorithm. A bipartite matching is a subgraph that pairs every vertex.
of a bipartite graph with exactly one other vertex. A bipartite graph is a graph for which the set of vertices can be partitioned in two parts such that no two vertices in the same part are joined by an edge. Part (A) of Figure 3.8 shows a bipartite graph with a possible division of the vertices in black and white, while part (B) depicts with a thick line a bipartite matching of this graph.

3.7.33 Bipartite matching in convex bipartite graphs ➤ [2 CONS]

- alldifferent.
- alldifferent_cst.

Denotes that, for a given constraint, a bipartite matching algorithm using Glover’s rule for constructing a maximum matching of a convex bipartite graph can be used. Given a convex bipartite graph $G = (U, V, E)$ where $U = \{u_1, u_2, \ldots, u_n\}$ and $V = \{v_1, v_2, \ldots, v_m\}$, Glover [179] showed how to efficiently compute a maximum matching in such a graph:

1. First start with the empty matching.
2. Second for each vertex $v_j$ of $V$, ($j = 1, 2, \ldots, m$), if $v_j$ has still a free neighbour in $U$, then add to the current matching the edge $(u_i, v_j)$ for which $u_i$ is free and $\alpha_i = \max\{j : (x_i, y_j) \in E, y_j \in V\}$ is as small as possible.

3.7.34 Boolean channel ➤ [1 CONS]

- domain_constraint.

A constraint that allows for making the link between a set of 0-1 variables $B_1, B_2, \ldots, B_n$ and a domain variable $V$. It enforces a condition of the form $V = i \Leftrightarrow B_i = 1$. 
3.7.35 ▼ Boolean constraint ➜ [9 CONS]

- and,
- clause_and,
- clause_or,
- equivalent,
- imply,
- nand,
- nor,
- or,
- xor.

A Boolean constraint is a constraint of the form \( v = f(v_1, \ldots, v_n) \) \((n \geq 2)\) where \( v, v_1, \ldots, v_n \) are 0-1 variables and where \( f(v_1, \ldots, v_n) \) is a logical expression involving connectors, such as \( \neg, \lor, \) or \( \land \).

3.7.36 ▼ Border ➜ [1 CONS]

- period.

A constraint that can be related to the notion of border, which we define now. Given a sequence \( s = urv \), \( r \) is a prefix of \( s \) when \( u \) is empty, \( r \) is a suffix of \( s \) when \( v \) is empty, \( r \) is a proper factor of \( s \) when \( r \neq s \). A border of a non-empty sequence \( s \) is a proper factor of \( s \), which is both a prefix and a suffix of \( s \). We have that the smallest period of a sequence \( s \) is equal to the size of \( s \) minus the length of the longest border of \( s \).

3.7.37 ▼ Bound-consistency ➜ [19 CONS]

- all_different,
- all_min_dist,
- atmost1,
- atmost_nvalue,
- nvalue,
- global_cardinality,
- global_cardinality_low_up,
- increasing_sum,
- k_all_different,
- multi_inter_distance,
- same,
- same_and_global_cardinality_low_up,
- sliding_sum,
- soft_all_equal_max_var,
- soft_all_equal_min_ctr,
- sort,
- sum_free,
- sum_ofIncrements,
- used_by.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

Denotes that, for a given constraint, there is a filtering algorithm or a reformulation in term of other constraints that ensures bound-consistency for its domain variables. A filtering algorithm or a reformulation ensure bound-consistency for a given constraint \( ctr \) using distinct domain variables if and only if for every domain variable \( V \) of \( ctr \):

- There exists at least one solution for \( ctr \) such that \( V = V \) and every other domain variable \( W \) of \( ctr \) is assigned to a value located in its range \([W, W]\).
- There exists at least one solution for \( ctr \) such that \( V = V \) and every other domain variable \( W \) of \( ctr \) is assigned to a value located in its range \([W, W]\).

This consistency is called bound(Z) consistency in [55]. One of its interest is that it sometimes gives the opportunity to come up with a filtering algorithm that has a lower complexity than the algorithm that achieves arc-consistency. Discarding holes from the domain variables usually leads to graphs with a specific structure for which one can take advantage in order to derive more efficient graph algorithms. Filtering algorithms that achieve bound-consistency can also be used in a pre-processing phase before applying a more costly filtering algorithm that achieves arc-consistency.

Note that there is a second definition of bound-consistency, called bound(D) consistency in [55], where the range \([W, W]\) is replaced by the domain of the variable \( W \). However within the context of global constraints most filtering algorithms do not refer to this second definition.

Finally, within the context of constraints involving only set variables, bound-consistency is defined in the following way. A constraint \( ctr \) defined on distinct set variables is bound-consistent if and only if for every pair \((V, v)\) such that \( V \) is a set variable of \( ctr \) and \( v \) an integer value, if \( v \in V \) then \( v \) belongs to the set assigned to \( V \) in all solutions to \( ctr \) and if \( v \in V \setminus V \) then \( v \) belongs to the set assigned to \( V \) in at least one solution and is excluded from this set in at least one solution.

3.7.38 Business rules ➤ [3 CONS]

- cycle.
- diffn.
- geost.

Denotes that a dedicated language was introduced within an argument of a global constraint for directly specifying a specific type of business rules:

- The cycle constraint was extended in order to accept rules specifying forbidden sequences of vertices within each cycle [79].
- The diffn constraint was extended in order to accept calendar rules specifying the way tasks can be interrupted or not on each resource [23]. This was done

\footnote{In the context of the nvalue constraint, bound-consistency is only achieved if and only if, the minimum of the variable that denotes the number of distinct values is not constrained at all. In the context of the k_alldifferent constraint, bound-consistency is only achieved when we have two overlapping alldifferent constraints, see [69] for more details.}
since many real scheduling problems have not only to consider disjunctive and assignment constraints, but also operational rules expressing how tasks can be interrupted.

- The geost constraint was extended in order to directly accept a great variety of packing and placement rules [93].

### 3.7.39 Centered cyclic(1) constraint network(1) ➤ [9 CONS]

- between_min_max,
- domain_constraint,
- in,
- maximum,
- minimum,
- minimum_except_0,
- not_in,
- open_maximum,
- open_minimum.

![Hypergraph associated with a centered cyclic(1) constraint network(1)](image)

Figure 3.9: Hypergraph associated with a centered cyclic(1) constraint network(1)

A constraint network corresponding to the pattern depicted by Figure 3.9. Circles depict variables, while arcs are represented by a set of variables. Grey circles correspond to optional variables. All pairs of constraints have at most one variable in common.

### 3.7.40 Centered cyclic(2) constraint network(1) ➤ [8 CONS]

- elem,
- element,
- element_greatereq,
- element_lesseq,
- element_sparse,
- in_same_partition,
- minimum_greater_than,
- stage_element.

A constraint network corresponding to the pattern depicted by Figure 3.10. Circles depict variables, while arcs are represented by a set of variables. Grey circles correspond to optional variables.
3.7. **KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS**

3.7.41 **Centered cyclic(3) constraint network**

- element_matrix
- next_element

A constraint network corresponding to the pattern depicted by Figure 3.11. Circles depict variables, while arcs are represented by a set of variables. Grey circles correspond to optional variables.

3.7.42 **Channel routing**

- connect_points

A constraint that can be used for modelling *channel routing* problems. *Channel routing* consists of creating a layout in a rectangular region of a VLSI chip in order to link together the terminals of different modules of the chip. Connections are usually made by wire segments on two different layers: horizontal wire segments on the first layer are placed along lines called tracks, while vertical wire segments on the second layer connect terminals to the horizontal wire segments, with vias at the intersection.
3.7.43 ▼ Channelling constraint ➔ [8 CONS]

- calendar,
- domain_constraint,
- inverse,
- inverse_offset,
- inverse_set,
- inverse_within_range,
- link_set_to_booleans,
- same.

Constraints that allow for linking two models of the same problem [195]. Usually channelling constraints show up in the following context:

- When a problem can be modelled by using different types of variables (e.g., 0-1 variables, domain variables, set variables),
- When a problem can be modelled by using two distinct matrices of variables representing the same information redundantly,
- When, in a problem, the roles of the variables and the values can be interchanged. This is typically the case when we have a bijection between a set of variables and the values they can take.
- When, in a problem, we use two time coordinates systems (e.g., see calendar).

3.7.44 ▼ Circuit ➔ [5 CONS]

- balance_cycle,
- circuit,
- cutset,
- cycle,
- symmetric_alldifferent.

A constraint such that its initial or its final graph corresponds to zero (e.g., cutset), one (e.g., circuit) or several (see, e.g., the cycle, and symmetric_alldifferent constraints) vertex-disjoint circuits.

3.7.45 ▼ Circular sliding cyclic(1) constraint network(2) ➔ [1 CONS]

- circular_change.

A constraint network corresponding to the pattern depicted by Figure 3.12. Circles depict variables, while arcs are represented by a set of variables.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.46 ▼ Cluster ➔ [1 CONS]

- circuit_cluster.

A constraint that partitions the vertices of an initial graph into several clusters.

3.7.47 ▼ Coloured ➔ [5 CONS]

- assign_and_counts,
- coloured_cumulative,
- coloured_cumulatives,
- cycle_card_on_path,
- interval_and_count.

A constraint with a collection where one of the attributes is a colour.

3.7.48 ▼ Compulsory part ➔ [9 CONS]

- coloured_cumulative,
- coloured_cumulatives,
- cumulative,
- cumulative_convex,
- cumulative_product,
- cumulative_two_d,
- cumulatives,
- diffn,
- disjunctive.

A constraint for which the filtering algorithm may use the notion of compulsory part. The notion of compulsory part was introduced by A. Lahrichi within the context of cumulative scheduling problems [230], [232], [231] as well as within the context of rectangles placement problems [233]. Within these two contexts, the compulsory part respectively corresponds to the intersection of all feasible instances of a task or to the intersection of all feasible instances of a rectangle.
Figure 3.13: Illustration of the notion of compulsory part

Figure 3.13 illustrates the notion of compulsory part in the context of scheduling and placement problems. The first, second and third rows respectively correspond to the cumulative \cite{299,300}, the cumulative trapeze \cite{299,300} and the diffn \cite{42} constraints. The first, second and third columns respectively correspond to the shape of the object for which we compute the compulsory part, to the extreme positions of the object and to the corresponding compulsory part.

3.7.49 ▼Conditional constraint ➤ [2 CONS]

- size_max_seq_all different,
- size_max_starting_seq_all different.

A constraint that allows for expressing that some constraints can be enforced during the enumeration phase.

3.7.50 ▼Configuration problem ➤ [1 CONS]

- element_product.

A constraint that was used for modelling configuration problems. Within the context of configuration problems \cite{376}, it is crucial to identify all variable-value pairs which do not participate to any solution. This stems from the fact that one wants typically to avoid proposing invalid choices to the user of such configuration systems.

Note also that open constraints are also useful in the context of configuration problems.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.51 ▼Connected component ➤

- alldifferent_on_intersection,
- balance_cycle,
- balance_path,
- balance_tree,
- binary_tree,
- change_continuity,
- connected,
- cycle,
- cycle_card_on_path,
- cycle_resource,
- global_contiguity,
- group,
- k_cut,
- map,
- nvalue_on_intersection,
- path,
- proper_forest,
- temporal_path,
- tree,
- tree_range,
- tree_resource.

Denotes that a constraint uses in its definition a graph property (e.g., MAX_NCC, MIN_NCC, NCC) constraining the connected components of its associated final graph.

3.7.52 ▼Consecutive loops are connected ➤

- group,
- stretch_path,
- stretch_path_partition.

Denotes that the graph constraints of a global constraint use only the PATH and the LOOP arc generators and that their final graphs do not contain consecutive vertices that are not connected together by an arc. Moreover all vertices of their final graphs have a loop.

3.7.53 ▼Consecutive values ➤

- max_size_set_of_consecutive_var,
- min_size_set_of_consecutive_var,
- nset_of_consecutive_values.

A constraint for which the definition involves the notion of consecutive values assigned to the variables of a collection of domain variables.
3.7.54 ▼ Constraint between two collections of variables ➞ [26 CONS]

- alldifferent_on_intersection,
- common,
- common_interval,
- common_modulo,
- common_partition,
- same,
- same_and_global_cardinality,
- same_and_global_cardinality_low_up,
- same_intersection,
- same_interval,
- same_modulo,
- same_partition,
- soft_same_interval_var,
- soft_same_modulo_var,
- soft_same_partition_var,
- soft_same_var,
- soft_used_by_interval_var,
- soft_used_by_modulo_var,
- soft_used_by_partition_var,
- soft_used_by_var,
- sort,
- uses,
- used_by,
- used_by_interval,
- used_by_modulo,
- used_by_partition.

A constraint involving only two collections of domain variables in its arguments.

3.7.55 ▼ Constraint between three collections of variables ➞ [2 CONS]

- correspondence,
- sort_permutation.

A constraint involving only three collections of domain variables in its arguments.

3.7.56 ▼ Constraint involving set variables ➞ [32 CONS]

- alldifferent_between_sets,
- atmost1,
- bipartite,
- clique,
- connected,
- dag,
- disj,
- dom_reachability,
- eq_set,
- graph_isomorphism,
- in_set,
- inverse_set,
- k_cut,
- link_set_to_booleans,
- open_alldifferent,
- open_among,
- open_atleast,
- open_atmost,
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

- open_global_cardinality,
- open_global_cardinality.low_up,
- path_from_to,
- proper_forest,
- roots,
- set_value_precede,
- strongly_connected,
- subgraph_isomorphism,
- sum_free,
- sum_set,
- symmetric,
- symmetric_cardinality,
- symmetric_gcc,
- tour.

A constraint involving set variables in its arguments.

3.7.57 ▼Constraint on the intersection ➔ [4 CONS]

- common,
- alldifferent_on_intersection,
- nvalue_on_intersection,
- same_intersection.

Denotes that a constraint involving two collections of variables imposes a restriction on the values that occur in both collections.

3.7.58 ▼Constructive disjunction ➔ [5 CONS]

- case,
- disjunctive,
- geost,
- diffn,
- two_orth_do_not_overlap.

A constraint for which a filtering algorithm uses constructive disjunction. Constructive disjunction [395, 417] is a technique for handling in an active way a set of disjunctive constraints. It consists to try out each alternative of a disjunction and then to remove values that were pruned in all alternatives. Table 3.10 illustrates this technique in the context of a non-overlapping constraint between two rectangles (i.e., a special case of the two_orth_do_not_overlap constraint). The first rectangle $R_1$ has a width of 3 and a height of 2, while the second rectangle $R_2$ has a width of 2 and a height of 5. The coordinates $(x_1, y_1)$ of the lower lefmost corner of $R_1$ have to be respectively located within intervals $[3, 5]$ and $[6, 7]$. Similarly the coordinates $(x_2, y_2)$ of the lower lefmost corner of $R_2$ have to be located within $[2, 4]$ and $[3, 4]$.

- In the context of the case constraint, constructive disjunction is applied on each sink node of the dag describing the set of solutions (i.e., we remove values that are removed in all the sink nodes).

- In the context of the disjunctive (respectively diffn) constraint, constructive disjunction can be applied on each pair of tasks (respectively objects). However, as described in the Algorithm slots of these two constraints, more specific and efficient filtering algorithms exist for both constraints.
Table 3.10: Illustrating constructive disjunction in the context of a non-overlapping constraint between two rectangles.

<table>
<thead>
<tr>
<th>Hypothesis regarding the respective position of $R_1$ and $R_2$</th>
<th>Removed values from each variable according to each hypothesis</th>
<th>Values finally removed: value 3 from $X_1$ and value 4 from $X_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_2$ before $R_1$: $X_2 + 2 \leq X_1$</td>
<td>$X_1 : {3}$</td>
<td>• In the context of the geost constraint, constructive disjunction is applied on the different potential values of the shape variable of an object in order to prune its coordinates.</td>
</tr>
<tr>
<td>$[2, 3] + 2 \leq [4, 5]$</td>
<td>$X_2 : {4}$</td>
<td></td>
</tr>
<tr>
<td>$R_2$ after $R_1$: $X_1 + 3 \leq X_2$</td>
<td>$X_1 : {3, 4, 5}$</td>
<td></td>
</tr>
<tr>
<td>$[3, 5] + 3 \leq [2, 4]$</td>
<td>$X_2 : {2, 3, 4}$</td>
<td></td>
</tr>
<tr>
<td>contradiction</td>
<td>$X_1 : {3, 4, 5}$</td>
<td></td>
</tr>
<tr>
<td>$R_2$ below $R_1$: $Y_2 + 5 \leq Y_1$</td>
<td>$Y_1 : {6, 7}$</td>
<td></td>
</tr>
<tr>
<td>$[3, 4] + 5 \leq [6, 7]$</td>
<td>$Y_2 : {3, 4}$</td>
<td></td>
</tr>
<tr>
<td>contradiction</td>
<td>$Y_1 : {6, 7}$</td>
<td></td>
</tr>
<tr>
<td>$R_2$ on top of $R_1$: $Y_1 + 2 \leq Y_2$</td>
<td>$Y_2 : {3, 4}$</td>
<td></td>
</tr>
<tr>
<td>$[6, 7] + 2 \leq [3, 4]$</td>
<td>$Y_2 : {3, 4}$</td>
<td></td>
</tr>
</tbody>
</table>

3.7.59 ▼Contact ➤ [2 CONS]

- orths_are_connected.
- two_orth_are_in_contact.

A constraint enforcing that some orthotopes touch each other. Part (A) of Figure 3.14 shows two orthotopes that are in contact while parts (B) and (C) give two examples of orthotopes that are not in contact.

![Figure 3.14: Illustration of the notion of contact](image-url)
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.60 Contractible

- all_differ_from_at_least_k_pos (contractible wrt. VECTORS),
- all_equal (contractible wrt. VARIABLES),
- all_incomparable (contractible wrt. VECTORS),
- all_min_dist (contractible wrt. VARIABLES),
- alldifferent (contractible wrt. VARIABLES),
- alldifferent_between_sets (contractible wrt. VARIABLES),
- alldifferent_cst (contractible wrt. VARIABLES),
- alldifferent_except_0 (contractible wrt. VARIABLES),
- alldifferent_interval (contractible wrt. VARIABLES),
- alldifferent_modulo (contractible wrt. VARIABLES),
- alldifferent_on_intersection (contractible wrt. VARIABLES),
- alldifferent_on_intersection (contractible wrt. VARIABLES),
- alldifferent_partition (contractible wrt. VARIABLES),
- allperm (suffix-contractible wrt. MATRIX vec),
- among (contractible wrt. VARIABLES when $NVAR = 0$),
- among (contractible wrt. VARIABLES when $NVAR = |VARIABLES|$),
- among_diff_0 (contractible wrt. VARIABLES when $NVAR = 0$),
- among_diff_0 (contractible wrt. VARIABLES when $NVAR = |VARIABLES|$),
- among_interval (contractible wrt. VARIABLES when $NVAR = 0$),
- among_interval (contractible wrt. VARIABLES when $NVAR = |VARIABLES|$),
- among_low_up (contractible wrt. VARIABLES when $UP = 0$),
- among_low_up (contractible wrt. VARIABLES when $UP = |VARIABLES|$),
- among_modulo (contractible wrt. VARIABLES when $NVAR = 0$),
- among_modulo (contractible wrt. VARIABLES when $NVAR = |VARIABLES|$),
- among_seq (contractible wrt. VARIABLES when $SEQ = 1$),
- among_seq (contractible wrt. VARIABLES when $SEQ = 1$),
- among_seq (prefix-contractible wrt. VARIABLES),
- among_seq (suffix-contractible wrt. VARIABLES),
- among_var (contractible wrt. VARIABLES when $NVAR = 0$),
- among_var (contractible wrt. VARIABLES when $NVAR = |VARIABLES|$),
- arith (contractible wrt. VARIABLES),
- arith_or (contractible wrt. $\{VARIABLES1, VARIABLES2\}$),
- arith_sliding (contractible wrt. VARIABLES when $RELOP \in \{<, \leq\}$ and $\text{minval}(VARIABLES.var) \geq 0$),
- arith_sliding (suffix-contractible wrt. VARIABLES),
- assign_and_counts (contractible wrt. ITEMS when $RELOP \in \{<, \leq\}$),
- assign_and_nvalues (contractible wrt. ITEMS when $RELOP \in \{<, \leq\}$),
- atmost (contractible wrt. VARIABLES),
- atmost1 (contractible wrt. SETS),
- atmost_nvalue (contractible wrt. VARIABLES),
- atmost_nvector (contractible wrt. VECTORS),
- bin_packing (contractible wrt. ITEMS).
CHAPTER 3. DESCRIPTION OF THE CATALOGUE

- \texttt{bin\_packing\_caps} (contractible wrt. \texttt{ITEMS}),
- \texttt{calendar} (contractible wrt. \texttt{INSTANTS}),
- \texttt{change} (contractible wrt. \texttt{VARIABLES} when \(CTR \in \{\neq, <, >, \leq\}\) and \(\text{NCHANGE} = 0\)),
- \texttt{change} (contractible wrt. \texttt{VARIABLES} when \(CTR \in \{=, <, >, \geq\}\) and \(\text{NCHANGE} = |\text{VARIABLES} - 1|\)),
- \texttt{coloured\_cumulative} (contractible wrt. \texttt{TASKS}),
- \texttt{coloured\_cumulatives} (contractible wrt. \texttt{TASKS}),
- \texttt{compare\_and\_count} (contractible wrt. \([\text{VARIABLES1}, \text{VARIABLES2}]\) when \(\text{COUNT} \in [<, \leq]\)),
- \texttt{contains\_sboxes} (suffix-contractible wrt. \texttt{OBJECTS}),
- \texttt{count} (contractible wrt. \texttt{VARIABLES} when \(\text{RELOP} \in [<, \leq]\)),
- \texttt{counts} (contractible wrt. \texttt{VARIABLES} when \(\text{RELOP} \in [<, \leq]\)),
- \texttt{covers\_sboxes} (suffix-contractible wrt. \texttt{OBJECTS}),
- \texttt{cumulative} (contractible wrt. \texttt{TASKS}),
- \texttt{cumulative\_convex} (contractible wrt. \texttt{TASKS}),
- \texttt{cumulative\_product} (contractible wrt. \texttt{TASKS}),
- \texttt{cumulative\_two\_d} (contractible wrt. \texttt{RECTANGLES}),
- \texttt{cumulative\_with\_level\_of\_priority} (contractible wrt. \texttt{TASKS}),
- \texttt{cumulatives} (contractible wrt. \texttt{TASKS} when \(\text{RELOP} \in [\leq]\) and \(\minval(\text{TASKS.height}) \geq 0\)),
- \texttt{decreasing} (contractible wrt. \texttt{VARIABLES}),
- \texttt{diffn} (contractible wrt. \texttt{ORTHOTOPES}),
- \texttt{diffn\_column} (contractible wrt. \texttt{ORTHOTOPES}),
- \texttt{diffn\_include} (contractible wrt. \texttt{ORTHOTOPES}),
- \texttt{disjoint} (contractible wrt. \texttt{VARIABLES1}),
- \texttt{disjoint} (contractible wrt. \texttt{VARIABLES2}),
- \texttt{disjoint\_sboxes} (suffix-contractible wrt. \texttt{OBJECTS}),
- \texttt{disjoint\_tasks} (contractible wrt. \texttt{TASKS1}),
- \texttt{disjoint\_tasks} (contractible wrt. \texttt{TASKS2}),
- \texttt{disjunctive} (contractible wrt. \texttt{TASKS}),
- \texttt{disjunctive\_or\_same\_end} (contractible wrt. \texttt{TASKS}),
- \texttt{disjunctive\_or\_same\_start} (contractible wrt. \texttt{TASKS}),
- \texttt{domain} (contractible wrt. \texttt{VARIABLES}),
- \texttt{equal\_sboxes} (suffix-contractible wrt. \texttt{OBJECTS}),
- \texttt{global\_cardinality} (contractible wrt. \texttt{VALUES}),
- \texttt{global\_cardinality\_low\_up} (contractible wrt. \texttt{VALUES}),
- \texttt{global\_contiguity} (contractible wrt. \texttt{VARIABLES}),
- \texttt{golomb} (contractible wrt. \texttt{VARIABLES}),
- \texttt{increasing} (contractible wrt. \texttt{VARIABLES}),
- \texttt{inside\_sboxes} (suffix-contractible wrt. \texttt{OBJECTS}),
- \texttt{int\_value\_precede} (suffix-contractible wrt. \texttt{VARIABLES}),
- \texttt{int\_value\_precede\_chain} (contractible wrt. \texttt{VALUES}),
- \texttt{int\_value\_precede\_chain} (suffix-contractible wrt. \texttt{VARIABLES}),
- \texttt{interval\_and\_count} (contractible wrt. \texttt{COLOURS}),
- \texttt{interval\_and\_count} (contractible wrt. \texttt{TASKS}),
- \texttt{interval\_and\_sum} (contractible wrt. \texttt{TASKS}),
- \texttt{k\_alldifferent} (contractible wrt. \texttt{VARS}),
- \texttt{k\_disjoint} (contractible wrt. \texttt{SETS}),
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

- \texttt{k\_same} (contractible wrt. \texttt{SETS}),
- \texttt{k\_same\_interval} (contractible wrt. \texttt{SETS}),
- \texttt{k\_same\_modulo} (contractible wrt. \texttt{SETS}),
- \texttt{k\_same\_partition} (contractible wrt. \texttt{SETS}),
- \texttt{k\_used\_by} (contractible wrt. \texttt{SETS}),
- \texttt{k\_used\_by\_interval} (contractible wrt. \texttt{SETS}),
- \texttt{k\_used\_by\_modulo} (contractible wrt. \texttt{SETS}),
- \texttt{k\_used\_by\_partition} (contractible wrt. \texttt{SETS}),
- \texttt{lex\_all\_different} (contractible wrt. \texttt{VECTORS}),
- \texttt{lex\_between} (suffix-contractible wrt. \{\texttt{LOWER\_BOUND}, \texttt{VECTOR}, \texttt{UPPER\_BOUND\_BOUND}\}),
- \texttt{lex\_chain\_less} (contractible wrt. \texttt{VECTORS}),
- \texttt{lex\_chain\_lesseq} (contractible wrt. \texttt{VECTORS}),
- \texttt{lex\_chain\_lesseq} (suffix-contractible wrt. \texttt{VECTORS.vec}),
- \texttt{lex\_equal} (contractible wrt. \{\texttt{VECTOR1}, \texttt{VECTOR2}\}),
- \texttt{lex\_greater\_eq} (suffix-contractible wrt. \{\texttt{VECTOR1}, \texttt{VECTOR2}\}),
- \texttt{lex\_less\_eq} (suffix-contractible wrt. \{\texttt{VECTOR1}, \texttt{VECTOR2}\}),
- \texttt{lex\_less\_eq\_allperm} (suffix-contractible wrt. \{\texttt{VECTOR1}, \texttt{VECTOR2}\}),
- \texttt{meet\_sboxes} (suffix-contractible wrt. \texttt{OBJECTS}),
- \texttt{multi\_inter\_distance} (contractible wrt. \texttt{VARIABLES}),
- \texttt{multi\_global\_contiguity} (contractible wrt. \texttt{VARIABLES}),
- \texttt{and} (contractible wrt. \texttt{VARIABLES} when \texttt{VAR} = 0),
- \texttt{nequivalence} (contractible wrt. \texttt{VARIABLES} when \texttt{NEQUIV} = 1 and |\texttt{VARIABLES}| > 0),
- \texttt{nequivalence} (contractible wrt. \texttt{VARIABLES} when \texttt{NEQUIV} = |\texttt{VARIABLES}|),
- \texttt{ninterval} (contractible wrt. \texttt{VARIABLES} when \texttt{NVAL} = 1 and |\texttt{VARIABLES}| > 0),
- \texttt{ninterval} (contractible wrt. \texttt{VARIABLES} when \texttt{NVAL} = |\texttt{VARIABLES}|),
- \texttt{no\_peak} (contractible wrt. \texttt{VARIABLES}),
- \texttt{no\_valley} (contractible wrt. \texttt{VARIABLES}),
- \texttt{non\_overlap\_sboxes} (suffix-contractible wrt. \texttt{OBJECTS}),
- \texttt{nor} (contractible wrt. \texttt{VARIABLES} when \texttt{VAR} = 1),
- \texttt{not\_in} (contractible wrt. \texttt{VALUES}),
- \texttt{npair} (contractible wrt. \texttt{PAIRS} when \texttt{NPAIRS} = 1 and |\texttt{PAIRS}| > 0),
- \texttt{npair} (contractible wrt. \texttt{PAIRS} when \texttt{NPPAIRS} = |\texttt{PAIRS}|),
- \texttt{nvalue} (contractible wrt. \texttt{VARIABLES} when \texttt{NVAL} = 1 and |\texttt{VARIABLES}| > 0),
- \texttt{nvalue} (contractible wrt. \texttt{VARIABLES} when \texttt{NVAL} = |\texttt{VARIABLES}|),
- \texttt{nvalue\_on\_intersection} (contractible wrt. \texttt{VARIABLES1} when \texttt{NVAL} = 0),
- \texttt{nvalue\_on\_intersection} (contractible wrt. \texttt{VARIABLES2} when \texttt{NVAL} = 0),
- \texttt{nvalues} (contractible wrt. \texttt{VARIABLES} when \texttt{RELOP} \in \llbracket <, \leq \rrbracket),
- \texttt{nvalues} (contractible wrt. \texttt{VARIABLES} when \texttt{RELOP} \in [=] and \texttt{LIMIT} = 1 and |\texttt{VARIABLES}| > 0),
- \texttt{nvalues} (contractible wrt. \texttt{VARIABLES} when \texttt{RELOP} \in [=] and \texttt{LIMIT} = |\texttt{VARIABLES}|),
- \texttt{nvalues\_except\_0} (contractible wrt. \texttt{VARIABLES} when \texttt{RELOP} \in \llbracket <, \leq \rrbracket),
- \texttt{nv\_vector} (contractible wrt. \texttt{VECTORS} when \texttt{NVEC} = 1 and |\texttt{VECTORS}| > 0),
- \texttt{nv\_vector} (contractible wrt. \texttt{VECTORS} when \texttt{NVECS} = |\texttt{VECTORS}|),
- \texttt{nv\_vectors} (contractible wrt. \texttt{VECTORS} when \texttt{RELOP} \in \llbracket <, \leq \rrbracket),
- \texttt{open\_all\_different} (suffix-contractible wrt. \texttt{VARIABLES}),
- \texttt{open\_among} (suffix-contractible wrt. \texttt{VARIABLES} when \texttt{NVAR} = 0),
- \texttt{open\_at\_most} (suffix-contractible wrt. \texttt{VARIABLES}),
CHAPTER 3. DESCRIPTION OF THE CATALOGUE

- or (contractible wrt. VARIABLES when VAR = 0),
- ordered_atmost_nvector (contractible wrt. VECTORS),
- ordered_global_cardinality (contractible wrt. VALUES),
- ordered_nvector (contractible wrt. VECTORS when NV = 1 and |VECTORS| > 0),
- ordered_nvector (contractible wrt. VECTORS when NV = |VECTORS|),
- orth_link_ori_siz_end (contractible wrt. ORTHOTYPE),
- overlap_sboxes (suffix-contractible wrt. OBJECTS),
- pattern (prefix-contractible wrt. VARIABLES),
- pattern (suffix-contractible wrt. VARIABLES),
- peak (contractible wrt. VARIABLES when N = 0),
- period (contractible wrt. VARIABLES when CTR ∈ [=} and PERIOD = 1),
- period (prefix-contractible wrt. VARIABLES),
- period (suffix-contractible wrt. VARIABLES),
- period_except_0 (contractible wrt. VARIABLES when CTR ∈ [=} and PERIOD = 1),
- period_except_0 (prefix-contractible wrt. VARIABLES),
- period_except_0 (suffix-contractible wrt. VARIABLES),
- period_vectors (prefix-contractible wrt. VARIABLES),
- period_vectors (suffix-contractible wrt. VARIABLES),
- product_ctr (contractible wrt. VARIABLES when CTR ∈ [<, ≤] and minval(VARIABLES.var) > 0),
- range_ctr (contractible wrt. VARIABLES when CTR ∈ [<, ≤]),
- same_and_global_cardinality (contractible wrt. VALUES),
- same_and_global_cardinality_low_up (contractible wrt. VALUES),
- scalar_product (contractible wrt. LINEAR TERM when CTR ∈ [<, ≤], minval(LINEAR TERM.coeff) ≥ 0 and minval(LINEAR TERM.var) ≥ 0),
- set_value_precede (suffix-contractible wrt. VARIABLES),
- sliding_distribution (contractible wrt. VARIABLES when SEQ = 1),
- sliding_distribution (prefix-contractible wrt. VARIABLES),
- sliding_distribution (suffix-contractible wrt. VARIABLES),
- sliding_distribution (contractible wrt. VALUES),
- sliding_sum (contractible wrt. VARIABLES when SEQ = 1),
- sliding_sum (prefix-contractible wrt. VARIABLES),
- sliding_sum (suffix-contractible wrt. VARIABLES),
- sliding_time_window (contractible wrt. TASKS),
- sliding_time_window_from_start (contractible wrt. TASKS),
- sliding_time_window_sum (contractible wrt. TASKS),
- smooth (contractible wrt. VARIABLES when NCHANGE = 0),
- smooth (contractible wrt. VARIABLES when NCHANGE = |VARIABLES| - 1),
- strictly_decreasing (contractible wrt. VARIABLES),
- strictly_increasing (contractible wrt. VARIABLES),
- sum_ctr (contractible wrt. VARIABLES when CTR ∈ [<, ≤] and minval(VARIABLES.var) ≥ 0),
- sum_ctr (contractible wrt. VARIABLES when CTR ∈ [≥, >] and maxval(VARIABLES.var) ≤ 0),
- sum_cubes_ctr (contractible wrt. VARIABLES when CTR ∈ [<, ≤] and minval(VARIABLES.var) ≥ 0),
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

- `sum_cubes_ctr` (contractible wrt. VARIABLES when CTR ∈ [≥, >] and \[\text{maxval}(\text{VARIABLES.var}) ≤ 0\]),
- `sum_of_increments` (prefix-contractible wrt. VARIABLES),
- `sum_of_increments` (suffix-contractible wrt. VARIABLES),
- `sum_squares_ctr` (VARIABLES) when CTR ∈ [<, ≤],
- `twin` (contractible wrt. PAIRS),
- `used_by` (VARIABLES2),
- `used_by_interval` (contractible wrt. VARIABLES2),
- `used_by_modulo` (contractible wrt. VARIABLES2),
- `used_by_partition` (contractible wrt. VARIABLES2),
- `uses` (contractible wrt. VARIABLES2),
- `valley` (contractible wrt. VARIABLES when N = 0),
- `vec_eq_tuple` (contractible wrt. [VARIABLES, TUPLE]).

A contractible constraint is a constraint for which, given any satisfied ground instance, one can remove any item from one of its collection arguments, without affecting that the resulting constraint still holds, assuming all its restrictions hold. A typical example of a contractible constraint is the alldifferent constraint: given any ground satisfied instance, e.g., alldifferent((3, 8, 1)), we can remove any value from its unique argument without affecting that the resulting constraint still holds. We generalize slightly the original definition of contractibility introduced by [253] in the following ways:

- The sequence of variables is replaced by a collection. Consequently, variables are replaced by items. For instance, in the context of the cumulative(TASKS, LIMIT) constraint, we can remove any task from TASKS from any satisfied instance without affecting that the resulting constraint still holds (e.g., if the resource limit LIMIT is not exceeded at any point in time, this still is the case if we remove any task, i.e., since task heights are restricted to be non negative).

- Since the constraint may have more than one argument, one has to explicitly specify the argument from which one may remove items.

- Items can not only be removed from the end of a collection like in [253], but also from the beginning or from any part. Allowing to remove items from the beginning is called prefix-contractibility, while permitting to remove items from the end is called suffix-contractibility. Removing items from any part is just called contractibility. As an example, consider the among_seq(LOW, UP, SEQ, VARIABLES, VALUES) constraint that enforces all sequences of SEQ consecutive variables of the collection VARIABLES to be assigned at least LOW and at most UP values from VALUES. The constraint among_seq is not contractible w.r.t. the collection VARIABLES, since removing an item in the middle of VARIABLES creates a new sequence for which the restriction with respect to LOW and UP may not hold. However, if we restrict ourselves to removing just a prefix or suffix from VARIABLES, then the corresponding among_seq constraint still holds, since no new sequence is created.
A constraint may be contractible only if certain restrictions apply to some of its arguments. This is done by explicitly providing a list of restrictions, each restriction corresponding to one of the restrictions described in Section 2.1.3. We call this conditional contractibility. Given a source and a target constraint (i.e., the target constraint corresponds to the source constraint from which we remove some items in some arguments) all arguments of the target constraint should be identical to the arguments of the source constraint, except:

- Argument corresponding to a collection from which we remove items.
- Argument \(\text{arg}\) occurring in the list of conditional restrictions with restriction of the form \(\text{arg} = f(|c|)\), where \(c\) is an argument corresponding to a collection from which we remove items and \(f\) a function.

In addition, all restrictions from the list of restrictions should apply both to the source and target constraints.

We now provide two examples of conditional contractibility with respect to the \text{among}(\text{NVAR}, \text{VARIABLES}, \text{VALUES})\) constraint, which enforces \text{NVAR} to be the number of variables of the collection \text{VARIABLES} that are assigned a value in \text{VALUES}.

- In general \text{among} is not contractible since removing an item from \text{VARIABLES} may change the value of \text{NVAR}. However, given a ground satisfied instance for which \text{NVAR} is set to 0, we can remove any item from \text{VARIABLES} without affecting that the constraint still holds. In this context, the two arguments \text{NVAR} and \text{VALUES} are left unchanged within the source and the target constraint.

  As an illustration, consider the source constraint \text{among}(0, \langle 2, 4, 2 \rangle, \langle 1, 5 \rangle)\) and the target constraint \text{among}(0, \langle 2, 2 \rangle, \langle 1, 5 \rangle). Since \text{NVAR} is set to 0 both in the source and the target constraint and since \text{VALUES} is set to the same list of values both in the source and the target constraint, we have that \text{among}(0, \langle 2, 4, 2 \rangle, \langle 1, 5 \rangle)\) implies \text{among}(0, \langle 2, 2 \rangle, \langle 1, 5 \rangle).

- Similarly, when \text{NVAR} is equal to \(|\text{VARIABLES}|\), all variables are assigned a value in \text{VALUES}. In this context, we can remove any variable from \text{VARIABLES} to get a new constraint that still holds, provided that the restriction \text{NVAR} = \(|\text{VARIABLES}|\) still holds. In this example only the argument \text{VALUES} is left unchanged between the source and the target constraint. \text{NVAR} changes since it occurs in a restriction of the form \text{NVAR} = \(|\text{VARIABLES}|\) in the list of conditional restrictions.

  As an illustration, consider the source constraint \text{among}(3, \langle 2, 4, 2 \rangle, \langle 0, 2, 4, 6, 8 \rangle)\) and the target constraint \text{among}(2, \langle 4, 2 \rangle, \langle 0, 2, 4, 6, 8 \rangle). Since \text{NVAR} is set to the number of items of the \text{VARIABLES} collection both in the source and the target constraint, and since \text{VALUES} is set to the same list of values both in the source and the target constraint, we have that \text{among}(3, \langle 2, 4, 2 \rangle, \langle 0, 2, 4, 6, 8 \rangle)\) implies \text{among}(2, \langle 4, 2 \rangle, \langle 0, 2, 4, 6, 8 \rangle).
Finally, a last extension corresponds to the fact that the sequence of variables from which we remove elements may be replaced by several collections. In this context, items are removed simultaneously from all collections from exactly the same set of positions. A set of collections is either defined by a list of collections, or by a collection and one of its attributes, which is itself a collection.

As a first example, consider the \texttt{lex\_greatesteq}(\texttt{VECTOR1}, \texttt{VECTOR2}) constraint, which given two vectors each defined by a collection of variables of the same length, enforces that \texttt{VECTOR1} is lexicographically greater than or equal to \texttt{VECTOR2}. We have that \texttt{lex\_greatesteq} is suffix-contractible with respect to \texttt{VECTOR1} and \texttt{VECTOR2}. This means that we can remove the \( k \) (\( 1 \leq k \leq \text{|VECTOR1|} \)) last items from collections \texttt{VECTOR1} and \texttt{VECTOR2}. Note that the \( k \) items should be removed from both collections simultaneously. As an illustration, consider the source constraint \texttt{lex\_greatesteq}((5, 2, 8, 9), (5, 2, 6, 2)) and the target constraint \texttt{lex\_greatesteq}((5, 2, 8), (5, 2, 6)). Since \texttt{lex\_greatesteq} is suffix-contractible with respect to the two collections \texttt{VECTOR1} and \texttt{VECTOR2}, we have that \texttt{lex\_greatesteq}((5, 2, 8, 9), (5, 2, 6, 2)) implies \texttt{lex\_greatesteq}((5, 2, 8), (5, 2, 6)).

As a second example, consider the \texttt{lex\_chain\_lesseq}(\texttt{VECTORS}) constraint, which given a collection of vectors each of them defined by a collection of variables of the same length, enforces the \( i^{th} \) vector to be lexicographically less than or equal to the \((i + 1)^{th}\) vector (\( 1 \leq i < \text{|VECTORS|} \)). We have that \texttt{lex\_chain\_lesseq} is suffix-contractible with respect to \texttt{VECTORS}.vec. This means that we can remove the \( k \) last components of each vectors of the \texttt{VECTORS} collection. As in the previous example the \( k \) items should be removed from all collections simultaneously. As an illustration, consider the source constraint \texttt{lex\_chain\_lesseq}((\text{vec} – (5, 2, 3, 9), \text{vec} – (5, 2, 6, 2), \text{vec} – (5, 2, 6, 2))) and the target constraint \texttt{lex\_chain\_lesseq}((\text{vec} – (5, 2, 3), \text{vec} – (5, 2, 6), \text{vec} – (5, 2, 6))). Since \texttt{lex\_chain\_lesseq} is suffix-contractible with respect to \texttt{VECTORS}.vec, we have that \texttt{lex\_chain\_lesseq}((\text{vec} – (5, 2, 3, 9), \text{vec} – (5, 2, 6, 2), \text{vec} – (5, 2, 6, 2))) implies \texttt{lex\_chain\_lesseq}((\text{vec} – (5, 2, 3), \text{vec} – (5, 2, 6), \text{vec} – (5, 2, 6))).

The keyword extensible introduces a dual notion, where items can be added to a collection that is passed as an argument of a satisfied global constraint without affecting the fact that the resulting constraint is satisfied. Contractibility is a more common property than extensibility.

\section*{3.7.61 Convex [2 CONS]}

- \texttt{cumulative\_convex},
- \texttt{global\_contiguity}.

A constraint involving the notion of \textit{convexity}. A subset \( S \) of the plane is called \textit{convex} if and only if for any pair of points \( p, q \) of this subset the corresponding
line-segment is contained in $\mathcal{S}$. Part (A) of Figure 3.15 gives an example of convex set, while part (B) depicts an example of non-convex set.

![A convex set and a non-convex set](image)

Figure 3.15: A convex set and a non-convex set

### 3.7.62 Convex bipartite graph

- `alldifferent`
- `alldifferent_cst`
- `nvalue`

Denotes that, for a given constraint, its filtering algorithm can take advantage of having a convex bipartite graph. A bipartite graph $G = (U, V, E)$ is called convex according to its second set of vertices $V$ if there is an ordering on $V$ such that, for any vertex $u$ of $U$, the neighbours of $u$ form an interval in the previous ordering. Some graph algorithms or some problems become simpler in the context of a convex bipartite graph.

### 3.7.63 Convex hull relaxation

- `sum`

Given a non-convex set $\mathcal{S}$, $\mathcal{R}$ is a convex outer approximation of $\mathcal{S}$ if:

- $\mathcal{R}$ is convex,
- If $s \in \mathcal{S}$, then $s \in \mathcal{R}$.

Given a non-convex set $\mathcal{S}$, $\mathcal{R}$ is the convex hull of $\mathcal{S}$ if:

- $\mathcal{R}$ is a convex outer approximation of $\mathcal{S}$,
- For every $\mathcal{T}$ where $\mathcal{T}$ is a convex outer approximation of $\mathcal{S}$, $\mathcal{R} \subseteq \mathcal{T}$.

Part (A) of Figure 3.16 depicts a non-convex set, while part (B) gives its corresponding convex hull.

Within the context of linear programming the convex hull relaxation of a non-convex set $\mathcal{S}$ corresponds to the set of linear constraints characterising the convex hull of $\mathcal{S}$. 
3.7. **KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS**

3.7.64  **Conway packing problem**  ➤  [2 CONS]

- diffn,
- geost.

Denotes that a constraint can be used for solving the Conway packing problem, which consists of placing 6 orthotopes of size $4 \times 2 \times 1$, 6 orthotopes of size $3 \times 2 \times 2$ and 5 unit cubes within a $5 \times 5 \times 5$ cube.

3.7.65  **Core**  ➤  [11 CONS]

- alldifferent,
- cumulative,
- cycle,
- diffn,
- disjunctive,
- element (see also `ele` for the usage),
- global_cardinality,
- global_cardinality_with_costs,
- minimum_weight_alldifferent,
- nvalue,
- sort.

Denotes that a global constraint is an important constraint. In fact many constraints can be seen as variations or extensions around one of the following notions:

- The notion of all different enforces a set of domain variables to be assigned distinct values. Given a set of domain variables $\{v_1, v_2, \ldots, v_n\}$, the alldifferent$(\langle v_1, v_2, \ldots, v_n \rangle)$ imposes such a condition. For instance, the ground instance alldifferent$(\langle 3, 8, 2, 1 \rangle)$ is satisfied, while alldifferent$(\langle 1, 8, 2, 1 \rangle)$ is not, since value 1 is assigned twice.

- The notion of functional dependency states that a domain variable depends directly of another domain variable. A functional dependency can either be defined in intention or in extension.

  - On the one hand, functional dependencies defined by intension are usually associated with numerical constraints such as, for instance, `abs_value`(y, x) that enforce the condition $y = |x|$. They can also be associated with global constraints that mention a characteristics that is computed from one or several collections of variables. This is for instance the
case for the \( \text{nvalue}(y, \langle x_1, x_2, \ldots, x_n \rangle) \) constraint that enforce \( y \) to be equal to the number of distinct values assigned to \( x_1, x_2, \ldots, x_n \).

- On the other hand, functional dependencies defined by extension are more general since they allow representing any kind of functional dependency. The \( \text{element}(x, t, y) \) constraint allows expressing that a variable \( y \) is determined by a variable \( x \) via a table of integers \( t \), i.e., \( y = t[x] \). For instance, the ground instance \( \text{element}(2, \langle 3, 8, 3, 1 \rangle, 8) \) is satisfied since \( 8 \) is equal to the second entry of the table \( 3, 8, 3, 1 \). Typical usages of the \( \text{element} \) constraint are for instance:

  * Representing a numerical constraint that is not available in a solver, e.g. a non-linear constraint like \( y = x^3 \) (see first item of the \text{Usage} slot of the \text{elem} constraint).
  * Expressing the link between a discrete choice and its corresponding choice (see second item of the \text{Usage} slot of the \text{elem} constraint).

Both, the \text{element} and the \text{alldifferent} constraints, are the most commonly used global constraints. Many core global constraints can be seen as an extension of the \text{alldifferent}(\langle x_1, x_2, \ldots, x_n \rangle) \) constraint along one of the two following lines:

- In the first line we replace the fact that each value should not be used more than once by some more involved \text{counting constraints} like:

  - Counting the \text{total number of effectively used distinct values} like the \text{nvalue}(y, \langle x_1, x_2, \ldots, x_n \rangle) \) constraint that enforce \( y \) to be equal to the number of distinct values assigned to \( x_1, x_2, \ldots, x_n \). When \( y \) is set to the total number of variables, i.e. \( y = n \), \text{nvalue}(n, \langle x_1, x_2, \ldots, x_n \rangle) \) and \text{alldifferent}(\langle x_1, x_2, \ldots, x_n \rangle) \) are equivalent.

  - Counting the \text{number of cycles of a permutation}, i.e. we assume that the values assigned to variables \( x_1, x_2, \ldots, x_n \) belong to interval \([1, n]\), like the \text{cycle}(y, \langle x_1, x_2, \ldots, x_n \rangle) \) constraint. When (1) \( y \) is unconstrained, i.e. its can take any value in \([1, n]\), and when (2) all variables \( x_1, x_2, \ldots, x_n \) belong to \([1, n]\), \text{cycle}(y, \langle x_1, x_2, \ldots, x_n \rangle) \) and \text{alldifferent}(\langle x_1, x_2, \ldots, x_n \rangle) \) are equivalent.

  - Counting the \text{number of occurrences of each assigned value} like the \text{global_cardinality}(\langle x_1, x_2, \ldots, x_n \rangle, \langle v_1, o_1, v_2, o_2, \ldots, v_m, o_m \rangle) \) constraint that enforce each value \( v_i \) (\( 1 \leq i \leq m \)) to be assigned to exactly \( o_i \) variables of \( x_1, x_2, \ldots, x_n \). When (1) all the occurrence variables \( o_1, o_2, \ldots, o_m \) are 0-1 variables, and when (2) all variables \( x_1, x_2, \ldots, x_n \) can only be assigned values in \( \{v_1, v_2, \ldots, v_m\} \), \text{global_cardinality}(\langle x_1, x_2, \ldots, x_n \rangle, \langle v_1, o_1, v_2, o_2, \ldots, v_m, o_m \rangle) \) and \text{alldifferent}(\langle x_1, x_2, \ldots, x_n \rangle) \) are equivalent.

- In the second line we \text{generalise the disequality between two variables} in some way like:
3.7. **KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS**

Figure 3.17: Three counting based generalisations of the `alldifferent` constraint: the `nvalue`, the `cycle` and the `global_cardinality` (i.e., `gcc`) constraints; the same example `alldifferent((3, 2, 4, 1))` is reinterpreted with respect to the three generalisations.
- disjunctive.
- cumulative.
- diffn.

3.7.66 ▼Costas arrays ➜ [1 CONS]

- alldifferent.

A constraint that allows for expressing the Costas arrays problem. A Costas array is a permutation $p_1, p_2, \ldots, p_n$ of $n$ integers $1, 2, \ldots, n$ such that $\forall \delta \in [1, n - 2], \forall i \in [1, n - \delta - 1], \forall j \in [i + 1, n - \delta] : p_i - p_{i+\delta} \neq p_j - p_{j+\delta}$. A. Vellino compares in [405] three approaches respectively using Prolog, Pascal and CHIP for solving the Costas arrays problem. In fact the weaker formulation $\forall \delta \in [1, \lfloor \frac{n-1}{2} \rfloor], \forall i \in [1, n - \delta - 1], \forall j \in [i + 1, n - \delta] : p_i - p_{i+\delta} \neq p_j - p_{j+\delta}$ was shown to be equivalent to the original one in [104].

3.7.67 ▼Cost filtering constraint ➜ [5 CONS]

- cond_lex_cost.
- global_cardinality_with_costs.
- minimum_weight_alldifferent.
- sum_of_weights_of DISTINCT_VALUES.
- weighted_partial_alldiff.

A constraint that has a set of decision variables as well as a cost variable and for which there exists a filtering algorithm that restricts the state variables from the minimum or maximum value of the cost variable.

3.7.68 ▼Cost matrix ➜ [2 CONS]

- global_cardinality_with_costs.
- minimum_weight_alldifferent.

A constraint for which a first argument corresponds to a collection of variables Vars, a second argument to a cost matrix M, and a third argument to a cost variable C. Let ValS denote the set of values that can be assigned to the variables of Vars. The cost matrix defines for each pair $v, u$ ($v \in \text{Vars}, u \in \text{ValS}$) an elementary cost, which is used for computing C when value $u$ is assigned to variable $v$. 
3.7.69 Counting constraint

- among,
- among_diff_0,
- among_interval,
- among_low_up,
- among_modulo,
- among_var,
- atleast_nvalue,
- atleast_nvector,
- atmost_nvalue,
- atmost_nvector,
- count,
- counts,
- discrepancy,
- exactly,
- global_cardinality,
- global_cardinality_low_up,
- increasing_nvalue,
- increasing_nvalue_chain,
- length_first_sequence,
- length_last_sequence,
- max_nvalue,
- min_nvalue,
- nclass,
- nequivalence,
- ninterval,
- npair,
- nvalue,
- nvalue_on_intersection,
- nvalues,
- nvalues_except_0,
- nvector,
- nvectors,
- open_among,
- open_global_cardinality,
- open_global_cardinality_low_up,
- ordered_atleast_nvector,
- ordered_atmost_nvector,
- ordered_nvector,
- roots.

A constraint restricting the number of occurrences of some values (respectively some pairs of values) within a given collection of domain variables (respectively pairs of domain variables).

3.7.70 Cumulative longest hole problems

- cumulative.

A constraint that can use some filtering based on the longest closed and open hole problems [35]. We follow the presentation from the previous paper. Before presenting the longest closed open hole scheduling problems, let us first introduce some notation related to the cumulative(TASKS, LIMIT) constraint that will be used within the context of the longest closed and open hole problems.

Here, TASKS is a collection of tasks, and for a task \( t \in \text{TASKS} \), \( t\.\text{origin} \), \( t\.\text{duration} \) and \( t\.\text{height} \) denote respectively its start, duration and height, while \( \text{LIMIT} \in \mathbb{Z}^+ \) is the height of the resource. The constraint is equivalent to finding
an assignment $s : \text{TASKS}.\text{origin} \rightarrow \mathbb{Z}^+$ that solves the cumulative placement of \text{TASKS} of maximum height $\text{LIMIT}$, i.e.:

$$\forall i \in \mathbb{Z} : \sigma_s(i) = \text{LIMIT} - P(\text{TASKS}, i) \geq 0$$

where the coverage $P(\text{TASKS}, i)$ by \text{TASKS} of instant $i \in \mathbb{Z}$ is:

$$P(\text{TASKS}, i) = \sum_{t \in \text{TASKS} \mid t.\text{origin} \leq i < t.\text{origin} + t.\text{duration}} t.\text{height}$$

We are now in position to define the longest closed and open hole problems. Given a quantity $\sigma \in \mathbb{Z}^+$ of slack (i.e. the difference between the available space and the total area of the tasks to place), the longest closed hole problem is to find the largest integer $l_{\text{cmax}}^{\text{LIMIT}}(\sigma)$ for which there exists a cumulative placement $s$ of a subset of tasks $\text{TASKS}' \subseteq \text{TASKS}$ of maximum height $\text{LIMIT}$, such that the resource area that is not occupied by $s$ on interval $[0, l_{\text{cmax}}^{\text{LIMIT}}(\sigma)]$ does not exceed the maximum allowed slack value $\sigma$:

$$\sum_{i=0}^{l_{\text{cmax}}^{\text{LIMIT}}(\sigma) - 1} \sigma_s(i) \leq \sigma.$$ 

The longest open hole problem is to find the largest integer $l_{\text{omax}}^{\text{LIMIT}}(\sigma)$ for which there exist a cumulative placement $s$ of a subset of tasks $\text{TASKS}' \subseteq \text{TASKS}$ of maximum height $\text{LIMIT}$ and an interval $[i', i' + l_{\text{omax}}^{\text{LIMIT}}(\sigma)] \subset \mathbb{Z}$ of length $l_{\text{omax}}^{\text{LIMIT}}(\sigma)$, such that the resource area that is not occupied by $s$ on $[i', i' + l_{\text{omax}}^{\text{LIMIT}}(\sigma)]$ does not exceed the maximum allowed slack value $\sigma$:

$$\sum_{i=i'}^{i'+l_{\text{omax}}^{\text{LIMIT}}(\sigma) - 1} \sigma_s(i) \leq \sigma.$$ 

As an example, consider seven tasks of respective size $11 \times 11$, $9 \times 9$, $8 \times 8$, $7 \times 7$, $6 \times 6$, $4 \times 4$, $2 \times 2$. Part (A) of Figure 3.18 provides a cumulative placement corresponding to the longest open hole problem according to $\text{LIMIT} = 11$ and $\sigma = 0$. The longest open hole $l_{\text{omax}}^{11}(\{11 \times 11, 9 \times 9, 8 \times 8, 7 \times 7, 6 \times 6, 4 \times 4, 2 \times 2\}) = 17$ since:

- The task $8 \times 8$ cannot contribute since a gap of 3 cannot be filled by the unique candidate the task $2 \times 2$.
- The task $6 \times 6$ can also not contribute since a gap of 5 cannot be completely filled by the candidates $4 \times 4$ and $2 \times 2$.

The longest close hole $l_{\text{cmax}}^{11}(\{11 \times 11, 9 \times 9, 8 \times 8, 7 \times 7, 6 \times 6, 4 \times 4, 2 \times 2\}) = 15$: it corresponds to the longest time interval on which the resource is saturated by the illustrated placement and such that one bound of the interval does not intersect any tasks.

---

10 Without loss of generality we assume the earliest start of each task to be greater than or equal to 0.
Second, consider a task of size $3 \times 2$. Part (B) of Figure 3.18 provides a cumulative placement corresponding to the longest open hole problem according to $\epsilon = 11$ and $\sigma = 20$. The longest open hole $l_{max}^{11\{(3 \times 2)\}} = 2$.

![Figure 3.18: Examples for illustrating the longest closed and open holes problems](image)

Figure 3.18: Examples for illustrating the longest closed and open holes problems

Figure 3.19 provides examples of the longest closed hole when we have 15 squares of sizes 1, 2, . . . , 15 and a zero slack. Parts (A), (B), . . . , (O) respectively give a solution achieving the longest closed hole for a gap of 1, 2, . . . , 15. For comparison, Figure 3.20 provides the same examples of the longest open hole with zero slack.
Figure 3.19: Examples of longest closed holes for various gaps
Figure 3.20: Examples of longest open holes for various gaps
3.7.71 **Cycle** ➤

- balance_cycle,
- cycle,
- symmetric_alldifferent.

A constraint that can be used for restricting the number of cycles of a permutation (i.e., cycle), or for restricting the size of the cycles of a permutation (i.e., symmetric_alldifferent), or for restricting the difference between the largest and the smallest cycle (i.e., balance_cycle).

3.7.72 **Cyclic** ➤

- circular_change,
- cyclic_change_joker,
- cyclic_change,
- stretch_circuit.

A constraint that involves a kind of cyclicity in its definition. It either uses the arc generator CIRCUIT or an arc constraint involving \( \text{mod} \).

3.7.73 **Data constraint** ➤

- elem,
- elem_from_to,
- element,
- elementn,
- element_greaterq,
- element_lessseq,
- element_matrix,
- element_product,
- element_sparse,
- elements,
- elements_alldifferent,
- elements_sparse,
- in_relation,
- ith_pos_different_from_0,
- next_element,
- next_greater_element,
- stage_element,
- sum.

In the literature also known as *ad-hoc constraints*. A constraint that allows for representing an access to an element of a data structure (e.g., a table, a matrix, a relation) or to compute a value from a given data structure.
3.7.74 **Deadlock breaking**

- cutset.

A constraint that was used within the application area of *deadlock breaking*.

3.7.75 **Decomposition**

- all_min_dist,
- all_differ_from_at_least_k_pos,
- all_incomparable,
- among_seq,
- arith,
- arith_or,
- arith_sliding,
- decreasing,
- diffn,
- diffn_column,
- diffn_include,
- disj,
- disjunctive,
- disjunctive_or_same_end,
- disjunctive_or_same_start,
- domain_constraint,
- geost,
- geost_time,
- increasing,
- k_alldifferent,
- k_disjoint,
- k_same,
- k_same_interval,
- k_same_modulo,
- k_same_partition,
- k_used_by,
- k_used_by_interval,
- k_used_by_modulo,
- k_used_by_partition,
- lex_alldifferent,
- lex_chain_less,
- lex_chain_lessq,
- link_set_to_booleans,
- orth_link_ori_siz_end,
- precedence,
- roots,
- sequence_folding,
- sliding_distribution,
- sliding_sum,
- strictly_decreasing,
- strictly_increasing,
- symmetric_cardinality,
- symmetric_gcc,
- visible.

A constraint for which the catalogue provides a description in terms of a conjunction of more elementary constraints. This is the case when the constraint is described by one or several graph constraints that all satisfy the following property: the description uses the NARC graph property and forces all arcs of the initial graph to belong to the final graph. Most of the time we have only one single graph constraint. But some constraints (e.g., `diffn`) use more than one. Note that the arc constraint can sometimes be a logical expression involving several constraints (e.g., `domain_constraint`).
3.7.76 ♦ Decomposition-based violation measure ➔ [2 CONS]

- soft args
  alldifferent
  all
  equal
  ctr
- soft all
  equal
  min
  ctr

A soft constraint associated with a constraint that can be described in terms of a conjunction of more elementary constraints for which the violation cost is the number of violated elementary constraints.

3.7.77 ♦ DFS-bottleneck ➔ [11 CONS]

- all
  different
- balance
  cycle
- balance
  path
- bipartite
- circuit
- cycle
- global
cardinality
- global
cardinality
  low
  up
- path
- same
- used
  by

A constraint for which a depth first search based procedure usually constitutes a bottleneck of its filtering algorithm. This is a pity, especially on dense graphs where most of the invocations to the filtering algorithm do not usually bring any new deductions. Motivated by this fact, randomized filtering algorithms were introduced in [214] and in [217] in the context of the global
cardinality
  low
  up
and the all
different
constraints.

3.7.78 ♦ Demand profile ➔ [3 CONS]

- cumulatives
- same
global
cardinality
- same
global
cardinality
  low
  up

A constraint that allows for representing problems where one has to allocate resources in order to cover a given demand. A profile specifies for each instant the minimum, and possibly maximum, required demand.

---

A common implementation trick relies on the fact that, quite often on dense graphs, a depth first search procedure develops one single path such that one can directly reach (i.e. with one single arc) the first node of the path from the last one (i.e., we have one single strongly connected component). In this context the trick is to stop the depth first search procedure as soon as the last node of the path is reached, in order to avoid scanning through all remaining arcs of the graph.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.79 Degree of diversity of a set of solutions ➔ [2 CONS]

- **lex_chain_less**,  
- **soft_alldifferent_ctr**.

A constraint that allows for finding a set of solutions with a certain degree of diversity. As an example, consider the problem of finding 9 diverse solutions for the 10-queens problem. For this purpose we create a 10 by 9 matrix $M$ of domain variables taking their values in interval $[0, 9]$. Each row of $M$ corresponds to a solution of the 10-queens problem. We assume that the variables of $M$ are assigned row by row, and that within a given row, they are assigned from the first to the last column. Moreover values are tried out in increasing order. We first post for each row of $M$ the 3 alldifferent constraints related to the 10-queens problem (see Figure 5.5 for an illustration of the 3 alldifferent). With a *lex_chain_less* constraint, we lexicographically order the first two variables of each row of $M$ in order to enforce that the first two variables of any pair of solutions are always distinct. We then impose a *soft_alldifferent_ctr* constraint on the variables of each column of $M$. Let $C_i$ denote the corresponding cost variable associated with the *soft_alldifferent_ctr* constraint of the $i$-th column of $M$ (i.e., the first argument of the *soft_alldifferent_ctr* constraint). We put a maximum limit (e.g., 3 in our example) on these cost variables. We also impose that the sum of these cost variables should not exceed a given maximum value (e.g., 8 in our example). Finally, in order to balance the diversity over consecutive variables we state that the sum of two consecutive cost variables should not exceed a given threshold (e.g., 2 in our example). As a result we get the following nine solutions depicted below.

- $S_1 = \langle 0, 2, 5, 7, 9, 4, 8, 1, 3, 6 \rangle$,
- $S_2 = \langle 0, 3, 5, 8, 2, 9, 7, 1, 4, 6 \rangle$,
- $S_3 = \langle 1, 3, 7, 2, 8, 5, 9, 0, 6, 4 \rangle$,
- $S_4 = \langle 2, 4, 8, 3, 9, 6, 1, 5, 7, 0 \rangle$,
- $S_5 = \langle 3, 6, 9, 1, 4, 7, 0, 2, 5, 8 \rangle$,
- $S_6 = \langle 5, 9, 2, 6, 3, 1, 8, 4, 0, 7 \rangle$,
- $S_7 = \langle 6, 8, 1, 5, 0, 2, 4, 7, 9, 3 \rangle$,
- $S_8 = \langle 8, 1, 4, 9, 7, 0, 3, 6, 2, 5 \rangle$,
- $S_9 = \langle 9, 5, 0, 4, 1, 8, 6, 3, 7, 2 \rangle$.

The costs associated with the *soft_alldifferent_ctr* constraints of columns 1, 2, …, 10 are respectively equal to 1, 1, 1, 0, 1, 0, 1, 1, 1, and 1. The different types of constraints between the previous 9 solutions are illustrated by the next figure.
CHAPTER 3. DESCRIPTION OF THE CATALOGUE

diversity repartition constraints:

\[ 1 + 1 + 0 + 1 + 0 + 1 + 1 + 1 \leq 8 \]
\[ 1 + 1 \leq 2 \]
\[ 1 + 1 \leq 2 \]
\[ 1 + 0 \leq 2 \]
\[ 0 + 1 \leq 2 \]
\[ 1 + 0 \leq 2 \]
\[ 1 + 1 \leq 2 \]
\[ 1 + 1 \leq 2 \]

diversity on last column:

\text{soft\_alldiff\_str(1,(6,6,4,0,8,7,3,5,2))}

queen constraints of last row:

\text{alldifferent\_cst((9,5+1,0+2,4+3,1+4,8+5,6+6,3+7,7+8,2+9))}
\text{alldifferent((9,5,0,4,1,8,6,3,7,2))}
\text{alldifferent\_cst((9 + 9,5 + 8,0 + 7,4 + 6,1 + 5,8 + 4,6 + 3,3 + 2,7 + 1,2))}

diversity of initial part of solutions:

\text{lex\_chain\_less((vec -(0,2),vec -(0,3),vec -(1,3),vec -(2,4),vec -(3,6),vec -(5,9),vec -(6,8),vec -(8,1),vec -(9,5)))}

Figure 3.21: Constraint network associated with the problem of finding 9 diverse solutions for the 10-queens problem
Approaches for finding diverse and similar solutions based on the Hamming distance between each pair of solutions are presented by E. Hebrard and al. in [189].
3.7.80  Derived collection  ➤  [30 CONS]

- assign_and_counts,
- correspondence,
- cumulative_two_d,
- cumulative_with_level_of_priority,
- cumulatives,
- cycle_resource,
- domain_constraint,
- element,
- element_matrix,
- element_sparse,
- elements_sparse,
- golomb,
- in,
- in_interval,
- in_relation,
- in_same_partition,
- lex_greater,
- lex_greatereq,
- lex_less,
- lex_lesseq,
- link_set_to_bools,
- minimum_greater_than,
- next_element,
- next_greater_element,
- not_in,
- sliding_time_window_from_start,
- sort_permutation,
- track,
- tree_resource,
- two_layer_edge_crossing.

A constraint that uses one or several derived collections. Derived collections were introduced in Section 2.2.2 on page 42.

3.7.81  Difference  ➤  [2 CONS]

- golomb,
- sum_of_increments.

Denotes that the definition of a constraint involves one or several differences between pairs of variables.

3.7.82  Difference between pairs of variables  ➤  [1 CONS]

- lex_alldifferent.

A constraint that allows expressing that a set of pairs of variables are different. Two pairs of variables \((X_1, Y_1)\) and \((X_2, Y_2)\) are different if and only if \(X_1 \neq X_2\) or \(Y_1 \neq Y_2\).
3.7.83 Directed acyclic graph

- cutset.

A constraint that forces the final graph to be a directed acyclic graph. A directed acyclic graph is a digraph with no path starting and ending at the same vertex.

3.7.84 Disequality

- all_different_from_at_least_k_pos,
- alldifferent,
- alldifferent_between_sets,
- alldifferent_cst,
- alldifferent_consecutive_values,
- disjoint,
- elements_alldifferent,
- golomb,
- k_alldifferent,
- k_disjoint,
- lex_different,
- neq_cst,
- not_all_equal,
- not_in,
- open_alldifferent,
- permutation,
- roots,
- size_max_starting_seq_alldifferent,
- size_max_seq_alldifferent,
- soft_alldifferent_ctr,
- soft_alldifferent_var,
- symmetric_alldifferent.

Denotes that a disequality between two domain variables, one domain variable and a fixed value, or two set variables is used within the definition of a constraint. Denotes also that the notion of disequality can be used within the informal definition of a constraint. This is for instance the case for the relaxation of the alldifferent constraint (i.e., soft_alldifferent_ctr, soft_alldifferent_var), which do not strictly enforce a disequality.
3.7.85 ▶ Disjunction ➔ [12 CONS]

- case,
- arith_or,
- clause_or,
- diffn,
- disjunctive,
- disjunctive_or_same_end,
- disjunctive_or_same_start,
- element,
- elem,
- geost,
- geost_time,
- or.

Denotes that a constraint can be used for modelling some kind of disjunction.

3.7.86 ▶ Domain channel ➔ [1 CONS]

- domain_constraint.

A constraint that allows for making the link between a domain variable \( V \) and a set of 0-1 variables \( B_1, B_2, \ldots, B_n \). It enforces a condition of the form \( V = i \iff B_i = 1 \).

3.7.87 ▶ Domain definition ➔ [6 CONS]

- arith,
- domain,
- in,
- in_interval,
- in_intervals,
- not_in.

A constraint that is used for defining the initial domain of one or several domain variables or for removing some values from the domain of one or several domain variables.

3.7.88 ▶ Dominating queens ➔ [1 CONS]

- nvalue.

A constraint that can be used for modelling the dominating queens problem. Place a number of queens on a \( n \) by \( n \) chessboard in such a way that all squares are either attacked by a queen or are occupied by a queen. A queen can attack all squares located on the same column, on the same row or on the same diagonal. Values of the minimum number of queens for \( n \) less than or equal to 120 are reported in [279]. They are in fact all either equal to \( \left\lfloor \frac{n+1}{2} \right\rfloor \) or to \( \left\lfloor \frac{n+1}{2} \right\rfloor + 1 \).
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.89  ▶ Domination ◀  

- `atleast_nvector`,  
- `atmost_nvector`,  
- `nvalue`,  
- `nvectors`,  
- `sum_of_weights_of_distinct_values`.

A constraint that can be used for expressing directly the fact that we search for a dominating set in an undirected graph. Given an undirected graph $G = (V, E)$ where $V$ is a finite set of vertices and $E$ a finite set of unordered pairs of distinct elements from $V$, a set $S$ is a dominating set if for every vertex $u \in V - S$ there exists a vertex $v \in S$ such that $u$ is adjacent to $v$. Part (A) of Figure 3.22 gives an undirected graph $G$, while part (B) depicts a dominating set $S = \{e, f, g\}$ in $G$.

![Graph and Dominating Set](image)

Figure 3.22: A graph and one of its dominating set

3.7.90  ▶ Dual model ◀  

- `inverse`,  
- `inverse_offset`,  
- `inverse_set`,  
- `inverse_within_range`.

A constraint that can be used as a channelling constraint in a problem where the roles of the variables and the values can be interchanged. This is for instance the case when we have a bijection between a set of variables and the values they can take.
3.7.91  ▶Duplicated variables ➔

- global_cardinality,
- k_alldifferent,
- lex_greater,
- lex_greatereq,
- lex_less,
- lex_leq,
- scalar_product,
- stretch_circuit.

A constraint for which the situation where the same variable can occur more than once was considered in order to derive a better filtering algorithm or to prove a complexity result for achieving arc-consistency. Also in the case of the stretch_circuit constraint, a constraint for which the reformulation duplicates some variables.

3.7.92  ▶Dynamic programming ➔

- among_seq,
- change,
- cumulative,
- stretch_circuit,
- stretch_path.

A constraint for which a filtering algorithm uses dynamic programming. Note that dynamic programming was also used by M. A. Trick within the context of linear constraints [383].

3.7.93  ▶Empty intersection ➔

- disjoint,
- k_disjoint.

A constraint that enforces an empty intersection between two sets of variables.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.94  ▼Entailment ➤

- alldifferent.
- among_low_up.
- global_cardinality_low_up.
- maximum.
- minimum.
- not_in.

Denotes that the catalogue mentions a sufficient condition for the entailment of a constraint. Consider a constraint \( C(V_1, V_2, \ldots, V_n) \) and the potential sets of values \( dom(V_1), dom(V_2), \ldots, dom(V_n) \) that can respectively be assigned to the distinct domain variables \( V_1, V_2, \ldots, V_n \). The constraint \( C(V_1, V_2, \ldots, V_n) \) is entailed if and only if \( C(V_1, V_2, \ldots, V_n) \) holds whatever values \( val_1 \in dom(V_1), val_2 \in dom(V_2), \ldots, val_n \in dom(V_n) \) will respectively be assigned variables \( V_1, V_2, \ldots, V_n \). A satisfied constraint for which all variables are already fixed is trivially entailed.

Entailment is usually not considered as very important when designing a filtering algorithm, even if it can sometimes save waking again and again a constraint that will for sure be satisfied. Failure to detect entailment can lead to a memory leak if the constraint system is supposed to reclaim memory for entailed constraints for which it is no more possible to backtrack over the point where the constraint was posted. From a modelling point of view, entailment detection is mandatory for coming up with the reified version of a constraint (see also reified automaton constraint).

3.7.95  ▼Equality ➤

- eq.set.

Denotes that the notion of equality can be used within the informal definition of a constraint.
3.7.96 ▼Equality between multisets ➔

- \texttt{k\_same},
- \texttt{same},
- \texttt{same\_and\_global\_cardinality},
- \texttt{same\_and\_global\_cardinality\_low\_up}.

A constraint that can be used for modelling an equality constraint between two multisets.

3.7.97 ▼Equivalence ➔

- \texttt{atleast\_nvalue},
- \texttt{atleast\_nvector},
- \texttt{atmost\_nvalue},
- \texttt{atmost\_nvector},
- \texttt{balance\_interval},
- \texttt{balance\_modulo},
- \texttt{balance\_partition},
- \texttt{balance},
- \texttt{increasing\_nvalue},
- \texttt{max\_nvalue},
- \texttt{min\_nvalue},
- \texttt{nclass},
- \texttt{nequivalence},
- \texttt{ninterval},
- \texttt{not\_all\_equal},
- \texttt{npair},
- \texttt{nvalue},
- \texttt{nvalues},
- \texttt{nvectors},
- \texttt{soft\_all\_different\_var}.

Denotes that a constraint is defined by a graph constraint for which the final graph is reflexive, symmetric and transitive.

3.7.98 ▼Euler knight ➔

- \texttt{alldifferent},
- \texttt{cycle}.

Denotes that a constraint can be used for modelling some parts of the Euler knight problem. The Euler knight problem consists of finding a sequence of moves on a chessboard by a knight such that each square of the board is visited exactly once.
3.7. **KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS**

### 3.7.99 ▶Excluded ▼

- **not in.**

A constraint that prevents certain values to be taken by a variable.

### 3.7.100 ▶Extensible ▼

- **all_differ_from_at_least_k_pos** (extensible wrt. VECTORS.vec),
- **and** (extensible wrt. VARIABLES when VAR = 0),
- **assign_and_counts** (extensible wrt. ITEMS when RELOP ∈ [≥, >]),
- **assign_and_nvalues** (extensible wrt. ITEMS when RELOP ∈ [≥, >]),
- **atleast** (extensible wrt. VARIABLES),
- **atleast_nvalue** (extensible wrt. VARIABLES),
- **atleast_nvector** (extensible wrt. VECTORS),
- **between_min_max** (extensible wrt. VARIABLES),
- **clause_and** (extensible wrt. POSVARS when VAR = 0),
- **clause_and** (extensible wrt. NEGVARS when VAR = 0),
- **clause_or** (extensible wrt. POSVARS when VAR = 1),
- **clause_or** (extensible wrt. NEGVARS when VAR = 1),
- **compare_and_count** (extensible wrt. [VARIABLES1, VARIABLES2] when COUNT ∈ [≥, >]),
- **count** (extensible wrt. VARIABLES when RELOP ∈ [≥, >]),
- **counts** (extensible wrt. VARIABLES when RELOP ∈ [≥, >]),
- **differ_from_at_least_k_pos** (extensible wrt. [VARIABLES1, VARIABLES2]),
- **element** (suffix-extensible wrt. TABLE),
- **element_product** (suffix-extensible wrt. TABLE),
- **elementn** (suffix-extensible wrt. TABLE),
- **in** (extensible wrt. VALUES),
- **in_intervals** (extensible wrt. INTERVALS),
- **in_relation** (extensible wrt. TUPLES_OF_VALS),
- **in_same_partition** (extensible wrt. PARTITIONS),
- **ith_pos_different_from_0** (suffix-extensible wrt. VARIABLES),
- **lex_alldifferent** (extensible wrt. VECTORS.vec),
- **lex_chain_less** (suffix-extensible wrt. VECTORS.vec),
- **lex_different** (extensible wrt. [VECTOR1, VECTOR2]),
- **lex_greater** (suffix-extensible wrt. [VECTOR1, VECTOR2]),
- **lex_less** (suffix-extensible wrt. [VECTOR1, VECTOR2]),
- **nand** (extensible wrt. VARIABLES when VAR = 1),
- **nclass** (extensible wrt. VARIABLES when NCLASS = |PARTITIONS|).
• **nequivalence** (extensible wrt. VARIABLES when \( \text{NEQUIV} = \text{M} \)),
• **nor** (extensible wrt. VARIABLES when \( \text{VAR} = 0 \)),
• **not_all_equal** (extensible wrt. VARIABLES),
• **nvalues** (extensible wrt. VARIABLES when \( \text{RELOP} \in [\geq, >] \)),
• **nvalues_except_0** (extensible wrt. VARIABLES when \( \text{RELOP} \in [\geq, >] \)),
• **npositions** (extensible wrt. VARIABLES when \( \text{RELOP} \in [\geq, >] \)),
• **nvalues_except_0** (extensible wrt. VARIABLES when \( \text{RELOP} \in [\geq, >] \)),
• **nvec** (extensible wrt. VARIABLES when \( \text{RELOP} \in [\geq, >] \)),
• **open_atleast** (suffix-extensible wrt. VARIABLES),
• **or** (extensible wrt. VARIABLES when \( \text{VAR} = 1 \)),
• **range_ctr** (extensible wrt. VARIABLES when \( \text{CTR} \in [\geq, >] \)),
• **scalar_product** (extensible wrt. \( \text{LINEARTERM} \) when \( \text{CTR} \in [\geq, >] \), \( \text{minval}(\text{LINEARTERM}.\text{coeff}) \geq 0 \) and \( \text{minval}(\text{LINEARTERM}.\text{var}) \geq 0 \)),
• **some_equal** (extensible wrt. VARIABLES),
• **stage_element** (suffix-extensible wrt. TABLE),
• **sum_ctr** (extensible wrt. VARIABLES when \( \text{CTR} \in [\geq, >] \) and \( \text{minval}(\text{VARIABLES}.\text{var}) \geq 0 \)),
• **sum_ctr** (extensible wrt. VARIABLES when \( \text{CTR} \in [\leq] \) and \( \text{maxval}(\text{VARIABLES}.\text{var}) \leq 0 \)),
• **sum_cubes_ctr** (extensible wrt. VARIABLES when \( \text{CTR} \in [\geq, >] \) and \( \text{minval}(\text{VARIABLES}.\text{var}) \geq 0 \)),
• **sum_cubes_ctr** (extensible wrt. VARIABLES when \( \text{CTR} \in [\leq] \) and \( \text{maxval}(\text{VARIABLES}.\text{var}) \leq 0 \)),
• **sumquares_ctr** (extensible wrt. VARIABLES when \( \text{CTR} \in [\geq, >] \)),
• **used_by** (extensible wrt. VARIABLES1),
• **used_by_interval** (extensible wrt. VARIABLES1),
• **used_by_modulo** (extensible wrt. VARIABLES1),
• **used_by_partition** (extensible wrt. VARIABLES1),
• **uses** (extensible wrt. VARIABLES1).

An **extensible** constraint is a constraint for which, given any satisfied ground instance (i.e., a source constraint), one can add any item without affecting that the resulting constraint (i.e., a target constraint) still holds, assuming all its restrictions holds. All the extensions of contractibility described at the corresponding keyword entry apply also for extensibility. In particular we also have the restricted notions of **prefix-extensible** and **suffix-extensible** constraints, which respectively means that items are added before the first item of a collection or after the last item. As for contractibility, extensibility may also be conditioned by a list of restrictions. Finally extensibility may involve more than one collection. In this context, items are **added simultaneously to all collections from exactly the same set of positions**. We now present different examples of extensible constraints, starting from a very simple one.

• As a first example, consider the **atleast**(\( N \), VARIABLES, VALUE) constraint, which enforces at least \( N \) variables of the VARIABLES collection to be assigned value VALUE. We have that **atleast** is **extensible** with respect to VARIABLES, since adding a variable to an already satisfied instance of **atleast** preserves the fact that the new constraint is satisfied.
As an illustration consider the source constraint `atleast(2, ⟨4, 2, 4, 5⟩, 4)` and the target constraint `atleast(2, ⟨4, 2, 4, 5, 0, 4⟩, 4)`. Since the first argument \(N\) is set to the same value, both in the source and the target constraint, and since the third `VALUE` is also set to the same value both in the source and the target constraint, we have that `atleast(2, ⟨4, 2, 4, 5⟩, 4)` implies `atleast(2, ⟨4, 2, 4, 5, 0, 4⟩, 4)`.

- As a second example, consider the `element(INDEX, TABLE, VALUE)` constraint, which enforces `VALUE` to equal the \(INDEX^{th}\) item of `TABLE`. We have that `element` is `suffix-extensible` with respect to `TABLE`, since adding new elements at the end of `TABLE` for an already satisfied instance of `element` preserves the fact that the new constraint is satisfied.

As an illustration consider the source constraint `element(3, ⟨6, 9, 2, 9⟩, 2)` and the target constraint `element(3, ⟨6, 9, 2, 9, 8, 0, 2⟩, 2)`. Since the first argument `INDEX` is set to the same value, both in the source and the target constraint, and since the third argument `VALUE` is also set to the same value both in the source and the target constraint, we have that `element(3, ⟨6, 9, 2, 9⟩, 2)` implies `element(3, ⟨6, 9, 2, 9, 8, 0, 2⟩, 2)`.

- As a third example, consider the `and(VAR, VARIABLES)` constraint, which enforces `VAR` to equal 1 if all variables of `VARIABLES` are set to 1, and 0 otherwise. We have that `and` is `extensible` with respect to `VARIABLES` when `VAR` is equal to 0. This stems from the fact that, given a satisfied instance of `and` where `VAR = 0`, adding any new variable to `VARIABLES` preserves the fact that the new constraint is satisfied. As an illustration consider the source constraint `and(0, ⟨1, 0, 1⟩)` and the target constraint `and(0, ⟨1, 0, 0, 1⟩)`. Since the first argument `VAR` is set to 0, both in the source and the target constraint, we have that `and(0, ⟨1, 0, 1⟩)` implies `and(0, ⟨1, 0, 0, 1⟩)`.

- As a fourth example, consider the `lex_greater(VECTOR1, VECTOR2)` constraint, which enforces `VECTOR1` to be lexicographically strictly greater than `VECTOR2`. We have that `lex_greater` is `suffix-extensible` with respect to `VECTOR1` and `VECTOR2`. This means that, given a satisfied instance of `lex_greater`, adding \(k\) items at the end of its first argument `VECTOR1` and adding \(k\) other items at the end of its second argument `VECTOR2` preserves the fact that the new constraint is satisfied.

As an illustration consider the source constraint `lex_greater(⟨5, 2, 7, 1⟩, ⟨5, 2, 6, 2⟩)` and the target constraint `lex_greater(⟨5, 2, 7, 1, 0⟩, ⟨5, 2, 6, 2, 9⟩)`. We have that `lex_greater(⟨5, 2, 7, 1⟩, ⟨5, 2, 6, 2⟩)` implies `lex_greater(⟨5, 2, 7, 1, 0⟩, ⟨5, 2, 6, 2, 9⟩)`.

- As a fifth example, consider the `lex_chain_less(VECTORS)` constraint, which given a collection of vectors each of which defined by a collection of variables of the same length, enforces the \(i^{th}\) vector to be lexicographically strictly less than the \((i + 1)^{th}\) vector \((1 \leq i < |VECTORS|)\). We have that `lex_chain_less`
is suffix-extensible with respect to VECTORS.vec. This means that, given a satisfied instance of \texttt{lex\_chain\_less}, adding \( k \) items at the end of all collections simultaneously preserves the fact that the new constraint is satisfied.

As an illustration consider the source constraint \texttt{lex\_chain\_less}((\texttt{vec} - (5, 2, 3, 9), \texttt{vec} - (5, 2, 6, 2), \texttt{vec} - (5, 2, 6, 3)) and the target constraint \texttt{lex\_chain\_less}((\texttt{vec} - (5, 2, 3, 9), \texttt{vec} - (5, 2, 6, 2, 8), \texttt{vec} - (5, 2, 6, 3, 7)).

Since each vector of the source constraint is a prefix of the vector located at the same position in the target constraint the source constraint implies the target constraint.

The keyword \texttt{contractible} introduces a dual notion, where items can be removed from a collection that is passed as an argument of a satisfied global constraint without affecting the fact that the resulting constraint is satisfied. Contractibility is a more common property than extensibility.

### 3.7.101 Extension

- \texttt{in\_relation}.

A constraint that is defined by explicitly providing all its solutions.

### 3.7.102 Facilities location problem

- \texttt{cycle\_or\_accessibility}, \texttt{sum\_of\_weights\_of\_distinct\_values}.

A constraint that allows for modelling a facilities location problem. In a facilities location problem one has to select a subset of locations from a given initial set so that a given set of conditions holds.

### 3.7.103 Floor planning problem

- \texttt{diffn}, \texttt{lex\_chain\_less}, \texttt{geost}.

A constraint that can be used for the floor planning problem. The floor planning problem involves various type of spaces, such as the placement space itself (i.e., the floor), the rooms to place within the placement space, and the circulation between the rooms. The placement space can be located on one single level or on several levels. Very often the placement space corresponds to one single rectangle and all rooms are rectangles with their borders parallel to the contour of the
placement space. Circulation typically corresponds to corridors or stairs that respectively allow to access from one room to another room or from one level to another level. Within the context of floor planning three main classes of constraints have been identified [260], namely dimensional, topological and implicit constraints:

- A **dimensional constraint** usually restricts the length, the width or the surface of one single space. Ratio constraints enforce aesthetic proportions between the length and the width of a single space or constrain the surfaces of two closely related spaces such as the toilets and the shower. Dimensional constraints can be expressed by reducing the domain of some variable or by stating some arithmetic constraints between two variables.

- A **topological constraint** imposes a condition between two spaces. Typical topological constraints are:
  
  - **Adjacency constraints with a minimum contact** between a room and a corridor or another room allow expressing that their must be enough place to put a door between two given spaces. In the context of staircases one has to enforce that fact that the first and last stairs are completely accessible. When a corridor is made up from two parts, one also has to enforce that the two parts are fully in contact.
  
  - **Adjacency with the contour constraints** between a room and a specified (or not) side of the contour allow expressing the orientation of a room (or just that a room must have some window).
  
  - **Relative positioning constraints** between two specified rooms allow for instance expressing the fact that a room is located to the north of another room.
  
  - **Minimum and maximum distance constraints** between two rooms allow expressing the proximity between two given rooms.

Topological constraints occur naturally in the preliminary design phase in architecture and can typically be expressed by using reified or global constraints.

- An **implicit constraint** puts a global condition that is inherent to floor planning problems between all the spaces of the floor. We typically have:

  - **Inclusion** of each room and circulation within the contour.
  
  - **Partitioning** of the placement space (i.e., no wasted space is permitted). This is usually a hard constraint which requires specific propagation in order to prevent the creation of wasted space.
  
  - **Non-overlapping** between rooms.
  
  - **Symmetry breaking constraints** between identical rooms imposes for instance a lexicographic order between their respective lower leftmost corners.

Such constraints can typically be expressed by using global constraints, such as **diffn**, **geost**, or **lex_chain_less**.
Finally, in order to allocate as much surface as possible to the rooms, one wants sometimes to minimise the total circulation area between the different rooms.

![Figure 3.23: A solution to Maculet floor planning problem which minimises the total area of the corridors](image)

In order to illustrate these constraints we now consider an example of floor planning problem taken from R. Maculet PhD thesis [251] involving 11 spaces. Constraints on the dimensions of these space are:

- The floor where to place everything has a size of 12 by 10 meters.
- The living has a surface between 33 and 42 square meters and a minimum size of 4 by 4.
- The kitchen has a surface between 9 and 15 square meters and a minimum size of 3 by 3.
- The shower has a surface between 6 and 9 square meters and a minimum size of 2 by 2.
- The toilet has a surface between 1 and 2 square meters and a minimum size of 1 by 1.
- The first and second parts of the corridor have both a surface between 1 and 12 square meters and a minimum size of 1 by 1.
- The first, second and third rooms have all a surface between 11 and 15 square meters and a minimum size of 3 by 3.
- The fourth room has a surface between 15 and 20 square meters and a minimum size of 3 by 3.

Topological constraints between spaces are:
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

- The living is located on the south-west contour. The kitchen, the first, second and third rooms are either located on the south or on the north contour. The fourth room is on the south contour.

- All spaces, except the kitchen, are adjacent to one of the corridors with at least 1 meter of full contact.

- The kitchen is adjacent to the living and to the shower.

- The toilet is adjacent to the kitchen or to the shower.

- The first and the second parts of the corridor are adjacent and fully in contact.

Finally no wasted space is permitted. Figure 3.23 presents a solution to the corresponding floor planning problem that minimises the area of the two corridors.

3.7.104 ▼ Flow ➝ [16 CONS]

- alldifferent,
- among_seq,
- global_cardinality,
- global_cardinality_low_up,
- global_cardinality_low_up_no_loop,
- global_cardinality_no_loop,
- open_alldifferent,
- open_global_cardinality

- open_global_cardinality.low.up,
- same,
- same_and_global_cardinality,
- same_and_global_cardinality.low.up,
- sliding_sum,
- symmetric_cardinality,
- symmetric_gcc,
- used_by.

A constraint for which there is a filtering algorithm based on an algorithm that finds a feasible flow in a graph. This graph is usually constructed\(^\text{12}\) from the variables of the constraint as well as from their potential values. The next sections provide standard flow models for the alldifferent, the open_alldifferent, the global_cardinality_low_up, the global_cardinality_low_up_no_loop, the used_by, the same, and the same_and_global_cardinality_low_up constraints.

Flow models for alldifferent and open_alldifferent

Figure 3.24 presents flow models for the alldifferent and the open_alldifferent constraints. Blue arcs represent feasible flows respectively corresponding to the solutions alldifferent\((\langle x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4, x_5 = 5 \rangle)\) and open_alldifferent\((\{1, 2, 3, 5\}, \langle x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 3, x_5 = 4 \rangle)\), while pink arcs correspond to arcs that cannot carry any flow if the constraint has a solution:

\(^{12}\)Sometimes it is also constructed from the reformulation of a global constraint in term of a conjunction of linear constraints. This is for instance the case for the among_seq and the sliding_sum global constraints.
Table 3.11: Domains of the variables for the \texttt{alldifferent} constraint of Figure 3.24.

<table>
<thead>
<tr>
<th>i</th>
<th>$\text{dom}(x_i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{1,2}</td>
</tr>
<tr>
<td>2</td>
<td>{1,2}</td>
</tr>
<tr>
<td>3</td>
<td>{1,2,3}</td>
</tr>
<tr>
<td>4</td>
<td>{2,3,4}</td>
</tr>
<tr>
<td>5</td>
<td>{3,4,5}</td>
</tr>
</tbody>
</table>

- Within the context of the \texttt{alldifferent} constraint the assignments $x_3 = 1$, $x_3 = 2$ and $x_4 = 2$ are forbidden since values 1 and 2 must already be assigned to $x_1$ and $x_2$. Finally the assignments $x_4 = 3$ and $x_5 = 3$ are also forbidden since values 1, 2 and 3 must be assigned to $x_1$, $x_2$ and $x_3$.

- Note that, within the context of the \texttt{open\_alldifferent} constraint, the assignment $x_4 = 3$ does not matter at all since the position of $x_4$ within $\langle x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 3, x_5 = 4 \rangle$ does not belong to the set \{1, 2, 3, 5\}. We can only prune according to those variables that for sure should be assigned distinct values. Consequently $x_3 = 1$ and $x_3 = 2$ are forbidden since values 1 and 2 must already be assigned to $x_1$ and $x_2$. Finally the assignment $x_5 = 3$ is also forbidden since values 1, 2 and 3 must be assigned to $x_1$, $x_2$ and $x_3$.

Figure 3.24: Flow models for the \texttt{alldifferent} and the \texttt{open\_alldifferent} constraints described in Tables 3.11 and 3.12.

Flow models for the \texttt{gcc\_low\_up} and the \texttt{gcc\_low\_up\_no\_loop} constraints

Figure 3.25 presents flow models for the \texttt{global\_cardinality\_low\_up} and the \texttt{global\_cardinality\_low\_up\_no\_loop} constraints. Blue arcs represent feasible flows respectively corresponding to the solutions \texttt{global\_cardinality\_low\_up}.
Table 3.12: Domains of the variables for the open_alldifferent constraint of Figure 3.24. In addition the lower bound of the first argument of the open_alldifferent constraint is equal to \{x_1, x_2, x_3\}.

<table>
<thead>
<tr>
<th>i</th>
<th>dom(x_i)</th>
<th>i</th>
<th>dom(x_i)</th>
<th>i</th>
<th>dom(x_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{1, 2}</td>
<td>2</td>
<td>{1, 2}</td>
<td>3</td>
<td>{1, 2, 3}</td>
</tr>
<tr>
<td>4</td>
<td>{2, 3}</td>
<td>5</td>
<td>{3, 4}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.25: Flow models for the global_cardinality_low_up and the global_cardinality_low_up_no_loop constraints described in Tables 3.13 and 3.14.

Table 3.13: Domains of the variables and minimum and maximum number of occurrences of each value for the global_cardinality_low_up constraint of Figure 3.25.

<table>
<thead>
<tr>
<th>i</th>
<th>dom(x_i)</th>
<th>i</th>
<th>dom(x_i)</th>
<th>i</th>
<th>[omin_i, omax_i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{1, 2}</td>
<td>5</td>
<td>{1, 2, 3}</td>
<td>1</td>
<td>{1, 2}</td>
</tr>
<tr>
<td>2</td>
<td>{1, 2}</td>
<td>6</td>
<td>{2, 3, 4, 5}</td>
<td>2</td>
<td>{1, 2}</td>
</tr>
<tr>
<td>3</td>
<td>{1, 2}</td>
<td>7</td>
<td>{3, 5}</td>
<td>3</td>
<td>{1, 1}</td>
</tr>
<tr>
<td>4</td>
<td>{1, 2}</td>
<td>4</td>
<td>{0, 2}</td>
<td>4</td>
<td>{0, 2}</td>
</tr>
</tbody>
</table>
Table 3.14: Domains of the variables and minimum and maximum number of occurrences of each value for the \texttt{global_cardinality_low_up_no_loop} constraint of Figure 3.25.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\text{dom}(x_i)$</th>
<th>$i$</th>
<th>$\text{dom}(x_i)$</th>
<th>$i$</th>
<th>$[\text{omin}_i, \text{omax}_i]$</th>
<th>$i$</th>
<th>$[\text{omin}_i, \text{omax}_i]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$[1,2]$</td>
<td>5</td>
<td>$[1,2]$</td>
<td>loop</td>
<td>$[2,2]$</td>
<td>4</td>
<td>$[1,2]$</td>
</tr>
<tr>
<td>2</td>
<td>$[1,2]$</td>
<td>6</td>
<td>${2,4,5}$</td>
<td>1</td>
<td>$[1,2]$</td>
<td>5</td>
<td>$[0,2]$</td>
</tr>
<tr>
<td>3</td>
<td>$[1,2]$</td>
<td>7</td>
<td>${3,4,5}$</td>
<td>2</td>
<td>$[2,3]$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>${1,2,3}$</td>
<td>3</td>
<td>$[1,1]$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$((x_1 = 1, x_2 = 1, x_3 = 2, x_4 = 2, x_5 = 3, x_6 = 5, x_7 = 5), (\text{val} - 1 \omin - 1 \text{omax} - 2, \text{val} - 2 \omin - 1 \text{omax} - 2, \text{val} - 3 \omin - 1 \text{omax} - 1, \text{val} - 4 \omin - 0 \text{omax} - 2))$ and \texttt{global_cardinality_low_up_no_loop} $(2, 2, (x_1 = 1, x_2 = 2, x_3 = 2, x_4 = 2, x_5 = 1, x_6 = 4, x_7 = 3), (\text{val} - 1 \omin - 1 \text{omax} - 2, \text{val} - 2 \omin - 2 \text{omax} - 3, \text{val} - 3 \omin - 1 \text{omax} - 1, \text{val} - 4 \omin - 1 \text{omax} - 2, \text{val} - 5 \omin - 0 \text{omax} - 2))$, while pink arcs correspond to arcs that cannot carry any flow if the constraint has a solution:

- Within the context of the \texttt{global_cardinality_low_up} constraint variables $x_1$, $x_2$, $x_3$ and $x_4$ take their value within $\{1, 2\}$. Since each value in $\{1, 2\}$ can be used at most 2 times, variables different from $x_1$, $x_2$, $x_3$, $x_4$ cannot be assigned a value in $\{1, 2\}$. Consequently, $x_3 \neq 1$, $x_3 \neq 2$, $x_4 \neq 1$ and $x_4 \neq 2$. Since 3 is the only remaining value for $x_3$, and since value 3 should have no more than one occurrence, $x_4 \neq 3$ and $x_5 \neq 3$ are also forbidden.

- Note that, within the context of the \texttt{global_cardinality_low_up_no_loop} we should have at least two assignments of the form $x_i = i$ ($i \in [1, 7]$). And $x_1$ and $x_2$ are the only two variables such that $i \in \text{dom}(x_i)$. Consequently $x_1 \neq 2$ and $x_2 \neq 1$. Since we should have at least $1 + 2 + 1 + 1 = 5$ assignments of the form $x_i = j$ ($i \neq j$, $j \in [1, 4]$) and since only 5 variables can take a value in $[1, 4]$, $x_6 \neq 4$ and $x_7 \neq 5$.

**Flow models for the used_by and the same constraints**

Figure 3.26 presents flow models for the \texttt{used_by} and the \texttt{same} constraints. Blue arcs represent feasible flows respectively corresponding to the solutions \texttt{used_by}($((x_1 = 2, x_2 = 4, x_3 = 6), (y_1 = 2, y_2 = 4))$) and \texttt{same($((x_1 = 2, x_2 = 4, x_3 = 5), (y_1 = 2, y_2 = 4, y_3 = 5))$, while pink arcs correspond to arcs that cannot carry any flow if the constraint has a solution. Within the context of the \texttt{same} constraint, the assignment $x_1 = 1$ is forbidden since $1 \notin \text{dom}(y_1) \cup \text{dom}(y_2) \cup \text{dom}(y_3)$. Consequently $x_1 = 2$ and, since $y_1$ is the only variable of $\{y_1, y_2, y_3\}$ that can be assigned value 2, the assignment $y_1 = 3$ is forbidden. Now since $3 \notin \text{dom}(y_1) \cup \text{dom}(y_2) \cup \text{dom}(y_3)$ the assignment $x_2 = 3$ is also forbidden. Finally $x_3 = 6$ is forbidden since $6 \notin \text{dom}(y_1) \cup \text{dom}(y_2) \cup \text{dom}(y_3)$.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

Figure 3.26: Flow models for the used_by and the same constraints described in tables 3.15 and 3.16.

Table 3.15: Domains of the variables for the used_by constraint of Figure 3.26.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\text{dom}(x_i)$</th>
<th>$i$</th>
<th>$\text{dom}(y_i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${1, 2}$</td>
<td>1</td>
<td>${2, 3}$</td>
</tr>
<tr>
<td>2</td>
<td>${3, 4}$</td>
<td>2</td>
<td>${4, 5}$</td>
</tr>
<tr>
<td>3</td>
<td>${4, 5, 6}$</td>
<td>3</td>
<td>${4, 5}$</td>
</tr>
</tbody>
</table>

Table 3.16: Domains of the variables for the same constraint of Figure 3.26.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\text{dom}(x_i)$</th>
<th>$i$</th>
<th>$\text{dom}(y_i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${1, 2}$</td>
<td>1</td>
<td>${2, 3}$</td>
</tr>
<tr>
<td>2</td>
<td>${3, 4}$</td>
<td>2</td>
<td>${4, 5}$</td>
</tr>
<tr>
<td>3</td>
<td>${4, 5, 6}$</td>
<td>3</td>
<td>${4, 5}$</td>
</tr>
</tbody>
</table>
Table 3.17: Domains of the variables and minimum and maximum number of occurrences of each value for the same_and_global_cardinality_low_up constraint of Figure 3.27.

<table>
<thead>
<tr>
<th>i</th>
<th>dom(x_i)</th>
<th>i</th>
<th>dom(y_i)</th>
<th>i</th>
<th>omin_i, omax_i</th>
<th>i</th>
<th>omin_i, omax_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{1, 2}</td>
<td>1</td>
<td>{2, 3}</td>
<td>1</td>
<td>[0, 1]</td>
<td>4</td>
<td>[2, 3]</td>
</tr>
<tr>
<td>2</td>
<td>{3, 4}</td>
<td>2</td>
<td>{4, 5}</td>
<td>2</td>
<td>[1, 2]</td>
<td>5</td>
<td>[0, 2]</td>
</tr>
<tr>
<td>3</td>
<td>{4, 5, 6}</td>
<td>3</td>
<td>{4, 5}</td>
<td>3</td>
<td>[0, 3]</td>
<td>6</td>
<td>[0, 1]</td>
</tr>
</tbody>
</table>

Flow model for the same_and_global_cardinality_low_up constraint

Figure 3.27 presents a flow model for the same_and_global_cardinality_low_up constraint. Blue arcs represent a feasible flow corresponding to the solution same_and_global_cardinality_low_up (\( x_1 = 2, x_2 = 4, x_4 = 4, y_1 = 2, y_2 = 4, y_3 = 4 \), \( \langle \text{val} - 1 \ominus 0 \\text{omax} - 1, \text{val} - 2 \ominus 1 \\text{omax} - 2, \text{val} - 3 \ominus 0 \\text{omax} - 3, \text{val} - 4 \ominus 2 \\text{omax} - 3, \text{val} - 5 \ominus 0 \\text{omax} - 2, \text{val} - 6 \ominus 0 \\text{omax} - 1 \rangle \)), while pink arcs correspond to arcs that cannot carry any flow if the constraint has a solution. The assignment \( x_1 = 1 \) is forbidden since \( 1 \not\in \text{dom}(y_1) \cup \text{dom}(y_2) \cup \text{dom}(y_3) \). Consequently \( x_1 = 2 \) and, since \( y_1 \) is the only variable of \{\( y_1, y_2, y_3 \)\} that can be assigned value 2, the assignment \( y_1 = 3 \) is forbidden. Now since \( 3 \not\in \text{dom}(y_1) \cup \text{dom}(y_2) \cup \text{dom}(y_3) \) the assignment \( x_2 = 3 \) is also forbidden. \( x_3 = 6 \) is forbidden since \( 6 \not\in \text{dom}(y_1) \cup \text{dom}(y_2) \cup \text{dom}(y_3) \). Finally \( x_3 = 5 \) and \( y_3 = 5 \) are also forbidden since value 4 must be assigned to at least two variables.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.105 • Frequency allocation problem ➤

- all_min_dist.

A constraint that was used for modelling frequency allocation problems.

3.7.106 • Functional dependency ➤

- abs_value (intension, first argument),
- alldifferent_same_value (intension, first argument),
- among (intension, first argument),
- among_diff_0 (intension, first argument),
- among_interval (intension, first argument),
- among_modulo (intension, first argument),
- among_var (intension, first argument),
- and (intension, first argument),
- balance (intension, first argument),
- balance_cycle (intension, first argument),
- balance_interval (intension, first argument),
- balance_modulo (intension, first argument),
- balance_partition (intension, first argument),
- balance_path (intension, first argument),
- balance_tree (intension, first argument),
- binary_tree (intension, first argument),
- cardinality_atleast (intension, first argument),
- cardinality_atmost (intension, first argument),
- cardinality_atmost_partition (intension, first argument),
- case (extension),
- change (intension, first argument),
- change_continuity (intension, first, second, . . . , eighth argument),
- change_pair (intension, first argument),
- change_partition (intension, first argument),
- change_vectors (intension, first argument),
- circular_change (intension, first argument),
- clique (intension, first argument),
- colored_matrix (intension, third attribute of fifth argument, third attribute of sixth argument),
- common (intension, first argument),
- common_interval (intension, first, second argument),
- common_modulo (intension, first, second argument),
• common\_partition (intension, first, second argument),
• connect\_points (intension, fourth argument),
• crossing (intension, first argument),
• cycle (intension, first argument),
• cycle\_or\_accessibility (intension, second argument),
• cyclic\_change (intension, first argument),
• cyclic\_change\_joker (intension, first argument),
• discrepancy (intension, second argument),
• distance (intension, third argument),
• distance\_between (intension, first argument),
• distance\_change (intension, first argument),
• elem (extension, second attribute of first argument),
• element (extension, third argument),
• element\_product (extension, fourth argument),
• elements (extension, second attribute of first argument),
• elements\_alldifferent (extension, second attribute of first argument),
• eq (intension, first, second argument),
• eq\_cst (intension, first, second, and third argument),
• equivalent (intension, first argument),
• exactly (intension, first argument),
• gcd (intension, third argument),
• global\_cardinality (intension, second attribute of second argument),
• global\_cardinality\_no\_loop (intension, first argument as well as second attribute of third argument),
• global\_cardinality\_with\_costs (intension, second attribute of second argument and fourth argument),
• graph\_crossing (intension, first argument),
• group (intension, first, second,. . . ,sixth argument),
• group\_skip\_isolated\_item (intension, first, second,. . . ,fourth argument),
• imply (intension, first argument),
• increasing\_nvalue (intension, first argument),
• inverse (intension, second and third attributes of first argument),
• inverse\_offset (intension, second and third attributes of third argument),
• longest\_change (intension, first argument),
• map (intension, first, second argument),
• max\_n (intension, first argument),
• max\_nvalue (intension, first argument),
• max\_size\_set\_of\_consecutive\_var (intension, first argument),
• maximum (intension, first argument),
• maximum\_modulo (intension, first argument),
• min\_n (intension, first argument),
• min\_nvalue (intension, first argument),
3.7. **KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS**

- `min_size_set_of_consecutive_var` (intension, first argument),
- `minimum` (intension, first argument),
- `minimum_except_0` (intension, first argument),
- `minimum_module` (intension, first argument),
- `minimum_weight_alldifferent` (intension, third argument),
- `nand` (intension, first argument),
- `nclass` (intension, first argument),
- `nequivalence` (intension, first argument),
- `ninterval` (intension, first argument),
- `nor` (intension, first argument),
- `npair` (intension, first argument),
- `nset_of_consecutive_values` (intension, first argument),
- `nvalue` (intension, first argument),
- `nvalue_on_intersection` (intension, first argument),
- `nvector` (intension, first argument),
- `nvisible_from_end` (intension, first argument),
- `nvisible_from_start` (intension, first argument),
- `open_among` (intension, second argument),
- `or` (intension, first argument),
- `orchard` (intension, first argument),
- `ordered_nvector` (intension, first argument),
- `orth_link_ori_size_end` (intension, first, second and third attributes of first argument),
- `path` (intension, first argument),
- `period` (intension, first argument),
- `period_except_0` (intension, first argument),
- `period_vectors` (intension, first argument),
- `power` (intension, third argument),
- `proper_forest` (intension, first argument),
- `remainder` (intension, third argument),
- `sign_of` (intension, first argument),
- `size_max_seq_alldifferent` (intension, first argument),
- `size_max_starting_seq_alldifferent` (intension, first argument),
- `smooth` (intension, first argument),
- `stage_element` (extension, second attribute of first argument),
- `sort` (intension, second argument),
- `sort_permutation` (intension, second, third argument),
- `sum` (intension, fourth argument),
- `sum_of_weights_of_distinct_values` (intension, third argument),
- `temporal_path` (intension, first argument),
- `tree` (intension, first argument),
- `tree_range` (intension, first, second argument),
- `two_layer_edge_crossing` (intension, first argument),
CHAPTER 3. DESCRIPTION OF THE CATALOGUE

- **weighted_partial_alldiff** (intension, fourth argument),
- **xor** (intension, first argument).

A constraint that allows for representing a *functional dependency* between possibly several domain variables and one single domain variable. A sequence of variables \(X_1, X_2, \ldots, X_n\) is said to *functionally determine* another variable \(Y\) if and only if each potential tuple of values of \(X_1, X_2, \ldots, X_n\) is associated with exactly one potential value of \(Y\) (i.e., \(Y\) is a function of \(X_1, X_2, \ldots, X_n\)). For each constraint we indicate whether its functional dependency is defined in *intention* or in *extension*. We also indicate which variable \(\text{var}\) is determined by the functional dependency. Within the Arg. properties slot of a constraint that mentions the functional dependency keyword, we also mention which variables determine \(\text{var}\).

Finally, the keyword **Pure functional dependency** provides the list of constraints that are only defined by one or several functional dependencies. For instance the \(\text{nvalue}(n, \langle v_1, v_2, \ldots, v_m \rangle)\) constraint is only defined in term of a functional dependency (i.e., \(n\) is equal to the number of distinct values in \(v_1, v_2, \ldots, v_m\)), while the \(\text{tree}(n, \langle \text{node}_1, \text{node}_2, \ldots, \text{node}_m \rangle)\) constraint is not only defined in term of a functional dependency since, in addition of counting trees, it also enforces no cycle in the corresponding graph.

### 3.7.107 Geometrical constraint


A constraint between geometrical objects (e.g., points, line-segments, rectangles, orthotopes) or a constraint selecting a subset of points so that a given geometrical property holds (e.g., distance).
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.108  ▶Golomb ruler ◀

- alldifferent,
- golomb.

A constraint that allows for expressing the Golomb ruler problem. A Golomb ruler is a set of integers (marks) \( a_1 < \cdots < a_k \) such that all the differences \( a_i - a_j \) \( (i > j) \) are distinct.

3.7.109  ▶Graph colouring ◀

- alldifferent,
- k_alldifferent.
- int_value_precede_chain.

A constraint that can be used for the graph colouring problem. The graph colouring problem is to colour with a restricted number of colours the vertices of a given undirected graph in such a way that adjacent vertices are coloured with distinct colours.

3.7.110  ▶Graph constraint ◀

- balance_cycle,
- balance_path,
- balance_tree,
- binary_tree,
- bipartite,
- circuit,
- circuit_cluster,
- clique,
- connected,
- cutset,
- cycle,
- cycle_card_on_path,
- cycle_or_accessibility,
- cycle_resource,
- dag,
- derangement,
- dom_reachability,
- graph_crossing,
- graph_isomorphism,
- inverse,
- inverse_offset,
- inverse_within_range,
- k_cut,
- map,
- path,
- path_from_to,
- proper_forest,
- stable_compatibility,
- strongly_connected,
- subgraph_isomorphism,
- symmetric,
- symmetric_alldifferent,
- temporal_path,
- tour,
- tree,
- tree_range,
- tree_resource.
A constraint that selects a subgraph from a given initial graph so that this subgraph satisfies a given property and/or belong to a specific graph class.

3.7.111 Graph partitioning constraint

- balance_cycle, graph_crossing,
- balance_path, map,
- balance_tree, path,
- binary_tree, symmetric_alldifferent,
- circuit, temporal_path,
- cycle, tree,
- cycle_card_on_path, tree_range,
- cycle_resource, tree_resource.

A constraint that partitions the vertices of a given initial graph and that keeps one single successor for each vertex so that each partition corresponds to a specific pattern.

3.7.112 Guillotine cut

- diffn_column, two_orth_column.

A constraint that can enforce some kind of guillotine cut. In a lot of cutting problems the stock sheet as well as the pieces to be cut are all shaped as rectangles. In a guillotine cutting pattern all cuts must go from one edge of the rectangle corresponding to the stock sheet to the opposite edge.

3.7.113 Hall interval

- alldifferent, global_cardinality.

A constraint for which some filtering algorithms take advantage of Hall intervals. Given a set of domain variables, a Hall set is a set of values $H = \{v_1, v_2, \ldots, v_h\}$ such that there are $h$ variables whose domains are contained in $H$. A Hall interval is a Hall set that consists of an interval of values (and can therefore be specified by its endpoints).
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.114 Hamiltonian ➡

- circuit,
- tour.

A constraint enforcing to cover a graph with one Hamiltonian circuit or cycle. This corresponds to finding a circuit (respectively a cycle) passing all the vertices exactly once of a given digraph (respectively undirected graph).

3.7.115 Heuristics ➡

- alldifferent,
- discrepancy,
- inverse,
- inverse.offset,
- inverse.within.range.

A constraint that was introduced for expressing a heuristic or a constraint (alldifferent) for which an algorithm that evaluate the number of solutions was proposed.

Remark: when we do not have good bounds on the cost variable of a constrained optimisation problem, skewed binary search was introduced in [357] in order to take advantage of the fact that it is usually easier to improve the current solution cost’s than to prove that a problem is not feasible.

3.7.116 Heuristics and Berge-acyclic constraint network ➡

Consider a conjunction $\mathcal{C}$ of constraints such that:

1. The constraint network $\mathcal{N}$ corresponding to the conjunction $\mathcal{C}$ is not Berge-acyclic.
2. The filtering algorithms associated with the different constraints of the conjunction $\mathcal{C}$ all achieve arc-consistency.

In this context, one can design a heuristics that fix enough variables, but not all, so that the remaining constraint network $\mathcal{N}$ becomes Berge-acyclic.\(^{13}\) This can be achieved by fixing the variables in such a way that some constraints get entailed even if they still mention some variables that are not yet fixed. Let us illustrate that idea on a matrix model where we have a $R \times K$ matrix $\mathcal{M}$ of domain variables taking a value in interval $[1, V]$. Assume that:

\(^{13}\)The point is that, as soon as the constraint network becomes Berge-acyclic no search is needed any more to check that there is a solution, provided we achieve arc-consistency on the remaining constraints. This stems from [123], which itself is a consequence of [159].
• On each row of $\mathcal{M}$ we have a constraint that can be described in term of a counter-free automaton.
• On each column of $\mathcal{M}$ we have a $\text{global.cardinality.low\_up}$ constraint that only imposes a minimum number of occurrences for each value in $[1, V]$ (i.e., the maximum number of occurrences is not constrained at all).

Note that arc-consistency can be achieved for such constraints. For this constraint pattern, an assignment strategy that systematically tries creating a Berge-acyclic constraint network can be achieved as follows. Fix some variables so that $K - 1$ column constraints (i.e., $\text{global.cardinality.low\_up}$ constraints) get entailed. If this is the case the remaining constraint network consists of $R$ rows constraints and of one single column constraint.

As illustrated by Figure 3.28, this typically corresponds to a Berge-acyclic constraint network. Let us now finally explain how to assign values to a subset of variables of a $\text{global.cardinality.low\_up}$ constraint that only restricts the minimum number of occurrences of certain values so that it becomes entailed. As an example, let us consider a $\text{global.cardinality.low\_up}$ constraint involving 10 variables that enforce at least three occurrences of value 1 and one occurrence of value 2. A heuristics needs only fixing 4 variables out of the 10 variables to values 1, 1, 1 and 2 so that the corresponding $\text{global.cardinality.low\_up}$ gets entailed. A typical instance of this pattern corresponds to nurse scheduling problems where:
• Each row of $\mathcal{M}$ corresponds to the timetable of a person over $K$ consecutive
days. Using a counter free automaton the corresponding row constraint encodes
all legal rules of a valid schedule.

• Each column of $\mathcal{M}$ describes the request for a minimum number of services on
a given day. Types of work (i.e., values in $[1, V]$) can for instance be interpreted
as a morning shift, an afternoon shift, a night shift or a day off.

The heuristics first addresses the coverage constraints only (i.e., the
global_cardinality_low_up constraints). It seeks to assign enough nurses to
given shifts on given day to satisfy all but one coverage constraints. Once this is done,
the remaining variables can be labelled without search.

### 3.7.117 Heuristics and lexicographical ordering

- **lex_chain_less**,  
- **lex_chain_lessseq**,  
- **lex_greater**,  
- **lex_greatereq**,  
- **lex_less**,  
- **lex_lessseq**.

Using a constraint that imposes a lexicographical ordering between vectors of vari-
ables may influence the heuristics used for fixing the variables. In particular it may be
a very bad idea to systematically fix the less significant components before the most
significant components.

### 3.7.118 Heuristics for two-dimensional rectangle placement prob-
lems

- **diffn**,  
- **geost**.

A constraint for which one of the following heuristics was used in the context of
two-dimensional rectangles placement problems where rectangles should not overlap.
For easy instances involving non-overlapping constraints where there is enough room,
a standard heuristics where one fixes each rectangle successively by trying out its pos-
sible values for its $x$-coordinate and its $y$-coordinate will do the job. However, for
more difficult problems a less aggressive heuristics is usually required, specially when
the filtering algorithms attached to the constraints are weak. The paradox is that less
aggressive heuristics sometimes do not find rapidly a first solution to easy instances
since they may potentially artificially create infeasible subproblems.
CHAPTER 3. DESCRIPTION OF THE CATALOGUE

Dual strategy for rectangle placement problems with no slack

When the available space is equal to the total area of the rectangles to place (i.e., we have no slack) this is a two-phase search procedure originally introduced in [1] where we first fix all the \(x\)-coordinates and then, in the second phase, all the \(y\)-coordinates. The intuitions behind this heuristics are:

- To systematically fill the placement space from right to left in order to avoid creating small holes that cannot be filled.

- To decrease the combinatorial aspect of the problem by focusing first on all \(x\)-coordinates. This stems from the fact that it is usually easy to extend a partial solution, where all \(x\)-coordinates are fixed, to a full solution.

Fixing the \(x\)-coordinates is done by:

- First, compute the minimum \(\min_x\) over the minimum values of the \(x\)-coordinates of the rectangles for which the \(x\)-coordinate is not already fixed.

- Second, create a choice point and, in each branch:
  - Fix the \(x\)-coordinate of a rectangle \(R\) for which the \(x\)-coordinates is not already fixed to value \(\min_x\). Usually rectangles are considered by decreasing height (and decreasing width in case of tie).
  - On backtracking, enforce that the \(x\)-coordinate of rectangle \(R\) is strictly greater than \(\min_x\).

- Third, fail when all branches issued from a choice point have been tried (since otherwise we would create a hole at position \(\min_x\) because, on the \(x\) axis all rectangles that could start at position \(\min_x\) were delayed after \(\min_x\); in order to not cut valid choices, this third part assumes that the minimum value of the \(x\)-coordinate of each rectangle is pruned with respect to the compulsory part profile of the corresponding cumulative constraint.).

Since, as we said early on, it is usually easy to extend a partial solution, where all \(x\)-coordinates are fixed, to a full solution where all \(y\)-coordinates are also fixed, the search strategy used for fixing the \(y\)-coordinates is usually not so important, at least when strong filtering algorithms are used [35].

Strategy that gradually creates a compulsory part

This is a four-phase search procedure that can be used even when the slack is not equal to zero. We first gradually restrict all the \(x\)-coordinates and then, in the second phase, all \(y\)-coordinates without fixing them immediately. Then in the third phase we fix all the \(x\)-coordinates by trying each value (or by making a dichotomic search). Finally in the last phase we fix all the \(y\)-coordinates as in the third phase. The intuitions behind this heuristics are:
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

- To restrict the $x$-coordinate of each rectangle $R$ in order to just create some compulsory part for $R$ on the $x$ axis. The hope is that it will trigger the filtering algorithm associated with the cumulative constraint involved by the non-overlapping constraint, even if the starts of the rectangles on the $x$ axis are not yet completely fixed.

- Again, as in the previous heuristics, to decrease the combinatorial aspect of the problem by first focusing on all $x$-coordinates.

Restricting gradually the $x$-coordinates in phase one is done by partitioning the domain of the $x$-coordinate of each rectangle $R$ into intervals whose sizes induce a compulsory part on the $x$ axis for rectangle $R$. To achieve this, the size of an interval has to be less than or equal to the size of rectangle $R$ on the $x$ axis. Picking the best fraction of the size of a rectangle on the $x$ axis depends on the problem as well as on the filtering algorithms behind the scene. Within the context of the smallest rectangle area problem [368] and of the SICStus implementation of disjoint2 and cumulative H. Simonis and B. O’Sullivan have shown empirically that the best fraction was located within interval $[0.2, 0.3]$. Restricting the $y$-coordinates in phase two can be done in a way similar to restricting the $x$-coordinates in phase one.

3.7.119 Hungarian method for the assignment problem ➤ [1 CONS]

- minimum_weight_alldifferent.

A constraint that can use the Hungarian method for the assignment problem [225] in order to evaluate the minimum or maximum value of one of its argument. Given $n$ persons, $n$ tasks and a corresponding $n$ by $n$ cost matrix, the assignment problem is the search for an assignment of persons to tasks so that the sum of the costs is maximised.

3.7.120 Hybrid-consistency ➤

- proper_forest, roots.

Denotes that, for a given constraint involving both domain and set variables, there is a filtering algorithm that ensures hybrid-consistency. A constraint $ctr$ defined on the distinct domain variables $V^d_1, \ldots, V^d_n$ and the distinct set variables $V^s_{n+1}, \ldots, V^s_m$ is hybrid-consistent if and only if:

- For every pair $(V^d, v)$ such that $V^d$ is a domain variable of $ctr$ and $v \in dom(V^d)$, there exists at least one solution to $ctr$ in which $V^d$ is assigned the value $v$. 
For every pair \((V^s, v)\) such that \(V^s\) is a set variable of \(ctr\), if \(v \in V^s\) then \(v\) belongs to the set assigned to \(V^s\) in all solutions to \(ctr\) and if \(v \in V^s \setminus V^s\) then \(v\) belongs to the set assigned to \(V^s\) in at least one solution and is excluded from this set in at least one solution.

### 3.7.121 Hypergraph

- among_seq,
- arith_sliding,
- orchard,
- relaxed_sliding_sum,
- size_max_seq_alldifferent,
- size_max_starting_seq_alldifferent,
- sliding_distribution,
- sliding_sum.

Denotes that a constraint uses in its definition at least one arc constraint involving more than two vertices.

### 3.7.122 Included

- in,
- in_set.

Enforces that a domain or a set variable take a value within a list of values (possibly one single value).

### 3.7.123 Inclusion

- k_used_by,
- k_used_by_interval,
- k_used_by_modulo,
- used_by,
- used_by_interval,
- used_by_modulo,
- used_by_partition,
- uses.

Denotes that a constraint can model the inclusion of one multiset within another multiset. Usually we consider multiset of values (e.g., \(used_by\)) but this can also be multisets of equivalence classes (see, e.g., the \(used_by_interval, used_by_modulo,\) and \(used_by_partition\) constraints).
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.124  ▶Incompatible pairs of values  ➞  [1 CONS]

- alldifferent_partition.

A constraint that is related to the fact that some pairs of values are incompatible (i.e., the two values of each pair of values cannot simultaneously be part of a solution).

3.7.125  ▶Indistinguishable values  ➞  [3 CONS]

- int_value_precede.
- int_value_precede_chain.
- set_value_precede.

A constraint that can be used for breaking symmetries of indistinguishable values [239]. Indistinguishable values in a solution of a problem can be swapped to construct another solution of the same problem.

3.7.126  ▶Interval  ➞  [16 CONS]

- alldifferent_interval.
- among_interval.
- balance_interval.
- common_interval.
- domain.
- in_interval.
- in_intervals.
- interval_and_count.
- interval_and_sum.
- k_same_interval.
- k_used_by_interval.
- ninterval.
- same_interval.
- soft_same_interval_var.
- soft_used_by_interval_var.
- used_by_interval.

Denotes that a constraint puts a restriction related to a set of fixed intervals (or on one fixed interval).
3.7.127 ▼ Joker value ➞ [10 CONS]

- alldifferent_except_0,
- among_diff_0,
- connect_points,
- cyclic_change_joker,
- ith_pos_different_from_0,
- minimum_except_0,
- nvalues_except_0,
- period_except_0,
- symmetric_alldifferent_except_0,
- weighted_partial_alldiff.

Denotes that, for some variables of a given constraint, there exists specific values that have a special meaning: for instance they can be assigned without breaking the constraint. As an example consider the alldifferent_except_0 constraint, which forces a set of variables to take distinct values, except those variables that are assigned to 0.

3.7.128 ▼ Klee’s measure problem ➞ [1 CONS]

- diffn.

Denotes that, checking the feasibility of a ground instance of a constraint, is related to the Klee’s measure problem: given a collection of axis-aligned multi-dimensional boxes, how quickly can one compute the volume of their union.

3.7.129 ▼ Labelling by increasing cost ➞ [2 CONS]

- elem,
- element.

Some optimization problems involve minimizing a cost \( c \) consisting of a sum of elementary costs \( c_1, c_2, \ldots, c_n \), where each elementary cost \( c_i \) \((1 \leq i \leq n)\) is directly linked to the value assigned to a decision variable \( v_i \). Without loss of generality we assume that each decision variable will be assigned a value between 1 and \( m \). The link between a decision variable \( v_i \) and its corresponding cost \( c_i \) is usually expressed by a constraint of the form \( \text{element}(v_i, (c_{i,1}, c_{i,2}, \ldots, c_{i,m}), c_i) \) stating that \( c_i = j \Rightarrow c_i = c_{i,j} \). During search, while enumerating on the different values of a decision variable \( v_i \), we would like to try out values of \( v_i \) so that the corresponding cost \( c_i \) increases. This means we want to use a permutation \( \sigma_1, \sigma_2, \ldots, \sigma_m \) of \( 1, 2, \ldots, m \) such that \( c_i,\sigma_1 \leq c_i,\sigma_2 \leq \cdots \leq c_i,\sigma_m \). Note that such permutation can be obtained by sorting the costs \( c_{i,1}, c_{i,2}, \ldots, c_{i,m} \) by increasing order and by collecting the position \( \sigma_j \) where item \( c_{i,j} \) is located in the sorted list. Assuming that we perform arc-consistency on the element, we now describe three different ways to obtain the effect we want to achieve:
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

- A first direct way is to use a built-in facility that, given variable $v_i$ and the corresponding list of values $\sigma_1, \sigma_2, \ldots, \sigma_m$ introduced before, creates a choice point and tries to successively assign values $\sigma_1, \sigma_2, \ldots, \sigma_m$ to $v_i$. Note that, once $v_i$ is fixed there is no need to enumerate on the corresponding elementary cost variable $c_i$ since, by propagation, $\text{element}(v_i, (c_{i,1}, c_{i,2}, \ldots, c_{i,m}), c_i)$ will fix $c_i$. Consequently the cost variables do not need to be passed to the search procedure.

- A second indirect way, used when we want to only rely on a standard built-in that creates a choice point and tries to assign values to a variable in increasing value order, is to introduce an extra variable $u_i$. The idea is to link variable $u_i$ to variable $v_i$ in such a way that, when we try to assign values in increasing value order to variable $u_i$, both variables $v_i$ and $c_i$ get fixed and, in addition, values of $c_i$ are increasing. This can be modelled by introducing the following two $\text{element}$ constraints:

1. $\text{element}(u_i, (\sigma_1, \sigma_2, \ldots, \sigma_m), v_i)$
2. $\text{element}(v_i, (c_{i,1}, c_{i,2}, \ldots, c_{i,m}), c_i)$

The effect of a dedicated built-in that tries to assign values to a variable according to an explicit list of values is achieved by introducing the first $\text{element}$ constraint. Again, once $u_i$ is fixed the first $\text{element}$ constraint will fix variable $v_i$. Then the second $\text{element}$ constraint will also fix variable $c_i$. Consequently, both the cost and the decision variables do not need to be passed to the search procedure, i.e., we just need to pass the newly introduced variables $u_i$.

- Finally, we can first label on the cost variable $c_i$ in increasing value order. If the costs $c_{i,1}, c_{i,2}, \ldots, c_{i,m}$ are all distinct then the $\text{element}(v_i, (c_{i,1}, c_{i,2}, \ldots, c_{i,m}), c_i)$ constraint will fix $v_i$ by propagation since we assume $\text{element}$ to perform arc-consistency. Otherwise, when the costs $c_{i,1}, c_{i,2}, \ldots, c_{i,m}$ are not all distinct, we also need to label the decision variable $v_i$.

Figure 3.29 illustrates the three ways of labelling previously introduced. The primitive member($\text{var, list\_values}$) creates a choice point and tries to successively assign variable $\text{var}$ an integer value from the list $\text{list\_values}$ with respect to their ordering. The primitive indomain($\text{var}$) also creates a choice point and tries to successively assign variable $\text{var}$ an integer value of its domain, by increasing value order.
Figure 3.29: Given a decision variable $v$ and a corresponding cost variable $c$ linked by the $\text{element}(v, \langle 5, 6, 2, 9, 9 \rangle, c)$ constraint, illustration of three ways for labelling by increasing cost: Part (I) labels directly on the decision variable $v$ using an appropriate order so that successive values of $c$ are increasing; Part (II) introduces a variable $u$ linked to $v$ by the $\text{element}(u, \langle 3, 1, 2, 4, 5 \rangle, v)$ constraint and labels on $u$ by increasing value order; Part (III) labels first on the cost variable $c$ by increasing value order, and then on variable $v$. 
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.130 ▼Latin square ▶

- k_alldifferent.

![Latin square completion problem](image)

Figure 3.30: A partially filled Latin square and a possible completion

A constraint that can be used for modelling the Latin square completion problem. A Latin square of order $n$ is an $n \times n$ array in which $n$ distinct numbers in $[1, n]$ are arranged so that each number occurs once in each row and column. The Latin square completion problem is to complete a partially filled Latin square. Part (A) of Figure 3.30 gives a partially filled Latin square, while part (B) provides a possible completion. The Latin square completion problem is a pattern that occurs in some applications such as *dynamic wavelength routing* or *sport timetabling*.

3.7.131 ▼Lexicographic order ▶

- allperm.
- cond_lex_cost.
- cond_lex_greater.
- cond_lex_greatereq.
- cond_lex_less.
- cond_lex_lessseq.
- lex2.
- lex_between.
- lex_chain_less.
- lex_chain_lessseq.
- lex_greater.
- lex_greatereq.
- lex_less.
- lex_lesseq.
- lex_lessseq_allperm.
- strict_lex2.

A constraint involving a lexicographic ordering relation in its definition.
3.7.132 **Limited discrepancy search**

A constraint for simulating limited discrepancy search [178]. *Limited discrepancy search* is useful for problems for which there is a successor ordering heuristics that usually leads directly to a solution. It consists of systematically searching all paths that differ from the heuristic path in at most a very small number of discrepancies. Figure 3.31 illustrates the successive search steps (B), (C), (D), (E) and (F) on the search tree depicted by part (A). We successively explore the subtree of (A) corresponding to a discrepancy of 0, 1, 2, 3 and 4. The number on each leave indicates the total number of discrepancies to reach a leave.

3.7.133 **Linear programming**

A constraint for which a reference provides a linear relaxation (see, e.g., the *alldifferent*, the *circuit*, the *cumulative*, the *sum*, and the regular [118] constraints) or a constraint for which the flow model was derived by reformulating the constraint as a linear program (see, e.g., the *among_seq* and the *sliding_sum* constraints), or a constraint that was also proposed within the context of linear programming (see, e.g., the *circuit* and *domain constraint* constraints). In the context of linear programming the book of John N. Hooker [198] provides a significant set of relaxations for a number of global constraints.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

Figure 3.31: Illustration of limited discrepancy search
3.7.134 **Line-segments intersection →**

- crossing.
- graph_crossing.
- two_layer_edge_crossing.

A constraint on the number of line-segment intersections.

3.7.135 **Logic →**

- contains_sboxes.
- coveredby_sboxes.
- covers_sboxes.
- disjoint_sboxes.
- equal_sboxes.
- geost.
- geost_time.
- inside_sboxes.
- meet_sboxes.
- non_overlap_sboxes.
- orth_on_top_of_orth.
- overlap_sboxes.
- place_in_pyramid.
- two_orth_are_in_contact.
- two_orth_column.
- two_orth_do_not_overlap.
- two_orth_include.

A constraint which can be defined with first order logic formula encoded in the dedicated language introduced in [93].

3.7.136 **Logigraphe →**

- consecutive_groups_of_ones.

A constraint which can be used for modelling the logigraphe problem. The logigraphe problem, see Figure 3.32 for an instance taken from [297, page 36], consists of colouring a board of squares in black or white, so that each row and each column contains a specific number of sequences of black squares of given size. A sequence of integers \( s_1, s_2, \ldots, s_m \) \( (p \geq 1) \) enforces:

- a first block of \( s_1 \) consecutive black squares,
- a second block of \( s_2 \) consecutive black squares,
- ................................................
- a last block of \( s_p \) consecutive black squares.
Each block of consecutive black squares must be separated by at least one white square. Finally, white squares may eventually precede (respectively follow) the first (respectively the last) block of black squares. The logigraphe problem is NP-complete [386].

Figure 3.32: Part (A): an instance of a logigraphe and the initial deductions achieved after posting the constraints, Part (B): the corresponding unique solution.

Part (A) of Figure 3.32 shows an instance of a logigraphe and the corresponding initial deductions achieved after posting the `consecutive_groups_of.ones` constraints associated with each row and each column. We assume that each constraint achieves arc-consistency, which is actually the case when the `consecutive_groups_of.ones` constraint is represented as a counter free automaton. A white or black square indicates an initial deduction (i.e., setting a variable to 0 or to 1). Part (B) of Figure 3.32 provides the unique solution found after developing three choices, assuming that variables are assigned from the uppermost to the lowermost row. Within a given row, variables are assigned from the leftmost to the rightmost column. Value 0 is tried first before value 1. Seven additional choices are required for proving that this solution is unique. Figure 3.33 displays the corresponding search tree. Within this figure, a variable $V_{i,j}$ $(1 \leq i, j \leq 10)$ denotes the 0-1 variable associated with the $i^{th}$ row and the $j^{th}$ column of the board.

---

14 Each time we try to assign a value to a not yet fixed variable, the number of choices is incremented by 1 just before making the assignment.
Figure 3.33: Search tree developed for the logigraphe instance of Figure 3.32 (variables that are fixed by propagation were removed from the search tree)

3.7.137 Magic hexagon ➤

- alldifferent,
- global_cardinality_with_costs.

A constraint that can be used for modelling some parts of the magic hexagon problem. The magic hexagon problem, see Figure 3.34 for an example, consists of finding an arrangement of $n$ hexagons, where an integer from 1 to $n$ is assigned to each hexagon so that (1) each integer from 1 to $n$ occurs exactly once, (2) the sum of the numbers along any straight line is the same.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.138 Magic series ➔

- global_cardinality.

A constraint that allows for modelling the magic series problem with one single constraint. A non-empty finite series \( S = (s_0, s_1, \ldots, s_n) \) is magic if and only if there are \( s_i \) occurrences of \( i \) in \( S \) for each integer \( i \) ranging from 0 to \( n \). \( 3, 2, 1, 1, 0, 0, 0 \) is an example of such a magic series for \( n = 6 \).

3.7.139 Magic square ➔

- alldifferent.
- global_cardinality_with_costs.

A constraint that can be used for modelling some parts of the magic square problem. The magic square problem consists in filling an \( n \) by \( n \) square with \( n^2 \) distinct integers so that the sum of each row and column and of both main diagonals be the same.

3.7.140 Matching ➔

- symmetric_alldifferent.

A constraint that allows for expressing that we want to find a perfect matching on a graph with an even number of vertices. A perfect matching on a graph \( G \) with \( n \) vertices is a set of \( n/2 \) edges of \( G \) such that no two edges have a vertex in common.
3.7.141  ▪ Matrix ➔  [5 CONS]

- allperm,
- colored_matrix,
- element_matrix,
- lex2,
- strict_lex2.

A constraint on a matrix of domain variables (see, e.g., the allperm, colored_matrix, lex2, and strict_lex2 constraints) or a constraint that allows for representing the access to an element of a matrix (see, e.g., the element_matrix constraint).

3.7.142  ▪ Matrix model ➔  [4 CONS]

- allperm,
- colored_matrix,
- lex2,
- strict_lex2.

A constraint on a matrix of domain variables. A matrix model is a model involving one matrix of domain variables.

3.7.143  ▪ Matrix symmetry ➔  [11 CONS]

- allperm,
- increasing_global_cardinality,
- lex2,
- lex.chain.less,
- lex.chain.lesseq,
- lex.greater,
- lex.greatareq.
- lex.less,
- lex.lesseq.
- lex.lesseq.allperm,
- strict_lex2.

A constraint that can be used for breaking certain types of symmetries within a matrix of domain variables.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.144  ▶Maximum ➜

[6 CONS]

- max.index,
- max.n,
- max.nvalue,
- max.size.set.of.consecutive.var,
- maximum,
- maximum_modulo.

A constraint for which the definition involves the notion of maximum.

3.7.145  ▶Maximum clique ➜

[4 CONS]

- all.min.dist,
- alldifferent,
- clique,
- disjunctive.

A constraint (i.e., clique) that can be used for searching for a maximum clique in a graph, or a constraint (i.e., all.min.dist, alldifferent, disjunctive) that can be stated by extracting a large clique [83] from a specific graph of elementary constraints.

A maximum clique is a clique of maximum size, a clique being a subset of vertices such that each vertex is connected to all other vertices of the clique.

3.7.146  ▶Maximum number of occurrences ➜

[1 CONS]

- max.nvalue.

A constraint that restricts the maximum number of times that a given value is taken.

3.7.147  ▶maxint ➜

[4 CONS]

- deepest.valley,
- min_n,
- minimum,
- minimum_modulo.

A constraint that uses maxint in its definition in terms of graph properties or in terms of automata. maxint is the largest integer that can be represented on a machine.
A constraint that can be used for modelling the metro problem, i.e., finding the shortest distance from a given metro station to all other stations of the network.

Given an undirected graph $G = (V, E)$, with a non-negative distance attached to each edge of $E$, a conjunction of `leq_cat` constraints was used by H. Simonis in order to illustrate how propagation for such a conjunction simulates a naïve version of Dijkstra algorithm for computing the shortest distance from a given vertex $v_s$ of $V$ to all other vertices. The potential source of inefficiency comes from the fact that, depending on the scheduling policy of the underlying constraint engine, an inequality constraint can be reconsidered several times before reaching the fixed point. The problem was modelled in the following way:

- To each vertex $v_i \in V$ we associate a distance variable $D_i$, which represents the domain range of the distance between vertex $v_i$ and vertex $v_s$.
- To each edge $(v_i, v_j) \in E$ we impose two inequality constraints $D_i \leq D_j + d_{i,j}$ and $D_j \leq D_i + d_{i,j}$, where $d_{i,j}$ corresponds to the distance attached to edge $(v_i, v_j)$. This restricts the maximum difference between the distances variables associated with the two extremities of edge $(v_i, v_j)$.
- Finally, we set the distance variable attached to vertex $v_s$ to 0. Propagating the inequalities constraints by using arc-consistency enforces the maximum value of each distance variable $D_i$ to be equal to the shortest distance from vertex $v_i$ to $v_s$ when the fixed point is reached.

Figure 3.35 illustrates this problem on a metro map composed of four lines and 18 stations respectively labelled by $a$, $b$, ..., $r$. Its assumes that the distance associated with each connection is equal to 1. The figure displays the status (i.e., the minimum and maximum values) of the distance variables under the assumption that we want to compute the shortest path from station $i$. The inequalities constraints between the distance variables $D_a, D_b, \ldots, D_r$ corresponding to this metro map are:

- (constraints attached to the connections of the blue metro line)
  - $D_a \leq D_b + 1$, $D_b \leq D_a + 1$,
  - $D_b \leq D_c + 1$, $D_c \leq D_b + 1$,
  - $D_c \leq D_d + 1$, $D_d \leq D_c + 1$,
  - $D_d \leq D_e + 1$, $D_e \leq D_d + 1$,
  - $D_e \leq D_f + 1$, $D_f \leq D_e + 1$,
  - $D_f \leq D_a + 1$, $D_a \leq D_f + 1$. 

• `leq_cat`.
Figure 3.35: A metro map composed of four lines (a blue, a pink, a green and a yellow line) and the corresponding minimum and maximum values of the distance variables attached to each station, under the assumptions (1) that the distance attached to each connection is equal to 1 and (2) that we compute the shortest path from station $i$ (in red); the font size used for displaying the bounds of a distance variable is inversely proportional to the length of the shortest path to station $i$. 
• (constraints attached to the connections of the pink metro line)
  - \(D_g \leq D_f + 1, \ D_f \leq D_g + 1,\)
  - \(D_f \leq D_h + 1, \ D_h \leq D_f + 1,\)
  - \(D_h \leq D_c + 1, \ D_c \leq D_h + 1,\)
  - \(D_c \leq D_i + 1, \ D_i \leq D_c + 1,\)
  - \(D_i \leq D_j + 1, \ D_j \leq D_i + 1.\)

• (constraints attached to the connections of the green metro line)
  - \(D_p \leq D_q + 1, \ D_q \leq D_p + 1,\)
  - \(D_q \leq D_r + 1, \ D_r \leq D_q + 1,\)
  - \(D_r \leq D_a + 1, \ D_a \leq D_r + 1,\)
  - \(D_a \leq D_h + 1, \ D_h \leq D_a + 1,\)
  - \(D_h \leq D_d + 1, \ D_d \leq D_h + 1.\)

• (constraints attached to the connections of the yellow metro line)
  - \(D_k \leq D_l + 1, \ D_l \leq D_k + 1,\)
  - \(D_l \leq D_m + 1, \ D_m \leq D_l + 1,\)
  - \(D_m \leq D_a + 1, \ D_a \leq D_m + 1,\)
  - \(D_a \leq D_n + 1, \ D_n \leq D_a + 1,\)
  - \(D_n \leq D_o + 1, \ D_o \leq D_n + 1,\)
  - \(D_o \leq D_i + 1, \ D_i \leq D_o + 1.\)

3.7.149 • Minimum ➤

- \(\text{min.index},\)
- \(\text{min.n},\)
- \(\text{min.nvalue},\)
- \(\text{min.size.set_of_consecutive.var},\)
- \(\text{minimum},\)
- \(\text{minimum_except_0},\)
- \(\text{minimum_greater_than},\)
- \(\text{minimum_modulo},\)
- \(\text{next_element},\)
- \(\text{next_greater_element},\)
- \(\text{open_maximum},\)
- \(\text{open_minimum}.\)

A constraint for which the definition involves the notion of minimum.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

Table 3.18: Domains of the variables for the `soft_same_var` constraint of Figure 3.36.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$dom(x_i)$</th>
<th>$i$</th>
<th>$dom(y_i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${1, 2}$</td>
<td>1</td>
<td>${2}$</td>
</tr>
<tr>
<td>2</td>
<td>${2, 3}$</td>
<td>2</td>
<td>${2}$</td>
</tr>
<tr>
<td>3</td>
<td>${1, 3}$</td>
<td>3</td>
<td>${2, 3}$</td>
</tr>
</tbody>
</table>

3.7.150 **Minimum cost flow**

- `soft_alldifferent_ctr`
- `soft_same_var`

A constraint for which there is a filtering algorithm based on an algorithm that finds a minimum cost flow in a graph. This graph is usually constructed from the variables of the constraint as well as from their potential values. Figure 3.36 illustrates the minimum cost flow model used for the `soft_same_var` constraint. The demand and the capacity of the arcs are depicted by an interval on top of the corresponding arcs. The weight is given after that interval: a weight of 0 (respectively 1) is depicted by a dotted (respectively plain) arc. Weights of 1 are assigned to arcs linking two values since they model the correction of a discrepancy between variables $x_1, x_2, x_3$ and variables $y_1, y_2, y_3$. Blue arcs represent a feasible flow corresponding to the solution `soft_same_var(2, {1, 3, 3}, {2, 2, 3})`.

Figure 3.36: Minimum cost flow model for the `soft_same_var` constraint described in table 3.18.
3.7.151 • Minimum feedback vertex set ➔ [1 CONS]
- cutset.

Denotes that a constraint is related to the minimum feedback vertex set problem: given a connected graph \( G = (V, E) \), find out a minimum cardinality subset \( V' \) of \( V \) such that the graph \( G' \) induced by \( V \setminus V' \) does not contain any cycle. A survey on the feedback vertex set problem is given in [152].

3.7.152 • Minimum hitting set cardinality ➔ [1 CONS]
- nvalue.

Denotes that, by reduction to the problem of finding the cardinality of a minimum hitting set, deciding whether a constraint has a solution or not, or getting a sharp lower bound for one of its arguments, was shown to be NP-hard. The cardinality of a minimum hitting set problem can be described as follows: given a collection \( C \) of subsets of a set \( S \), find the minimum cardinality of \( S' \subseteq S \) such that \( S' \) contains at least one element from each subset in \( C \).

3.7.153 • Minimum number of occurrences ➔ [1 CONS]
- min_nvalue.

A constraint that restricts the minimum number of times that a given value is taken.

3.7.154 • Modulo ➔ [12 CONS]
- alldifferent_modulo,
- among_modulo,
- balance_modulo,
- common_modulo,
- k_same_modulo,
- k_used_by_modulo,
- maximum_modulo,
- minimum_modulo,
- same_modulo,
- soft_same_modulo_var,
- soft_used_by_modulo_var,
- used_by_modulo.

Denotes that the arc constraint associated with a given constraint mentions the function mod.
3.7.155 Multi-site employee scheduling with calendar constraints

- calendar.
- geost.
- diffn.

An international software company located in France and Germany has offices in Paris, Lyon and Marseille as well as in Berlin, Hamburg and Munich. Four types of activities are performed by its employees, namely (1) software development, (2) software deployment, (3) software training courses, and (4) business trips. Software developments tasks and training courses are performed within company’s offices, while software deployment and business trips are done at customer’s sites. Scheduling activities to employees is typically done on a yearly basis from Jan. 1 of current year to Apr. 30 of next year. Considering the first four months of the next year is done in order to absorb eventual overload and to anticipate the effect of Christmas and winter vacations. Without loss of generality we assume that our planning period is from Jan. 1, 2010 to Apr. 30, 2011. The level of granularity is the individual day. Since employees are located on different home sites, one has to consider the following holidays:

- Public holidays that do not fall on a weekend (i.e., a Saturday or a Sunday) are listed below.

- In the context of Germany, regional holidays related to the federal state where a home site is situated. For Munich (Bavaria) we have the following additional days off, that all fall outside a weekend: Jan. 6, June 3, Nov. 1 in 2010 and Jan. 6 in 2011.

- Each home site is closed for a known fixed period of nine consecutive days that is located during summer school vacations. In addition each employee has five consecutive days off, a priori known, crossing winter school vacation. Summer and winter school vacations are linked to the country and the area where a home site is located. Regarding school vacations, France is partitioned in three zones, while Germany is divided in 16 federal states. Paris, Lyon and Marseille are located in distinct zones, while Berlin, Hamburg and Munich are situated in different federal states. Summer vacations periods are:
  - From July 7, 2010 to Aug. 21, 2010 in Berlin.
  - From July 8, 2010 to Aug. 18, 2010 in Hamburg.

Winter vacations periods are:


The goal is to schedule a given set of known tasks to employees in such a way that each employee has 30 days off in 2010, some of them corresponding to the mandatory public and regional holidays depending of the home site of an employee. Each task has:

1. A type (i.e., software development, software deployment, software training courses, and business trips).
3. A latest end in 2010. Tasks which cannot be allocated with respect to their 2010 time window must be scheduled in early 2011, i.e., from Jan. 1, 2011 to Apr. 30, 2011.
4. A duration.
5. A number of required employees.
6. A list of home sites qualified to perform the task.

Business trips, training courses and software deployment cannot be interrupted at all, while software development tasks cannot be interrupted by summer vacation. Business trips have to start on a Monday or a Tuesday since the general company policy is to prevent people staying abroad during weekends. Each task has to be allocated to employees, which are all based on the same home site, in such a way that the same set of employees takes care of the task from its start towards its completion. Each employee has:

1. A home site (i.e., Paris, Lyon, Marseille, Berlin, Hamburg or Munich).
2. A five days period of winter 2010 vacation.
3. A five days period of winter 2011 vacation.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

4. A list of task types (i.e., software development, software deployment, software training courses, business trips) it can handle.

Finally, each home site has a nine days period of summer 2010 vacation where the home site is closed down.

3.7.156 ▼ Multiset ➔

- k\_same,
- k\_used\_by,
- same,
- same\_and\_global\_cardinality,
- same\_and\_global\_cardinality\_low\_up,
- used\_by.

A constraint using domain variables that can be used for modelling some constraint between multisets.

3.7.157 ▼ Multiset ordering ➔

- lex\_greater,
- lex\_greater\_eq,
- lex\_less,
- lex\_less\_eq.

Similar constraints exist also within the context of multisets.

3.7.158 ▼ No cycle ➔

- proper\_forest.

A constraint enforcing the fact that an undirected graph has no cycle.

3.7.159 ▼ No loop ➔

- all\_differ\_from\_at\_least\_k\_pos,
- all\_different\_on\_intersection,
- all\_incomparable,
- among\_low\_up,
- among\_var,
- arith\_or,
- assign\_and\_counts,
- assign\_and\_nvalues,
- bin\_packing,
- cardinality\_at\_least,
- cardinality\_at\_most\_partition,
- cardinality\_at\_most,
- change\_continuity,
- change\_pair,
• change_partition,
• change,
• common_interval,
• common_modulo,
• common_partition,
• common,
• correspondence,
• counts,
• crossing.

- cutset,
- cyclic_change_joker,
- cyclic_change,
- decreasing,
- inverse_within_range,
- lex_equal,
- two_orth_do_not_overlap,
- uses.

Denotes a constraint defined by a graph constraint for which the final graph doesn’t have any loop.

3.7.160  ➤ n-Amazon ➤  [4 CONS]

• alldifferent,
• alldifferent_cst,
• inverse,
• smooth.

A constraint that can be used for modelling the n-Amazon problem. Place n Amazons on a n by n chessboard in such a way that no Amazon attacks another. We say that two columns (respectively two rows) of a chessboard are almost adjacent if and only if the two columns (respectively the two rows) are separated by one single column (respectively one single row). Two Amazons attack each other if at least one of the following conditions holds:

1. They are located on the same column, on the same row or on the same diagonal.
2. They are located either on adjacent columns and on almost adjacent rows, or on almost adjacent columns and on adjacent rows.

As shown by these conditions, an Amazon combines the movements of a queen and of a knight. Figure 3.37 illustrates the movements of an Amazon. The n-Amazon problem has no solution when n is smaller than 10.

We now show how to model the n-Amazon problem with six global constraints. We start from the model that is used for the n-queen problem. We associate to the i-th column of the chessboard a domain variable \(X_i\) that gives the row number where the corresponding queen is located.

- The fact that two Amazons should not be located on the same column, on the same row or on the same diagonal can be modelled as the conjunction of three alldifferent constraints:
  
  - \text{alldifferent}(X_1, X_2 + 1, \ldots, X_n + n - 1) for the upper-left to lower-right diagonals,
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

- **alldifferent**\((X_1, X_2, \ldots, X_n)\) for the rows,
- **alldifferent**\((X_1 + n - 1, X_2 + n - 2, \ldots, X_n)\) for the lower-right to upper-left diagonals.

- The fact that two Amazons cannot both be located on adjacent columns and on almost adjacent rows can be modelled by disequality constraints of the form \(|X_i - X_{i+1}| \neq 2\) \((1 \leq i \leq n - 1)\).

- Similarly, the fact that two Amazons cannot both be located on almost adjacent columns and on adjacent rows can be modelled by disequality constraints of the form \(|X_i - X_{i+2}| \neq 1\) \((1 \leq i \leq n - 2)\). For a reason that will become clear later on, we rewrite this set of disequalities as \(|X_{2i+1} - X_{2i+3}| \neq 1\) \((0 \leq i \leq \lfloor \frac{n-3}{2} \rfloor)\) and \(|X_{2i} - X_{2i+2}| \neq 1\) \((1 \leq i \leq \lfloor \frac{n-2}{2} \rfloor)\).

If we combine the constraints of the form \(|X_i - X_{i+1}| \neq 2\) \((1 \leq i \leq n - 1)\) with the three **alldifferent** constraints we get the conjunction of constraints \(X_i - X_{i+1} \neq 0 \land |X_i - X_{i+1}| \neq 1 \land |X_i - X_{i+1}| \neq 2\) \((1 \leq i \leq n - 1)\). This conjunction of three disequalities can be expressed as one single inequality of the form \(|X_i - X_{i+1}| > 2\) \((1 \leq i \leq n - 1)\). Furthermore all these inequalities can be combined into one single **smooth** constraint of the form **smooth**\((n - 1, 2, (X_1, X_2, \ldots, X_n))\).\(^\text{15}\) Similarly we get the constraints \(|X_{2i+1} - X_{2i+3}| > 2\) \((0 \leq i \leq \lfloor \frac{n-3}{2} \rfloor)\) and \(|X_{2i} - X_{2i+2}| > 2\) \((1 \leq i \leq \lfloor \frac{n-2}{2} \rfloor)\). Again we obtain two **smooth** constraints of the form **smooth**\((\lfloor \frac{n-1}{2} \rfloor, 2, (X_1, X_3, \ldots, X_{n-1+i} + n \mod 2))\) and **smooth**\((\lfloor \frac{n-2}{2} \rfloor, 2, (X_2, X_4, \ldots, X_{n-\mod 2}))\).

Finally, the **inverse** constraint can also be used as a channelling constraint if we want to create an additional variable for each row. This may be for instance the case if we want to have a heuristics for selecting first the column or the row that has the smallest number of possibilities.

---

\(^{15}\)Since we enforce for all pairs of consecutive variables \(X_i, X_{i+1} (1 \leq i \leq n - 1)\) the constraint \(|X_i - X_{i+1}| > 2\), the name **smooth** seems odd. However the name **smooth** stands from the situation where the number of inequalities constraints should be minimised.
Figure 3.38 shows the unique solution, modulo symmetries, to the $n$-Amazon problem for $n = 10$. We have the following conjunction of constraints:

- **alldifferent.cst** ($\langle \text{var} - X_1 \text{ cst} - 0, \text{var} - X_2 \text{ cst} - 1, \text{var} - X_3 \text{ cst} - 2, \text{var} - X_4 \text{ cst} - 3, \text{var} - X_5 \text{ cst} - 4, \text{var} - X_6 \text{ cst} - 5, \text{var} - X_7 \text{ cst} - 6, \text{var} - X_8 \text{ cst} - 7, \text{var} - X_9 \text{ cst} - 8, \text{var} - X_{10} \text{ cst} - 9 \rangle$),

- **alldifferent** ($\langle X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10} \rangle$),

- **alldifferent.cst** ($\langle \text{var} - X_1 \text{ cst} - 9, \text{var} - X_2 \text{ cst} - 8, \text{var} - X_3 \text{ cst} - 7, \text{var} - X_4 \text{ cst} - 6, \text{var} - X_5 \text{ cst} - 5, \text{var} - X_6 \text{ cst} - 4, \text{var} - X_7 \text{ cst} - 3, \text{var} - X_8 \text{ cst} - 2, \text{var} - X_9 \text{ cst} - 1, \text{var} - X_{10} \text{ cst} - 0 \rangle$),

- **smooth** ($\langle 9, 2, \langle X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10} \rangle \rangle$),

- **smooth** ($\langle 4, 2, \langle X_1, X_3, X_5, X_7, X_9 \rangle \rangle$),

- **smooth** ($\langle 4, 2, \langle X_2, X_4, X_6, X_8, X_{10} \rangle \rangle$).
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.161 n-queen ➔

- alldifferent.
- alldifferent_cst.
- inverse.

A constraint that can be used for modelling the \( n \)-queen problem. Place \( n \) queens on a \( n \) by \( n \) chessboard in such a way that no queen attacks another. Two queens attack each other if they are located on the same column, on the same row or on the same diagonal. A constructive method for arbitrary \( n > 3 \) was first given in [147]. An effective heuristics for the \( n \)-queen problem was given in [211]. It consists of starting to place the queens in the center of the chessboard so that they eliminate the maximum number of potential positions.

3.7.162 Non-deterministic automaton ➔

- among.
- change.
- smooth.

A constraint for which the catalogue provides a non-deterministic automaton without counters and without array of counters. For the mentioned constraints it turn out that non-determinism is due to the fact that we introduce transitions labelled by the potential values of a counting variable to a single final state (i.e., see Figures 5.26, 5.99, and 5.548).

3.7.163 Non-overlapping ➔

- diffn.
- disjoint_tasks.
- geost.
- geost_time.
- orth_on_top_of_orth.
- orths_are_connected.
- place_in_pyramid.
- two_orth_are_in_contact.
- two_orth_do_not_overlap.

A constraint that forces a collection of geometrical objects to not pairwise overlap.
3.7.164  ▶ Number of changes  ➞  [8 CONS]

- change,
- change_pair,
- change_partition,
- change_vectors,
- circular_change,
- cyclic_change,
- cyclic_change_joker,
- smooth.

A constraint restricting the number of times that a given binary constraint holds on consecutive items of a given collection.

3.7.165  ▶ Number of distinct equivalence classes  ➞  [13 CONS]

- atLeast_nvalue,
- atLeast_nvector,
- atMost_nvalue,
- atMost_nvector,
- increasing_nvalue,
- nclass,
- nequivalence,
- ninterval,
- npair,
- nvalue,
- nvalues,
- nvectors.

A constraint on the number of distinct equivalence classes assigned to a collection of domain variables.

3.7.166  ▶ Number of distinct values  ➞  [11 CONS]

- atLeast_nvalue,
- atMost_nvalue,
- assign_and_nvalues,
- coloured_cumulative,
- coloured_cumulatives,
- increasing_nvalue,
- increasing_nvalue_chain,
- nvalue,
- nvalue_on_intersection,
- nvalues,
- nvalues_except_0.

A constraint on the number of distinct values assigned to one or several set of variables.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.167 ▲ Obscure ▲ [6 CONS]

- change_continuity,
- group,
- group_skip_isolated_item,
- longest_change,
- sliding_card_skip0,
- two_layer_edge_crossing.

A constraint for which a better description is needed (i.e., two_layer_edge_crossing), or a constraint for which the automata need to be checked because the removal of the dollar sign may have introduced an error (i.e., the five other constraints).

3.7.168 ▲ One succ ▲ [20 CONS]

- alldifferent,
- alldifferent_between_sets,
- alldifferent_cst,
- alldifferent_except_0,
- alldifferent_interval,
- alldifferent_modulo,
- alldifferent_partition,
- balance_cycle,
- balance_path,
- balance_tree,
- binary_tree,
- circuit,
- circuit_cluster,
- cycle,
- cycle_card_on_path,
- derangement,
- minimum_weight_alldifferent,
- path,
- permutation,
- tree.

Denotes that a constraint is defined by one single graph constraint such that:

- All the vertices of its initial graph belong to the final graph,
- All the vertices of its final graph have exactly one successor.
A constraint for which the set of solutions can be recognised by a so called open automaton. An open automaton is a finite deterministic automaton taking as input a sequence of variables $V_1 V_2 \ldots V_n$ as well as a sequence of 0-1 variables $B_1 B_2 \ldots B_n$. A variable $B_i (1 \leq i \leq n)$ set to value 0 means that the corresponding variable $V_i$ is removed from the sequence of variables $V_1 V_2 \ldots V_n$.

Consider a constraint $C$ for which we already have a finite deterministic automaton $A$ that only accepts the set of solutions of $C$. Constructing the finite deterministic automaton $A'$ that only recognises the set of solutions of the open version of constraint $C$ can be done in a systematic way from the automaton $A$. First, to each transition of $A$ we add the fact that the corresponding Boolean variable must also be equal to 1. Second, to each state of $A$ we add a loop transition for which the corresponding Boolean variable $B_i (1 \leq i \leq n)$ must be equal to 0 (since variable $V_i$ is ignored, we stay within the same state). Figure 3.39 illustrates this construction in the context of the minimum constraint and of its open counterpart, the open minimum constraint.

Figure 3.39: (Constructing the (B) automaton of the open minimum constraint from the (A) automaton of the minimum constraint)
3.7.170 ▶Open constraint ◀

- open_alldifferent,
- open_among,
- open_atleast,
- open_atmost,
- open_global_cardinality,
- open_global_cardinality_low_up,
- open_maximum,
- open_minimum,
- size_max_starting_seq_alldifferent.

A constraint from which all its variables are not completely known when the constraint is posted [402]. In many situations, such as configuration, planning, or scheduling of process dependant activities, the variables of a constraint are not completely known initially when the constraint is posted. Instead, they are revealed during the search process [20, 148, 149]. In practice, an additional argument of the constraint (a set variable or a set of 0-1 variables) provides the initial set of potential variables (the lower bound in the context of a set variable). In Bartak’s model [20], an open constraint admits a sequence of domain variables $V_1 V_2 \ldots V_m \ (m \geq 1)$ as well as an additional variable $C$ which gives the index of the last variable that effectively belongs to the constraint (i.e., variables $V_{C+1}, V_{C+2}, \ldots, V_m$ are discarded). This is for instance the case for the size_max_starting_seq_alldifferent constraint.

Within the context of open constraints, the notion of contractibility was introduced in [253] in order to characterise a global constraint for which any pruning rule that removes a value from one of its variable (or that enforces any type of condition) can be reused in the context of the corresponding open global constraint (i.e., the pruning rule still makes valid deductions in the context of the open case). Intuitively, many global constraints that impose a kind of at most condition are contractible, while this is typically not the case for global constraints that enforce a kind of at least condition.

See also the keywords open_automaton_constraint, contractible, and extensible.
3.7.171  Order constraint

- allperm,
- cond_lex_cost,
- cond_lex_greater,
- cond_lex_greatereq,
- cond_lex_less,
- cond_lex_leq,
- decreasing,
- increasing,
- increasing_global_cardinality,
- increasing_nvalue,
- increasing_nvalue_chain,
- increasing_sum,
- int_value_precede,
- int_value_precede_chain,
- lex2,
- lex_between,
- lex_chain_less,
- lex_chain_leq,
- lex_greater,
- lex_greatereq,
- lex_less,
- lex_leq,
- lex_leq_allperm,
- max_index,
- max_n,
- maximum,
- maximum_modulo,
- min_index,
- min_n,
- minimum,
- minimum_except_0,
- minimum_greater_than,
- minimum_modulo,
- next_greater_element,
- open_maximum,
- open_minimum,
- ordered_atleast_nvector,
- ordered_atmost_nvector,
- ordered_global_cardinality,
- ordered_nvector,
- precedence,
- set_value_precede,
- strict_lex2,
- strictly_decreasing,
- strictly_increasing.

A constraint involving an ordering relation in its definition. An ordering relation \( R \) on a set \( S \) is a relation such that, for every \( a, b, c \in S \):

- \( a R b \) or \( b R a \),
- If \( a R b \) and \( b R c \), then \( a R c \),
- If \( a R b \) and \( b R a \) then \( a = b \).
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.172 Orthotope ➔

- diffn
- diffn_column
- diffn_include
- orth_link_orient_end
- orth_on_the_ground
- orth_on_top_of_orth
- orths_are_connected
- place_in_pyramid
- two_orth_are_in_contact
- two_orth_column
- two_orth_do_not_overlap
- two_orth_include

![Illustration of orthotopes for various dimensions](image)

Figure 3.40: Illustration of the notion of orthotope for various dimensions

A constraint involving orthotopes. An orthotope corresponds to the generalisation of the rectangle and box to the $n$-dimensional case. In addition its sides are parallel to the axes of the placement space. Figure 3.40 illustrates the notion of orthotope for $n = 1, 2, 3$ and 4.

3.7.173 Overlapping alldifferent ➔

- k_alldifferent

A constraint expressing several alldifferent constraints having some variables in common.

3.7.174 Pair ➔

- change_pair
- npair
- twin

A constraint involving a collection of pairs of variables.
3.7.175  ▶Packing almost squares ◀[2 CONS]

- `diffn`,
- `geost`.

Denotes that a constraint can be used for solving the packing almost squares problem: tile a rectangle for which sides are consecutive integers by rectangles of size $1 \times 2, 2 \times 3, \ldots, n \times (n + 1)$ which can be rotated by 90 degrees. The problem is described in [http://www.stetson.edu/~efriedma/almost/](http://www.stetson.edu/~efriedma/almost/). Since there does not always exist a tiling, one can also consider a variant where the goal is to find the rectangle with minimal area. Figure 3.41 provides a solution for $n = 26$ found by H. Simonis.

![Figure 3.41: A solution to the packing almost squares problem for $n = 26$](image)

3.7.176  ▶Pallet loading ◀[2 CONS]

- `diffn`,
- `geost`.

A constraint that can be used for modelling the pallet loading problem. The pallet loading problem consists of packing a maximum number of identical rectangular boxes onto a rectangular pallet in such a way that boxes are placed with their edges parallel to the edges of the pallet. The problem often arises in distribution, when many boxes must be shipped and an increase of the number of boxes on a pallet saves costs. Even if the complexity of the problem is not yet known [272], many solutions have been developed over the past years:

- Exact algorithms based on tree search procedures extend a partial solution by positioning a new box according to different heuristics. One of the most used heuristics is the so called G4 heuristics [352] which recursively divides the placement space into four huge rectangles. Beside the use of an appropriate heuristics, the key point is the use of upper bounds on the maximum number of boxes that can be packed. Some bounds like the Barnes [18] and the Keber [219] bounds consider the geometric structure of the problem. Some other bounds are obtained by solving a linear programming problem [204].
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

- Approximate algorithms are based on constructive methods (i.e., methods that either divide the pallet into blocks or methods that divide the pallet in a recursive way) or metaheuristics based on genetic algorithms or tabu search [5].

Both in the context of exact and approximate algorithms, the problem is usually first normalised in order to reduce the set of possible solutions [133, 134].

3.7.177 ▼ Partition ➡ [14 CONS]

- alldifferent_partition,
- balance_partition,
- cardinality_atmost_partition,
- change_partition,
- common_partition,
- in_same_partition,
- k_same_partition,
- k_used_by_partition,
- nclass,
- same_partition,
- stretch_path_partition,
- soft_same_partition_var,
- soft_used_by_partition_var,
- used_by_partition.

A constraint involving in one of its argument a partitioning of a given finite set of integers.

3.7.178 ▼ Path ➡ [4 CONS]

- balance_path,
- path,
- path_from_to,
- temporal_path.

A constraint allowing for expressing that we search for one or several vertex-disjoint simple paths. Within a digraph a simple path is a set of links that are traversed in the same direction and such that each vertex of the simple path is visited exactly once.

3.7.179 ▼ Partridge ➡ [2 CONS]

- diffn,
- geost.

Denotes that a constraint can be used for solving the Partridge problem: the Partridge problem consists of tiling a square of size \( \frac{n(n+1)}{2} \) by \( \frac{n(n+1)}{2} \) squares of respective sizes

- 1 square of size 1,
• 2 squares of size 2,
• . . . ,
• \(n\) squares of size \(n\).

It was initially proposed by R. Wainwright and is based on the identity
\[
1 \cdot 1^2 + 2 \cdot 2^2 + \cdots + n \cdot n^2 = \left(\frac{n(n+1)}{2}\right)^2.
\]
The problem is described in http://mathpuzzle.com/partridge.html. Figure 3.42 gives a solution for \(n = 12\) found with geost.

\[
\begin{array}{cccccccc}
6 & 6 & 12 & 11 & 8 & 8 & 9 & 9 \\
12 & 12 & 11 & 8 & 8 & 7 & 10 & 10 \\
12 & 12 & 11 & 8 & 11 & 7 & 7 & 10 \\
12 & 12 & 11 & 9 & 10 & 7 & 7 & 10 \\
12 & 12 & 11 & 8 & 11 & 11 & 6 & 6 \\
12 & 12 & 12 & 7 & 5 & 5 & 9 & 9 \\
12 & 12 & 11 & 11 & 11 & 11 & 10 & 10 \\
\end{array}
\]

Figure 3.42: A solution to the Partridge problem for \(n = 12\)

\[3.7.180\] Pattern sequencing

A constraint allowing for expressing the pattern sequencing problem as one single global constraint. The pattern sequencing problem \([153]\) can be described as follows: given a 0-1 matrix in which each column \(j\) \((1 \leq j \leq p)\) corresponds to a product required by the customers and each row \(i\) \((1 \leq i \leq c)\) corresponds to the order of a particular customer (The entry \(c_{ij}\) is equal to 1 if and only if customer \(i\) has ordered some quantity of product \(j\).), the objective is to find a permutation of the products such that the maximum number of open orders at any point in the sequence is minimised. Order \(i\) is open at point \(k\) in the production sequence if there is a product required in order \(i\) that appears at or before position \(k\) in the sequence and also a product that appears at or after position \(k\) in the sequence.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.181 ▶Pentomino ◀

- diffn,
- geost,
- polyomino,
- regular.

A constraint (i.e., polyomino) that can be used to model a pentomino. A pentomino is an arrangement of five unit squares that are joined along their edges. Also denotes a constraint (i.e., diffn, geost, regular) that can be used for solving tiling problems involving pentominoes. For instance, the geost and regular constraints where respectively used in \[36\] and in \[228\] to solve such tiling problems. Figure 3.43 presents a tiling of a rectangle with distinct pentominoes.

![Figure 3.43: Tiling a rectangle with pentominoes](image)

3.7.182 ▶Periodic ◀

- period,
- period_except_0,
- period_vectors.

A constraint that can be used for modelling the fact that we are looking for a sequence that has some kind of periodicity.

3.7.183 ▶Permutation ◀

- alldifferent,
- alldifferent_consecutive_values,
- balance_cycle,
- change_continuity,
- circuit,
- circuit_cluster,
- correspondence,
- cycle,
- cycle_card_on_path,
- derangement,
- elements_alldifferent,
- inverse,
- k_alldifferent,
- k_same,
- k_same_interval,
- k_same_modulo,
- k_same_partition,
- same,
3.7.184  ▶Permutation channel ◀

- inverse.

A constraint that allows for modelling the link between a permutation and its inverse permutation. A permutation is a rearrangement of elements, where none are changed, added or lost.

3.7.185  ▶Phi-tree ◀

- disjunctive,
- cumulative.

A constraint for which one of its filtering algorithms uses a balanced binary tree in order to efficiently evaluate the maximum or minimum value of a formula over all possible subsets of tasks $\Omega$ of a given set of tasks $\Phi$. $\Phi$-trees were introduced by P. Vilím, first in the context of unary resources in [407] and in [408, pages 37–40], and later on in the context of cumulative resources [410, 409]. Without loss of generality, let us sketch the main idea behind a $\Phi$-tree in the context of a cumulative resource of capacity $C$. For this purpose we follow the description given in [410]. Given a set of tasks $\Phi$ where each task has an earliest possible start, a latest possible end, a duration and a resource consumption, assume we need to evaluate the earliest completion time over all tasks of $\Phi$ under the hypothesis that we should not exceed the maximum resource capacity $C$. Let us first introduce some notations:

- $\Omega$ denotes any non-empty subset of tasks of $\Phi$.
- $est_\Omega$ is the minimum over the earliest starts of the tasks in $\Omega$.
- $e_\Omega$ is the sum of the surfaces (i.e., the product of the duration by the resource consumption) of the tasks in $\Omega$. 

A constraint that can be used for modelling a permutation or a specific type or characteristic of a permutation. A permutation is a rearrangement of elements, where none are changed, added or lost.
Figure 3.44: Example of Φ-tree associated with four tasks of respective duration and resource consumption $3 \times 4, 1 \times 3, 5 \times 5, 2 \times 4$ and of respective earliest start 1, 3, 8, 9 under the assumption that the maximum capacity of the cumulative resource is equal to 5.

A common estimation of the earliest completion time over all tasks of $\Phi$ is $\max_{\Omega \subseteq \Phi} \left\{ \text{est}_\Omega + \left\lceil \frac{e_\Omega}{C} \right\rceil \right\}$ which can be rewritten as $\left\lceil \max_{\Omega \subseteq \Phi} \left\{ C_{\text{est}_\Omega} + e_\Omega \right\} \right\rceil$. The numerator of the last fraction is called the energy envelope of the set of tasks $\Phi$ and the purpose of a $\Phi$-tree is to evaluate this quantity efficiently. For a node $n$, let $\mathcal{L}(n)$ denote the set of leaves of the sub-tree rooted at $n$. The leaves of the $\Phi$-tree correspond to the tasks of $\Phi$ sorted from left to right by increasing earliest start. Each node $n$ of the $\Phi$-tree records both, the sum of the surfaces of the tasks in $\mathcal{L}(n)$, as well as the energy envelope of the tasks in $\mathcal{L}(n)$. The sum of the surfaces associated with a non-leaf node $n$ of the tree corresponds to the sum of the surfaces of the children of $n$, while the energy envelope of $n$ is equal to the maximum between on the one hand, the energy envelop of its right child and on the other hand the sum of the energy envelop of its left child and the recorded sum of surfaces of its right child (see [410] for a justification of these recursive formulae). Figure 3.44 illustrates the construction of a $\Phi$-tree associated with four given tasks.
3.7.186  Phylogeny  ➔

- stable_compatibility.

A constraint inspired by the area of phylogeny. Phylogeny is concerned by the classification of organism based on genetic connections between species.

3.7.187  Pick-up delivery  ➔

- cycle.

A constraint that was used for modelling a pick-up delivery problem. In a pick-up delivery problem, vehicles have to transport loads from origins to destinations without any transhipment at intermediate locations.

3.7.188  Planarity test  ➔

- circuit.

A constraint that can use the planarity test in its filtering algorithm. The planarity test determines whether a graph can be embedded in the plane.

3.7.189  Polygon  ➔

- diffn.

A constraint that can be generalised to handle polygons.

3.7.190  Positioning constraint  ➔

- diffn_column,  
- diffn_include,  
- two_orth_column,  
- two_orth_include.

A constraint restricting the relative positioning of two or more geometrical objects.
3.7. **KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS**

### 3.7.191 Predefined constraint →  

- abs_value
- atmost1
- bin_packing_capa
- calendar
- colored_matrix
- compare_and_count
- consecutive_values
- cumulative_two_d
- distance
- divisible
- divisible_or
- dom_reachability
- domain
- eq
- eq_cst
- eq_set
- gcd
- geost
- geost_time
- geq
- geq_cst
- graph_isomorphism
- gt
- in_interval_reified
- in_intervals
- in_set
- incomparable
- increasing_sum
- leq
- leq_cst
- lex2
- lex_less_eq_allperm
- lt
- meet_sboxes
- multi_global_contiguity
- multi_inter_distance
- neq
- neq_cst
- opposite_sign
- period
- period_except_0
- period_vectors
- power
- remainder
- same_sign
- scalar_product
- set_value_precede
- sign_of
- soft_cumulative
- strict_lex2
- subgraph_isomorphism
- sum_cubes_ctr
- sum_free
- sum_of_increments
- sum_squares_ctr
- symmetric_alldifferent_except_0
- twin
- visible

A constraint for which the meaning is not explicitly described in terms of graph properties or in terms of automata or in terms of first order logic.
### 3.7.192 Preferences

- `cond_lex_cost`, `cond_lex.less`.
- `cond_lex_greater`, `cond_lex.greatereq`.
- `cond_lex_less`, `cond_lex.lesseq`.

A constraint that can be used for modelling preferences.

### 3.7.193 Producer-consumer

- `cumulative`, `cumulatives`.

A constraint that can be used for modelling problems where a first set of tasks produces a non-renewable resource, while a second set of tasks consumes this resource so that a limit on the minimum or the maximum stock at each instant is imposed.

Parts (A) and (B) of Figure 3.45 describes the simplest variant of the producer-consumer problem [366] where no negative stock is allowed. Given an initial stock, a first set of tasks (i.e., producers) add instantaneously their respective production to the stock (when they are finished), and a second set of tasks (i.e., consumers) take instantaneously from the stock (when they start) the amount of non-renewable resource they need. The problem is to schedule these tasks (i.e., fix the end of the producers and fix the start of the consumers) and to fix for each task the quantity it produces or consumes, so that no negative stock occurs. Part (A) of Figure 3.45 describes an instance of such problem where we respectively have 2 producers and 3 consumers. Part (B) depicts the corresponding cumulative view of the problem. At each timepoint the difference between the top line and the top of the cumulated profile gives the amount of available stock at that timepoint.

A fundamental problem with the previous variant of the producer-consumer problem is that it does not allow to handle the fact that a resource is produced or used gradually. Parts (C) and (D) of Figure 3.45 describes a second variant where this is in fact possible. This is achieved by replacing the rectangle associated with a producer by a task with a decreasing height. At a given instant the cumulated quantity produced by a producer is the difference between the height of that task at its starting time and the height of that task at the considered instant. Conversely a consumer is modelled by a task with an increasing height. At a particular timepoint the cumulated quantity used by a consumer task is the difference between the height of that task at its end and the height of that task at the considered instant. Part (C) of Figure 3.45 describes an instance of such problem where, again, we respectively have 2 producers and 3 consumers. Part (D) depicts the corresponding cumulative view of the problem. As before, at each timepoint the difference between the top line and the top of the cumulated profile gives the amount of available stock at that timepoint.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

Figure 3.45: Producer-consumer models (A,C) and corresponding cumulative views (B,D)

3.7.194  ➤Product ➤ [2 CONS]

- cumulative_product,
  - product_ctr.

A constraint involving a product in its definition.

3.7.195  ➤Program verification ➤ [1 CONS]

- cutset.

A constraint that was used within the application area of program verification.

3.7.196  ➤Proximity constraint ➤ [3 CONS]

- alldifferent_same_value,
  - distance_change,
  - distance_between.

A constraint restricting the distance between two collections of variables according to some measure.
3.7.197 Pure functional dependency

• abs_value,
• among,
• among_diff_0,
• among_interval,
• among_modulo,
• among_var,
• and,
• balance,
• balance_interval,
• balance_modulo,
• balance_partition,
• cardinality_atleast,
• cardinality_atmost,
• cardinality_atmost_partition,
• change,
• change_pair,
• change_partition,
• change_vectors,
• circular_change,
• colored_matrix,
• common,
• common_interval,
• common_modulo,
• common_partition,
• crossing,
• cyclic_change,
• cyclic_change_joker,
• discrepancy,
• distance,
• distance_between,
• distance_change,
• elem,
• element,
• element_product,
• elements,
• eq,
• eq_cst,
A constraint for which the meaning is completely captured by one or more functional dependencies. The negation of such constraints can be directly expressed as a disjunction between the different functional dependencies. We illustrate this point on different examples:

• The negation of the \texttt{nvalue}(n, \langle v_1, v_2, \ldots, v_m \rangle) constraint is defined by \texttt{nvalue}(p, \langle v_1, v_2, \ldots, v_m \rangle) \land n \neq p.

• The negation of the \texttt{common}(n_1, n_2, \langle u_1, u_2, \ldots, u_p \rangle, \langle v_1, v_2, \ldots, v_q \rangle) constraint is defined by \texttt{common}(m_1, m_2, \langle u_1, u_2, \ldots, u_p \rangle, \langle v_1, v_2, \ldots, v_q \rangle) \land (n_1 \neq m_1 \lor n_2 \neq m_2).

• The negation of the \texttt{elements}(\langle \text{index} - i_1 \text{ value} - u_1, \text{index} - i_2 \text{ value} - u_2, \ldots, \text{index} - i_n \text{ value} - u_n \rangle, \langle \text{index} - 1 \text{ value} - v_1, \text{index} - 2 \text{ value} - v_2, \ldots, \text{index} - n \text{ value} - v_n \rangle) constraint is defined by \texttt{elements}(\langle \text{index} - i_1 \text{ value} - w_1, \text{index} - i_2 \text{ value} - w_2, \ldots, \text{index} - i_n \text{ value} - w_n \rangle, \langle \text{index} - 1 \text{ value} - v_1, \text{index} - 2 \text{ value} - v_2, \ldots, \text{index} - n \text{ value} - v_n \rangle) \land (u_1 \neq w_1 \lor u_2 \neq w_2 \lor \cdots \lor u_n \neq w_n).

• The negation of the \texttt{sort}(\langle u_1, u_2, \ldots, u_n \rangle, \langle v_1, v_2, \ldots, v_n \rangle) constraint is defined by \texttt{sort}(\langle u_1, u_2, \ldots, u_n \rangle, \langle w_1, w_2, \ldots, w_n \rangle) \land \texttt{lex}\_\texttt{different}(\langle v_1, v_2, \ldots, v_n \rangle, \langle w_1, w_2, \ldots, w_n \rangle).

\textbf{3.7.198 Quadtree ▶}

\begin{itemize}
  \item \texttt{cumulative\_\_two\_\_d}.
  \item \texttt{diffn}.
\end{itemize}

Denotes that, for a given constraint, a quadtree can be used within its filtering algorithm. A quadtree is a hierarchical data structure based on the recursive decomposition of space. Figure 3.46 illustrates the representation of a two-dimensional binary region (A) with a quadtree (C). A region is subdivided into quadrants, subquadrants, and so on (B), until blocks consist entirely of 1s or entirely of 0s.
3.7.199 ▶Range ◀  [1 CONS]

- range_ctr.

An arithmetic constraint involving a difference between a maximum and a minimum value.

3.7.200 ▶Rank ◀  [2 CONS]

- max_n.
- min_n.

A positioning constraint according to an ordering relation.

3.7.201 ▶RCC8 ◀  [8 CONS]

- contains_sboxes.
- coveredby_sboxes.
- covers_sboxes.
- disjoint_sboxes.
- equal_sboxes.
- inside_sboxes.
- meet_sboxes.
- overlap_sboxes.

Region Connection Calculus (i.e., RCC-8) [318] provides eight topological relations (i.e., disjoint, meet, overlap, equal, covers, coveredby, contains, inside) between two fixed objects such that any two fixed objects are in one and exactly one of these topological relations. Figure 3.47 illustrates the meaning of each topological relation.

Figure 3.46: A region (A), its subdivision in maximal blocks (B), and the corresponding quadtree (C)
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.202 • **Rectangle clique partition** ➜

- nvector.

Denotes that, by reduction to the rectangle clique partition problem, deciding whether a constraint has a solution or not was shown to be NP-hard. The rectangle clique partition problem can be described as follows: given a rectangle graph, can its set of vertices be partitioned into \( k \) subsets of vertices such that all corresponding induced subgraphs correspond to cliques? A rectangle graph is a graph that can be associated with a set of fixed rectangles whose sides are parallel to the axes of the placement space: to each rectangle corresponds a vertex of the rectangle graph, while to each pair of intersecting rectangles corresponds an edge.

3.7.203 • **Regret based heuristics** ➜

- elem.
- element.
- global_cardinality_with_costs.
- sum_ctr.

Assume you have a discrete optimisation problem where the sum of some cost variables should be minimized, and where the cost variables typically have holes in their domain. In this context a regret based heuristics first selects among the not yet fixed cost variables, the one with the largest difference between its second smallest value and its smallest value. The idea is to consider first a variable that would cause the biggest increase in cost if it could not be assigned its minimum value.
3.7.204 ★Regret based heuristics in matrix problems ★ [2 CONS]

- global_cardinality_with_costs,
- sum ctr.

Assume you have a discrete optimisation problem involving a matrix $M$ of decision variables such that there is a cost variable attached to each row of $M$. Moreover assume that the cost associated with each row corresponds to a sum of elementary costs connected with each decision variable of the same row (e.g., we have a sum ctr or a global_cardinality_with_costs constraint on each row of $M$). Now, suppose we want to use a heuristics for fixing the decision variables of matrix $M$ row by row. In this context a question is which row to select first. Since the cost variable $c_r$ associated with a row $r$ corresponds to a sum of elementary costs, it is very unlikely that the cost variable $c_r$ has a hole in its domain. Consequently, we cannot any more use a conventional regret based heuristics which relies on the fact that we have holes in the domains of the cost variables. We still want to use the idea of finding the variable that would potentially cause the biggest increase in cost in the worst case, i.e. if it would have to be assigned to its maximum value. For this purpose we consider the variable for which the difference between its largest value and its smallest value is maximal. In our context we select the row $r$ for which the corresponding cost variable maximizes such difference. First we enumerate in increasing value order on the cost variable associated with row $r$. Second we fix all decision variables of row $r$, using for instance the heuristics described in labelling by increasing cost. Using such cost based heuristics has both some advantage and some drawback:

- The big potential advantage is that, if we can find a first solution at all, then this solution should have a rather small overall cost.

- The potential drawback is that, depending on how strong the row constraints propagate from the maximum total cost associated with a row back to the decision variables of the row, it may be very difficult to find a feasible solution (since assigning the cost variable of a row to its minimum value potentially creates an infeasible problem for which we need to develop a large search tree).

3.7.205 ★Reified automaton constraint ★ [60 CONS]

- and,
- arith,
- arith_or,
- between_min_max,
- clause_and,
- clause_or,
- cond_lex_cost,
- consecutive_groups_of_ones,
- decreasing,
- domain_constraint,
- elem,
- elem_from_to,
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

- element
- element_greatereq
- element_lessseq
- element_matrix
- element_sparse
- elementn
- equivalent
- global_contiguity
- imply
- in
- in_interval
- in_same_partition
- increasing
- increasing_global_cardinality
- increasing_nvalue
- int_value_precede
- int_value_precede_chain
- lex_between
- lex_different
- lex_equal
- lex_greater
- lex_greatereq
- lex_less
- lex_lessseq
- maximum
- minimum
- minimum_except_0
- minimum_greater_than
- nand
- next_element
- no_peak
- no_valley
- nor
- not_all_equal
- not_in
- open_maximum
- open_minimum
- or
- pattern
- sequence_folding
- stage_element
- stretch_path
- stretch_path_partition
- strictly_decreasing
- strictly_increasing
- two_orth_are_in_contact
- two_orth_do_not_overlap
- xor

A constraint \( C(V_1, V_2, \ldots, V_n) \) for which the reified version can be mechanically constructed from the finite deterministic automaton \( A^C \) that only accepts the set of solutions of \( C \). This is done by deriving from \( A^C \) a so called reified automaton \( A^{\neg C}_C \) by:

- First, adding a 0-1 variable \( B \) in front of the sequence of variables \( V_1, V_2, \ldots, V_n \). This new sequence of variables will be passed to the reified automaton \( A^{\neg C}_C \).

- Second, constructing from \( A^C \) the automaton \( A^{\neg C}_C \) that only recognises non-solutions of \( C \).

- Third, building from the two automata \( A^C \) and \( A^{\neg C}_C \) the automaton \( A^{\neg C}_C \). This is done by:
  1. Creating the initial state \( s \) of \( A^{\neg C}_C \).
  2. Adding a transition labelled by value 1 from \( s \) to the initial state of \( A^C \).
  3. Adding a transition labelled by value 0 from \( s \) to the initial state of \( A^{\neg C}_C \).
Figure 3.48: (A) The automaton for recognising the solutions of the global contiguity constraint; (B) the automaton for recognising the non-solutions of the global contiguity constraint; (C) the automaton for the reified global contiguity constraint.

Figure 3.48 illustrates the construction of a reified automaton in the context of the global contiguity constraint. Part (A) recalls the automaton that only recognises the solutions of the global contiguity constraint. Assuming the same alphabet \( \{0, 1\} \), Part (B) provides the automaton that only recognises the non-solutions of the global contiguity constraint. Finally, Part (C) depicts the reified automaton constructed from the two automata given in parts (A) and (B).

3.7.206 **Reified constraint**

- \texttt{in\_interval\_reified} (reified version of \texttt{in\_interval}).

The reified version \( CR \) of a given constraint \( C \), where \( CR \) has as arguments all arguments of \( C \) plus one extra 0-1 variable. This 0-1 variable is set to 1 when constraint \( C \) holds, and 0 otherwise. Note that constraint \( CR \) inherits from all restrictions of constraint \( C \) (i.e., incorrect parameters for constraint \( C \) are also incorrect for constraint \( CR \)). Within the context of linear programming the extra 0-1 variable is often called an indicator variable.

It was shown in [33] how to reify a global constraint by reformulating it as a conjunction of pure functional dependency constraints together with a constraint that can be easily reified (e.g., an automaton with or without counter, or a Boolean combination of linear arithmetic equalities and inequalities and 0-1 variables).
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.207  ▶Relation ◀

- `in_relation`,
- `symmetric_cardinality`,
- `symmetric_gcc`.

A constraint that allows for representing the access to an element of a relation or to model a relation. A relation is a subset of the product of several finite sets.

3.7.208  ▶Relaxation ◀

- `alldifferent_except_0`,
- `diffn`,
- `geost`,
- `relaxed_sliding_sum`,
- `soft_alldifferent_ctr`,
- `soft_alldifferent_var`,
- `soft_all_equal_max_var`,
- `soft_all_equal_min_ctr`,
- `soft_all_equal_min_var`,
- `soft_cumulative`,
- `soft_same_interval_var`,
- `soft_same_partition_var`,
- `soft_same_var`,
- `soft_same_modulo_var`,
- `soft_used_by_interval_var`,
- `soft_used_by_partition_var`,
- `soft_used_by_var`,
- `sum_of_weights_of_distinct_values`,
- `weighted_partial_alldiff`.

Denotes that a constraint allows for specifying a partial degree of satisfaction. For the constraints `diffn` and `geost` see the keyword Relaxation dimension.
3.7.209  ▶Relaxation dimension ◀  [2 CONS]

- diffn,
- geost.

A constraint that allows to model constraint relaxation in the context of placement problems. This is achieved by adding an extra dimension to the placement space where objects that are really considered are in the foreground, while objects that are discarded are rejected in the background. As a concrete example, consider a slight modification on the data of the task assignment and scheduling problem that is described at the keyword entry assigning and scheduling tasks that run in parallel. In this problem the four nurses were all not available during the time periods [0, 0], [7, 7], [12, 12] and [22, 22]. We now rather consider the following unavailability periods [0, 0], [8, 8], [12, 12] and [22, 22]. Under this new hypothesis we cannot anymore schedule all the five operations tasks \( t_1, t_2, t_3, t_4 \) and \( t_5 \), i.e., we get a no solution answer if we use the model described in assigning and scheduling tasks that run in parallel. In this model we are using a two-dimensional geost constraint, where the first and second dimensions respectively correspond to the time and resource axes. Now, in order to permit relaxation, we introduce a third dimension, a relaxation dimension. The idea is to map each task to a parallelepiped for which the size in the relaxation dimension is equal to one. In addition, the coordinate of a parallelepiped in the relaxation dimension is a variable taking its value in the interval \([1, n]\), where \( n \) represents the number of operations to schedule (i.e., to each operation task \( t_i \) \( 1 \leq i \leq n = 5 \)) we create a coordinate variable \( r_i \) where \( r \) stands for relaxation. Then, all parallelepipeds for which the coordinate in the relaxation dimension if set to 1 correspond to operations that are effectively scheduled, while all other parallelepipeds represent operations that are discarded. On the one hand, this model allows to directly express relaxation right from the beginning without introducing any extra soft constraint and without dynamically adding any constraint during search. On the other hand, one disadvantage is that the model does not directly consider an optimisation criteria like, for instance, the maximum number of tasks effectively scheduled, or the sum of the duration of the tasks effectively done; this can be modelled using extra constraints but this does not provide sharp bounds on the optimisation criteria. Nevertheless, this gives a compact model, specially in the context where additional constraints make more difficult the computation of a sharp bound. Going back to the example described at the keyword entry assigning and scheduling tasks that run in parallel, we get the following three-dimensional geost constraint.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

geost(3,
  {oid - 1 sid - 2 x - (\text{oid}_1, \text{sid}_1, r_1),
   \text{oid - 2 sid - 2 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 3 sid - 2 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 4 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 5 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 6 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 7 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 8 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 9 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 10 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 11 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 12 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 13 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 14 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 15 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 16 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 17 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 18 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 19 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 20 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 21 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 22 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 23 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 24 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 25 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 26 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 27 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 28 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 29 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 30 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 31 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 32 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 33 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 34 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 35 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 36 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 37 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 38 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 39 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 40 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 41 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 42 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 43 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 44 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 45 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 46 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{oid - 47 sid - 4 x - (\text{oid}_1, \text{sid}_1, r_1),}
   \text{t - (0, 0, 0) 1 - (1, 1, 1),
   \text{sid - 2 t - (0, 0, 0) 1 - (2, 1, 1),
   \text{sid - 3 t - (0, 0, 0) 1 - (3, 1, 1),
   \text{sid - 4 t - (0, 0, 0) 1 - (4, 1, 1),
   \text{sid - 5 t - (0, 0, 0) 1 - (5, 1, 1),
   \text{t - (0, 0, 0) 1 - (6, 1, 1))},
}\]

Figure 3.49 depicts a solution to the problem corresponding to the assignment \( a_1 = 9, \) \( r_1 = 1, \) \( a_2 = 1, \) \( a_3 = 4, \) \( n_{11} = 5, \) \( n_{22} = 6, \) \( a_2 = 1, \) \( r_2 = 2, \) \( a_3 = 2, \) \( r_3 = 3, \) \( s_{31} = 3, \) \( s_{42} = 4, \) \( n_{31} = 5, \) \( n_{32} = 6, \) \( a_4 = 17, \) \( r_4 = 1, \) \( a_4 = 1, \) \( n_{41} = 5, \) \( n_{42} = 6, \) \( n_{43} = 7, \) \( a_5 = 16, \) \( r_5 = 1, \) \( a_5 = 2, \) \( s_{5} = 3, \) \( n_{5} = 8, \) during search, relaxation variables \( r_1, r_2, r_3, r_4, r_5 \) are first set to value one (i.e., the corresponding operations are scheduled) and then, upon backtracking, assigned to any value greater than one (i.e., there is no backtracking on the values that are greater than one since we just want to reject an operation in the background).

3.7.210 •Resource constraint ➞ [19 CONS]

• bin_packing,
• bin_packing_caps,
• coloured_cumulative,
• coloured_cumulatives,
• cumulative,
• cumulative_convex,
• cumulative_product,
• cumulative_with_level_of_priority,
• cumulatives,
• cycle_resource,
• operations effectively scheduled (in the foreground of the relaxation dimension): \( t_1 \), \( t_3 \), \( t_4 \), \( t_5 \)
• operation ignored (in the background of the relaxation dimension): \( t_2 \)

Figure 3.49: A partial solution for the operation scheduling problem that maximises the number of operations actually performed where only operation \( t_2 \) is not scheduled

A constraint restricting the utilisation of a resource. The utilisation of a resource is computed from all items that are assigned to that resource.

### 3.7.211 Run of a permutation

- change continuity.

A constraint that can be used for putting a restriction on the size of the longest run of a permutation. A run is a maximal increasing contiguous subsequence in a permutation.

### 3.7.212 SAT

- alldifferent,
- among,

A constraint for which a reference provides a reformulation in SAT. Encoding for the alldifferent and the among constraints were respectively provided in [175]
and in [14]. Based on Fekete et al. model of the multi-dimensional orthogonal packing problem [151], an encoding for the \texttt{diffn} constraint when all the sizes of all the orthotopes are fixed was described in [183].

3.7.213  \textbf{Scalar product}  ➤

- \texttt{global_cardinality_with_costs}.

A constraint that can be used for modelling a scalar product constraint.

3.7.214  \textbf{Sequence}  ➤

- \texttt{among_seq},
- \texttt{arith_sliding},
- \texttt{change_continuity},
- \texttt{cycle_card_on_path},
- \texttt{deepest_valley},
- \texttt{global_contiguity},
- \texttt{group},
- \texttt{group_skip_isolated_item},
- \texttt{highest_peak},
- \texttt{inflexion},
- \texttt{no_peak},
- \texttt{no_valley},
- \texttt{multi_global_contiguity},
- \texttt{invisible_from_end},
- \texttt{invisible_from_start},
- \texttt{peak},
- \texttt{period},
- \texttt{period_except_0},
- \texttt{period_vectors},
- \texttt{relaxed_sliding_sum},
- \texttt{sequence_folding},
- \texttt{size_max_seq_alldifferent},
- \texttt{size_max_starting_seq_alldifferent},
- \texttt{sliding_card_skip0},
- \texttt{sliding_distribution},
- \texttt{sliding_sum},
- \texttt{stretch_path},
- \texttt{stretch_path_partition},
• valley.

Constrains consecutive variables (possibly not all) of a given collection of domain variables or consecutive vertices of a simple path or a simple circuit. Also a constraint restricting a variable (when fixed to 0 the variable may be omitted) according to consecutive variables of a given collection of domain variables.

3.7.215 ▶ Sequence dependent set-up ◀

• diffn, • disjunctive, • elem, • elem.
• temporal_path.

Denotes that a constraint can be used for modelling sequence dependent set-up between pairs of tasks. Given,

• a collection of \( n \) tasks \( T \), where each task \( t_i \in T \) \( (1 \leq i \leq n) \) has an origin \( o_i \), a duration \( d_i \), an end \( e_i \) \( (o_i + d_i = e_i) \) and a machine \( m_i \) to which it will be assigned,

• and a \( n \) by \( n \) matrix \( M \) of positive integers \( \delta_{ij} \) \( i, j \in [1, n] \) where \( \text{row}_i \) denotes the \( i^{th} \) row of matrix \( M \),

we want to express that \( \delta_{ij} \) enforces a minimum distance between the completion of task \( t_i \in T \) and the start of task \( t_j \in T \) \( (i \neq j) \) under the hypotheses that (a) both tasks are assigned the same machine (i.e., \( m_i = m_j \)) and that (b) task \( t_j \) immediately follows task \( t_i \) (i.e., there is no task \( t_k \in T \) \( (k \notin \{i, j\}) \) such that \( m_k = m_i \land e_i \leq o_k \land e_k \leq o_j \)). In addition, tasks assigned to the same machine should not overlap (i.e., \( \forall i \in [1, n], \forall j \neq i \in [1, n] \) such that \( m_i = m_j \) we have \( e_i \leq o_j \lor e_j \leq o_i \)). We show how to model the previous sequence dependent set-up constraint under the hypothesis that we have one single machine. Without loss of generality we assume that \( \delta_{ii} = 0 \) for all \( i \in [1, n] \).

In a first phase we create for each task \( t_i \in T \) \( (1 \leq i \leq n) \) three additional variables \( s_i, g_i, c_i \) that respectively correspond to:

• The successor variable \( s_i \in [1, n] \) allows to get the immediate successor of task \( t_i \). On the one hand, the assignment \( s_i = i \) denotes that task \( t_i \) has no immediate successor (i.e., task \( t_i \) is the last task running on machine \( m_i \)), on the other hand, \( s_i = j \ (j \neq i) \) denotes that task \( t_j \) is the immediate successor of task \( t_i \).

• The gap variable \( g_i \) represents the size of the gap between the end of task \( t_i \) and the start of its immediate successor (the gap is equal to 0 when task \( t_i \) has no immediate successor).

• The extended completion variable \( c_i \) represents the sum of the end of task \( t_i \) and the corresponding gap variable \( g_i \) (i.e., \( c_i = e_i + g_i \)).
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

In a second phase we post for each task \( t_i \in T \ (1 \leq i \leq n) \) the following constraints:

- An \( \text{element}(s_i, r_{ow_i}, g_i) \) constraint to make the link between the successor variable \( s_i \) and the gap variable \( g_i \).
- A \( \text{sum}_\text{ctr}([e_i, g_i], =, c_i) \) constraint.

Finally in a third phase we create a collection of nodes \( \text{NODES} \) where each item corresponds to a task \( t_i \in T \ (1 \leq i \leq n) \) and has an index attribute set to \( i \), a \( \text{succ} \) attribute set to \( s_i \), a \( \text{start} \) attribute set to \( o_i \) and an \( \text{end} \) attribute set to \( c_i \). We post a \( \text{temporal\_path}(1, \text{NODES}) \) constraint for linking the successor variables, the start variables and the extended completion variables associated with the different tasks. The first argument of the \( \text{temporal\_path} \) constraint enforces one single path corresponding to the succession of the different tasks on the unique machine.

3.7.216 ▶ Sequencing with release times and deadlines ▶ [5 CONS]

- \( \text{cumulative}, \)
- \( \text{cumulatives}, \)
- \( \text{diffn}, \)
- \( \text{disj}, \)
- \( \text{disjunctive}. \)

Denotes that, by reduction to sequencing with release times and deadlines, deciding whether a constraint has a solution or not was shown to be NP-hard. The sequencing with release times and deadlines problem can be described as follows: given a set of non-overlapping tasks and, for each task a length, a release time and a deadline the question is to find a schedule that satisfies all release time constraints and meets all the deadlines.

3.7.217 ▶ Set channel ▶ [2 CONS]

- \( \text{inverse\_set}, \)
- \( \text{link\_set\_to\_booleans}. \)

A channelling constraint involving one or several set variables.
3.7.218  ★Set packing ★

- \texttt{k.alldifferent}.

Denotes that, by reduction to \textit{set packing}, deciding whether a constraint has a solution or not was shown to be NP-hard. The \textit{set packing} problem can be described as follows: given a collection \( C \) of \( n \) finite sets, and a positive integer \( m \leq n \), does \( C \) contains \( m \) disjoint sets?

3.7.219  ★Shikaku ★

- \texttt{diffn}.
- \texttt{geost}.

Denotes that a constraint can be used for solving the \textit{Shikaku} puzzle. Given a rectangular grid, where exactly \( n \) cells contain an integer value, the problem is to tile that grid by \( n \) rectangles in such a way that the surface of each rectangle is equal to the single integer it contains.

![Figure 3.50: An example of a Shikaku puzzle and its corresponding unique solution](https://member.nikoli.com/index.html)

Parts (A) and (B) of Figure 3.50 respectively show a small instance of such a puzzle and its corresponding unique solution taken from the Nikoli website [https://member.nikoli.com/index.html](https://member.nikoli.com/index.html).
3.7.220  Scheduling constraint → [19 CONS]

- all_min_dist,
- calendar,
- coloured_cumulative,
- coloured_cumulatives,
- cumulative,
- cumulative_convex,
- cumulative_product,
- cumulative_with_level_of_priority,
- cumulatives,
- disjoint_tasks,
- disj,
- disjunctive,
- disjunctive_or_same_end,
- disjunctive_or_same_start,
- multi_inter_distance,
- period,
- period_except_0,
- shift,
- soft_cumulative.

A constraint useful for the area of scheduling. Scheduling is concerned with the allocation or assignment of resources (e.g., manpower, machines, money), over time, to a set of tasks.

3.7.221  Scheduling with machine choice, calendars and preemption → [4 CONS]

- calendar,
- cumulatives,
- diffn,
- geost.

A set of constraints that can be used for modelling a scheduling problem where:

- We have tasks that have both to be assigned to machine and time.
- Each task has a fixed duration.
- Machines can run at most one task at a given instant.
- Each machine has its own fixed unavailability periods (i.e., a calendar of unavailability periods).
- An unavailability period that allows (respectively forbids) a task to be interrupted and resumed just after is called crossable (respectively non-crossable). A task that can be (respectively cannot be) interrupted by a crossable unavailability period is called resumable (respectively non-resumable).
- We have a precedence constraint between specific pairs of tasks. Each precedence enforces that a given task ends before the start of another given task.

This model illustrates the use of two time coordinates systems:
The first coordinate system, so called the virtual coordinate system, does not consider at all the crossable unavailability periods associated with the different machines. Since resumable tasks can be preempted by machine crossable unavailability, all resource scheduling constraints (i.e., diffn, geost) are expressed within this first coordinate system. This stands from the fact that resource scheduling constraints like diffn or geost do not support preemption.

The second coordinate system, so called the real coordinate system, considers all timepoints whether they correspond or not to crossable unavailability periods. All temporal constraints (i.e., precedence constraints represented by leq constraints in this model) are expressed with respect to this second coordinate system.

Consequently, each task has a start and an end that are expressed within the virtual coordinate system as well as within the real coordinate system.

Each task, whether it is resumable or not, is passed to the resource scheduling constraints as well as to the precedence constraints. In addition, we represent each non-crossable unavailability period as a fixed task that is also passed to the resource scheduling constraints.

The calendar constraint ensures the link between variables (i.e., the start and the end of the tasks no matter whether they are resumable or not) expressed in these two coordinate systems with respect to the crossable unavailability periods.

We now provide the corresponding detailed model. Given:

1. A set of machines $M = \{m_1, m_2, \ldots, m_p\}$, where each machine has a list of fixed unavailability periods. An unavailability $u_i$ is defined by the following attributes:
   
   (a) The crossable flag $c_i$ tells whether unavailability $u_i$ is crossable ($c_i = 1$) or not ($c_i = 0$).
   
   (b) The machine $r_i$ indicates the machine (i.e., a value in $[1, p]$) to which unavailability $u_i$ corresponds (i.e., since different machines may have different unavailability periods).
   
   (c) The start $s_i$ of the unavailability $u_i$ which indicates the first unavailable timepoint of the unavailability.
   
   (d) The end $e_i$ of the unavailability $u_i$ which gives the last unavailable timepoint of the unavailability.

2. A set of tasks $T = \{t_1, t_2, \ldots, t_n\}$, where each task $t_i$ (with $i \in [1, n]$) has the following attributes which are all domain variables except the resumable flag and the virtual duration:
   
   (a) The resumable flag $r_i$ tells whether task $t_i$ is resumable ($r_i = 1$) or not ($r_i = 0$).
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

(b) The machine $m_i$ indicates the machine (i.e., a value in $[1, p]$) to which task $t_i$ will be assigned.

c) The virtual start $v_s_i$ gives the start of task $t_i$ in the virtual coordinate system.

d) The virtual duration $v_d_i$ corresponds to the duration of task $t_i$ without counting the eventual unavailability periods crossed by task $t_i$.

e) The virtual end $v_e_i$ provides the end of task $t_i$ in the virtual coordinate system. We have that $v_s_i + v_d_i = v_e_i$.

f) The real start $r_s_i$ gives the start of task $t_i$ in the real coordinate system.

g) The real duration $r_d_i$ corresponds to the duration of task $t_i$ including the eventual unavailability periods crossed by task $t_i$. When task $t_i$ is non-resumable (i.e., $r_i = 0$) its real duration is equal to its virtual duration (i.e., $r_d_i = v_d_i$).

h) The real end $r_e_i$ indicates the end of task $t_i$ in the real coordinate system. We have that $r_s_i + r_d_i = r_e_i$.

The link between the virtual starts (respectively virtual ends) and the real starts (respectively real ends) of the different tasks of $T$ is ensured by a calendar(INSTANTS, MACHINES) constraint. More precisely, for each task $t_i$ (with $i \in [1, n]$), no matter whether it is resumable or not, we create the following items for the collection INSTANTS:

\[ \langle \text{machine} - m_i \text{ virtual} - v_s_i \text{ ireal} - r_s_i \text{ flagend} - 0 \rangle, \]
\[ \langle \text{machine} - m_i \text{ virtual} - v_e_i \text{ ireal} - r_e_i \text{ flagend} - 1 \rangle. \]

The first item links the virtual and the real start of task $t_i$, while the second item relates the virtual and real ends. For each machine $m_i$ (with $i \in [1, p]$) and its corresponding list of crossable unavailability periods, denoted $\text{crossable}\_\text{unavailability}_i$, we create the following item of the collection MACHINES:

\[ \langle \text{id} - i \text{ cal} - \text{crossable}\_\text{unavailability}_i \rangle. \]

To express the resource constraint, i.e., the fact that two tasks assigned to the same machine should not overlap in time, we use a geost(2, OBJECTS, SBOXES) constraint. For each task $t_i$ (with $i \in [1, n]$) we create one item for the OBJECTS collection as well as one item for the SBOXES collection:

\[ \langle \text{oid} - i \text{ sid} - i \text{ x} - \langle m_i, v_s_i \rangle \rangle, \]
\[ \langle \text{sid} - i \text{ t} - (0, 0) \text{ l} - (1, v_d_i) \rangle. \]

The first item corresponds to an object with $i$ as unique identifier, with a rectangular shape identifier $i$ and with $m_i, v_s_i$ as the coordinates of its leftmost lower corner. The second item corresponds to a rectangular shape with $i$ as unique identifier, $(0, 0)$ as shift offset with respect to its leftmost lower corner, and $(1, v_d_i)$ as the sizes of the rectangular shape.

Similarly, to express that each task does not overlap a non-crossable unavailability period, we create for each non-crossable unavailability period $i$ one item for the
Chapter 3. Description of the Catalogue

Objects collection as well as one item for the SBoxE collection:

\[
\langle \text{oid} - n + i \quad \text{sid} - n + i \quad \text{x} - \langle r_i, s_i \rangle \rangle,
\langle \text{sid} - n + i \quad \text{t} - \langle 0, 0 \rangle \quad \text{l} - \langle 1, e_i - s_i + 1 \rangle \rangle.
\]

Finally, a precedence constraint between two distinct tasks \( t_i \) and \( t_j \) (with \( i, j \in [1, n] \)) is modelled by an inequality constraint between the real end of task \( t_i \) and the real start of task \( t_j \), namely \( re_i \leq rs_j \). Figure 3.51 provides a toy example of such problem with:

- Four machines, numbered from 1 to 4, where:
  - Machine \( m_1 \) has two crossable unavailability periods respectively corresponding to intervals \([2, 2] \) and \([6, 7] \).
  - Machine \( m_2 \) has two crossable unavailability periods respectively corresponding to intervals \([2, 2] \) and \([6, 7] \), as well as one non-crossable unavailability period corresponding to interval \([3, 3] \).
  - Machine \( m_3 \) has one single non-crossable unavailability corresponding to interval \([6, 8] \).
  - Machine \( m_4 \) has one single crossable unavailability period corresponding to interval \([3, 4] \).

- Five tasks, numbered from 1 to 5, where:
  - Task \( t_1 \) is a non-resumable task that has a virtual duration of 3.
  - Task \( t_2 \) is a resumable task that has a virtual duration of 2.
  - Task \( t_3 \) is a non-resumable task that has a virtual duration of 3.
  - Task \( t_4 \) is a resumable task that has a virtual duration of 5.
  - Task \( t_5 \) is a resumable task that has a virtual duration of 2.

- Finally, (1) all five tasks should not overlap, (2) task \( t_3 \) should precedes task \( t_2 \) and (3) task \( t_1 \) should precedes task \( t_5 \).

A survey on machine scheduling problems with unavailability constraints both in the deterministic and stochastic cases can be found in [345]. Unavailability can have multiple causes such as:

- In the context of production scheduling, machine unavailability corresponds to accepted orders that were already scheduled for a given date. This can typically corresponds to unavailability periods at the beginning of the planning horizon. Preemptive maintenance can also be another cause of machine unavailability.
- In the context of timetabling, unavailability periods may come from work regulation which enforces not to work in a continuous way more than a given limit. Unavailability periods may also come from scheduled meetings during the working day.
In the context of distributed computing where cpu time is donated for performing huge tasks, machines are typically partially available [128].

A constraint for which the same table is shared by several element constraints. Within the context of the case constraint, the same directed acyclic graph can be shared by several tuples of variables. This happen for instance when the case constraint is used for encoding all the transitions of an automaton [34].

Within the context of planning, the idea of reusing the same constraint for encoding the transitions of an automaton\(^\text{16}\) was proposed under the name slice encoding by C. Pralet and G. Verfaillie in [301]. The motivation behind was to avoid to completely unfold the behaviour of the automaton (i.e., the successive triggered transitions) over the full planning horizon. From an implementation point of view, this encoding requires the possibility to reset the domains of the variables to some initial state.

\(^\text{16}\)Even if the original work was not presented in the context of automata, it can be partly reinterpreted as the encoding of an automaton.
3.7.223  **Schur number** ➔  [1 CONS]

- sum_free.

Denotes that a constraint was used for solving Schur problems. Given a non-negative integer \( k \), the Schur number \( S(k) \) is the largest integer \( n \) for which the set \( \{1, 2, \ldots, n\} \) can be partitioned into \( k \) sets \( S_1, S_2, \ldots, S_k \) such that \( \forall i \in [1, k] : i \in S_i \Rightarrow i + i \notin S_i \).

3.7.224  **SLAM problem** ➔  [1 CONS]

- nvector.

Denotes that a constraint was used in the context of the *simultaneous localization and map building* (SLAM) problem. Given a mobile autonomous robot that, for some reason do not has a direct way to perform self-location (i.e., for instance do not has a GPS), the problem is to dynamically build a map and locate its trajectory on that map from a set of partial snapshots of its environment. Within the context of constraint programming this problem is described in [208, 102].

3.7.225  **Sliding cyclic(1) constraint network(1)** ➔  [7 CONS]

- decreasing,
- increasing,
- no_peak,
- no_valley,
- not_all_equal,
- strictly_decreasing,
- strictly_increasing.

A constraint network corresponding to the pattern depicted by Figure 3.52. Circles depict variables, while arcs are represented by a set of variables.

![Hypergraph associated with a sliding cyclic(1) constraint network(1)](image)

Figure 3.52: Hypergraph associated with a sliding cyclic(1) constraint network(1)
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.226  Sliding cyclic(1) constraint network(2)  ➤  [12 CONS]

- change,
- change_continuity,
- cyclic_change,
- cyclic_change_joker,
- deepest_valley,
- highest_peak.

- inflexion,
- length_first_sequence,
- length_last_sequence,
- peak,
- smooth,
- valley.

A constraint network corresponding to the pattern depicted by Figure 3.53. Circles depict variables, while arcs are represented by a set of variables.

Figure 3.53: Hypergraph associated with a sliding cyclic(1) constraint network(2)

3.7.227  Sliding cyclic(1) constraint network(3)  ➤  [3 CONS]

- change,
- change_continuity,

- longest_change.

A constraint network corresponding to the pattern depicted by Figure 3.54. Circles depict variables, while arcs are represented by a set of variables.

Figure 3.54: Hypergraph associated with a sliding cyclic(1) constraint network(3)
3.7.228 Sliding cyclic(2) constraint network(2) ➔ [2 CONS]

- change_pair,
- distance_change.

Figure 3.55: Hypergraph associated with a sliding cyclic(2) constraint network(2)

A constraint network corresponding to the pattern depicted by Figure 3.55. Circles depict variables, while arcs are represented by a set of variables.

3.7.229 Sliding sequence constraint ➔ [17 CONS]

- among_seq,
- arith_sliding,
- cycle_card_on_path,
- elementn,
- pattern,
- relaxed_sliding_sum,
- sliding_card_skip0,
- sliding_distribution,
- size_max_seq_alldifferent,
- size_max_starting_seq_alldifferent,
- sliding_sum,
- sliding_time_window,
- sliding_time_window_from_start,
- sliding_time_window_sum,
- stretch_circuit,
- stretch_path.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

- `stretch_path_partition`.

A constraint enforcing a condition on sliding sequences of domain variables that partially overlap or a constraint computing a quantity from a set of sliding sequences. These sliding sequences can be either initially given or dynamically constructed. In the latter case they can for instance correspond to adjacent vertices of a path that has to be built.

3.7.230 ▲Smallest square for packing consecutive dominoes ▲

[2 CONS]

- `diffn`.
- `geost`.

Find the smallest square $S$ where one can place $n$ rectangles of respective size $1 \times 2, 2 \times 4, \ldots, n \times 2 \cdot n$ so that they do not overlap and so that their borders are parallel to the borders of $S$. Each rectangle can be rotated by 90 degrees. The problem is described in [http://www.stetson.edu/~efriedma/domino/](http://www.stetson.edu/~efriedma/domino/). Figure 3.56 gives a solution for $n = 22$ found by H. Simonis.

![Figure 3.56: A solution to the smallest square for packing consecutive dominoes problem for $n = 22$](image)

Figure 3.56: A solution to the smallest square for packing consecutive dominoes problem for $n = 22$
3.7.231 **Smallest rectangle area** [2 CONS]

- diffn,
- geost.

Denotes that a constraint can be used for finding the smallest rectangle area where one can pack a given set of rectangles (or squares). A first example of such packing problem attributed to S. W. Golomb is to find the smallest square that can contain the set of consecutive squares from $1 \times 1$ up to $n \times n$ so that these squares do not overlap each other. A program using the `diffn` constraint was used to construct such a table for $n \in \{1, 2, \ldots, 25, 27, 29, 30\}$ in [28]. New optimal solutions for this problem were found in [368] for $n = 26, 31, 35$. Figure 3.57 gives the solution found for $n = 35$ by H. Simonis and B. O’Sullivan. Algorithms and lower bounds for solving the same problem can also be respectively found in [85] and in [224].

![Figure 3.57: Smallest square (of size 123) for packing squares of size 1, 2, \ldots, 35](image)

In his paper (i.e., [224]), Richard E. Korf also considers the problem of finding the minimum-area rectangle that can contain the set of consecutive squares from $1 \times 1$ up to $n \times n$ and solve it up to $n = 25$. In 2008 this value was improved up to $n = 27$ by H. Simonis and B. O’Sullivan [368]. Figure 3.58 gives the solution found for $n = 27$ by H. Simonis and B. O’Sullivan.
Figure 3.58: Rectangle with the smallest surface (of size $148 \times 47$) for packing squares of size $1, 2, \ldots, 27$
3.7.232 Smallest square for packing rectangles with distinct sizes

• \textit{diffn},
• \textit{geost}.

Denotes that a constraint can be used for finding the smallest square where one can pack $n$ rectangles for which all the $2 \cdot n$ sizes are distinct integer values. The problem is described in http://www.stetson.edu/~efriedma/mathmagic/0899.html. Figures 3.59, 3.60 and 3.61 present the smallest square (not necessarily optimal) found with \textit{geost} for respectively placing 9, 10, 11, 12, 13 and 14 rectangles of distinct sizes.

Figure 3.59: (Left) Tiling a square of size 24 with 9 rectangles of distinct sizes 1 \times 18, 17 \times 2, 15 \times 3, 4 \times 14, 16 \times 5, 12 \times 6, 7 \times 13, 10 \times 8, 9 \times 11; (Right) Tiling a square of size 28 with 10 rectangles of distinct sizes 1 \times 20, 2 \times 19, 18 \times 3, 4 \times 17, 5 \times 16, 6 \times 15, 7 \times 14, 12 \times 8, 9 \times 13, 10 \times 11.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

Figure 3.60: (Left) Tiling a square of size 32 with 11 rectangles of distinct sizes $1 \times 22$, $21 \times 2$, $3 \times 20$, $18 \times 4$, $19 \times 5$, $16 \times 6$, $7 \times 17$, $8 \times 15$, $14 \times 9$, $13 \times 10$, $12 \times 11$; (Right) Tiling a square of size 37 with 12 rectangles of distinct sizes $1 \times 24$, $2 \times 23$, $3 \times 22$, $4 \times 21$, $5 \times 20$, $6 \times 19$, $7 \times 18$, $8 \times 17$, $9 \times 16$, $15 \times 10$, $11 \times 14$, $12 \times 13$.

Figure 3.61: (Left) Tiling a square of size 41 with 13 rectangles of distinct sizes $1 \times 26$, $2 \times 25$, $3 \times 24$, $4 \times 23$, $5 \times 22$, $21 \times 6$, $20 \times 7$, $19 \times 8$, $18 \times 9$, $17 \times 10$, $11 \times 16$, $15 \times 12$, $13 \times 14$; (Right) Tiling a square of size 46 with 14 rectangles of distinct sizes $1 \times 28$, $2 \times 27$, $3 \times 26$, $4 \times 25$, $5 \times 24$, $6 \times 23$, $7 \times 22$, $8 \times 21$, $20 \times 9$, $19 \times 10$, $18 \times 11$, $17 \times 12$, $16 \times 13$, $15 \times 14$. 
3.7.233 Soft constraint

- open_alldifferent,
- relaxed_sliding_sum,
- soft_alldifferent_ctr,
- soft_alldifferent_var,
- soft_all_equal_max_var,
- soft_all_equal_min_ctr,
- soft_all_equal_min_var,
- soft_cumulative,
- soft_same_interval_var,
- soft_same_modulo_var,
- soft_same_partition_var,
- soft_same_var,
- soft_used_by_interval_var,
- soft_used_by_modulo_var,
- soft_used_by_partition_var,
- soft_used_by_var,
- weighted_partial_alldiff.

A constraint that is a relaxed form of one other constraint.

3.7.234 Sort

- sort,
- sort_permutation.

A constraint involving the notion of sorting in its definition.

3.7.235 Sort based reformulation

- all_min_dist,
- alldifferent,
- alldifferent_consecutive_values,
- alldifferent_cst,
- alldifferent_except_0,
- alldifferent_interval,
- alldifferent_modulo,
- alldifferent_partition,
- alldifferent_same_value,
- allperm,
- consecutive_values,
- derangement,
- disjunctive,
- k_same,
- k_same_interval,
- k_same_modulo,
- k_same_partition,
- k_used_by,
- k_used_by_interval,
- k_used_by_modulo,
- k_used_by_partition,
- permutation,
- same,
- same_interval,
- same_modulo,
- same_partition,
- some_equal,
- used_by,
- used_by_interval,
- used_by_modulo,
- used_by_partition.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

A constraint using the sort constraint in one of its reformulation.

3.7.236 ▶ Sparse functional dependency ➔ [3 CONS]

- case,
- elements_sparse,
- element_sparse.

A constraint that allows for representing a functional dependency between two domain variables, where both variables have a restricted number of values. A variable $X$ is said to functionally determine another variable $Y$ if and only if each potential value of $X$ is associated with exactly one potential value of $Y$.

3.7.237 ▶ Sparse table ➔ [2 CONS]

- element_sparse,
- elements_sparse.

An element constraint for which the table is sparse.

3.7.238 ▶ Sport timetabling ➔ [2 CONS]

- symmetric_alldifferent,
- symmetric_alldifferent_except_0.

A constraint used for creating sports schedules.

3.7.239 ▶ Squared squares ➔ [3 CONS]

- cumulative,
- geost.
- diffn.

A constraint that can be used for modelling the squared squares problem [121] [404] (also called the perfect squared squares problem [136]): a perfect squared square of order $n$ is a square that can be tiled with $n$ smaller squares such that each of the smaller squares has a different integer size. It is called simple if it does not contain a subset of at least two squares, corresponding to a square or to a rectangle. Duijvestijn has shown in 1962 that no instances exist with less than 21 squares [136]. A single solution depicted by Figure 3.62 exists with 21 squares, where the squares have sizes 2, 4, 6, 7, 8, 9, 11, 15, 16, 17, 18, 19, 24, 25, 27, 29, 33, 35, 37, 42, 50 and must be packed into a square of size 112.
Figure 3.62: A simple perfect squared squares of order 21
A catalogue of such simple squared squares of orders 21 through 25 is provided in [80]. The following table contains all the problem instances from the previous catalogue. The different fields respectively give the problem number, the number of squares, the size of the master square and a list of the square sizes. Problems 166 and 167, 168 and 169, 182 and 183 are identical, but have two non-isomorphic solutions. A much bigger table can be found at the following link http://www.squaring.net/.

When the size of the squares is known four constraint programming approach are respectively reported in [1], in [391], in [365], in [36] and in [35].

![Table containing problem instances](http://www.squaring.net/)
CHAPTER 3. DESCRIPTION OF THE CATALOGUE
### 3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

<table>
<thead>
<tr>
<th>Number</th>
<th>3.7.1</th>
<th>3.7.2</th>
<th>3.7.3</th>
<th>3.7.4</th>
<th>3.7.5</th>
<th>3.7.6</th>
<th>3.7.7</th>
<th>3.7.8</th>
<th>3.7.9</th>
<th>3.7.10</th>
<th>3.7.11</th>
<th>3.7.12</th>
<th>3.7.13</th>
<th>3.7.14</th>
<th>3.7.15</th>
<th>3.7.16</th>
<th>3.7.17</th>
<th>3.7.18</th>
<th>3.7.19</th>
<th>3.7.20</th>
<th>3.7.21</th>
<th>3.7.22</th>
<th>3.7.23</th>
<th>3.7.24</th>
<th>3.7.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
<td>2.00</td>
</tr>
<tr>
<td>3</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
<td>3.00</td>
</tr>
<tr>
<td>4</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
<td>4.00</td>
</tr>
<tr>
<td>5</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
<td>5.00</td>
</tr>
<tr>
<td>6</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
</tr>
<tr>
<td>7</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
<td>7.00</td>
</tr>
<tr>
<td>8</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
<td>8.00</td>
</tr>
<tr>
<td>9</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
</tr>
</tbody>
</table>
3.7.240 **Statistics** ➤ [2 CONS]

- deviation,
- spread.

A constraint representing a function in statistics usually used for obtaining a balanced assignment.

3.7.241 **Strip packing** ➤ [2 CONS]

- diffn,
- geost.

A constraint that can be used to model the *strip packing problem*: Given a set of rectangles pack them into an open ended strip of given width in order to minimise the total overall height. Borders of the rectangles to pack should be parallel to the borders of the strip and rectangles should not overlap. Some variants of strip packing allow to rotate rectangles from 90 degrees. Benchmarks with known optima can be obtained from Hopper’s PhD thesis [201].
3.7.242 **Strong articulation point**

A constraint for which the filtering algorithm uses the notion of *strong articulation point*. A *strong articulation point* of a strongly connected digraph $G$ is a vertex such that if we remove it, $G$ is broken into at least two strongly connected components. Figure 3.63 illustrates the notion of strong articulation point on the digraph depicted by part (A). The vertex labelled by 3 is a strong articulation point since its removal creates the three strongly connected components depicted by part (B) (i.e., the first, second and third strongly connected components correspond respectively to the sets of vertices $\{1, 4\}$, $\{2\}$ and $\{5\}$). From an algorithmic point of view, it was shown in [205] how to compute all the strong articulation points of a digraph $G$ in linear time with respect to the number of arcs of $G$.

![Figure 3.63: A connected digraph and its strongly articulation point](image)

3.7.243 **Strong bridge**

A constraint for which the filtering algorithm may use the notion of *strong bridge* (i.e., enforce arcs corresponding to strong bridges to be part of the solution in order to avoid creating too many strongly connected components). A *strong bridge* of a strongly connected digraph $G$ is an arc such that, if we remove it, $G$ is broken into at least two strongly connected components. Figure 3.64 illustrates the notion of strong bridge on the digraph depicted by part (A). The arc from the vertex labelled by 2 to the vertex labelled by 1 is a strong bridge since its removal creates the three strongly connected components depicted by part (B) (i.e., the first, second and third strongly connected components correspond respectively to the sets of vertices $\{1, 3, 4\}$, $\{2\}$ and $\{5\}$). The other strong bridges of the digraph depicted by part (A) are the arcs $1 \rightarrow 3$ and $5 \rightarrow 2$. 
From an algorithmic point of view, it was shown in [205] how to compute all the strong bridges of a digraph $G$ in linear time with respect to the number of arcs of $G$.

Figure 3.64: A connected digraph and one of its strong bridge, the arc $2 \rightarrow 1$

\begin{itemize}
  \item \texttt{atleast_nvalue},
  \item \texttt{atleast_nvector},
  \item \texttt{atmost_nvalue},
  \item \texttt{atmost_nvector},
  \item \texttt{balance_cycle},
  \item \texttt{circuit_cluster},
  \item \texttt{connect_points},
  \item \texttt{cycle},
  \item \texttt{cycle_or_accessibility},
  \item \texttt{cycle_resource},
  \item \texttt{group_skip_isolated_item},
  \item \texttt{increasing_nvalue},
  \item \texttt{nclass},
  \item \texttt{nequivalence},
  \item \texttt{ninterval},
  \item \texttt{npair},
  \item \texttt{nset_of_consecutive_values},
  \item \texttt{nvalue},
  \item \texttt{nvalues},
  \item \texttt{nvalues_except_0},
  \item \texttt{nvector},
  \item \texttt{nvectors},
  \item \texttt{polyomino},
  \item \texttt{soft_alldifferent_var},
  \item \texttt{strongly_connected}.
\end{itemize}

Denotes that a constraint restricts the strongly connected components of its associated final graph. This is usually done by using a graph property like \texttt{MAX_NSCC}, \texttt{MIN_NSCC} or \texttt{NSCC}. 
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.245 ★Subset sum ★

- weighted_partial_alldiff.

Denotes that, by reduction to subset sum, deciding whether a constraint has a solution or not was shown to be NP-hard. The subset sum problem can be described as follows: given a finite set of integers in \( \mathbb{Z}^+ \) and an integer \( s \) in \( \mathbb{Z}^+ \), does any subset sum equal exactly \( s \)?

3.7.246 ★Sudoku ★

- alldifferent.
- \( k\)alldifferent.

A constraint that can be used for modelling the Sudoku puzzle problem. A Sudoku square is an \( 9 \times 9 \) array in which \( 9 \) distinct numbers in \( [1, 9] \) are arranged so that the following two conditions hold:

- Each number occurs once in each row and column.
- The numbers in each major \( 3 \times 3 \) block are distinct.

![Figure 3.65: A partially Sudoku square and its completion](image)

The Sudoku puzzle problem is to complete a partially filled board in order to get a Sudoku square. Part (A) of Figure 3.65 gives a partially filled Sudoku board, while part (B) provides a possible completion.
3.7.247 • Sum ➤ [10 CONS]

- increasing_sum,
- scalar_product,
- sliding_sum,
- sliding_time_window_sum,
- sum,
- sum_ctr,
- sum_of_increments,
- sum_set,
- sum_cubes_ctr,
- sum_squares_ctr.

A constraint involving one or several sums.

3.7.248 • Sweep ➤ [7 CONS]

- cumulatives,
- diffn,
- geost,
- geost_time,
- soft_all_equal_min_var,
- spread,
- visible.

A constraint for which the filtering algorithm may use a sweep algorithm. A sweep algorithm [302, pages 10–11] solves a problem by moving an imaginary object (usually a line, a plane or sometime a point). The object does not move continuously, but only at particular points where we actually do something. A sweep algorithm uses the following two data structures:

- A data structure called the sweep status, which contains information related to the current position of the object that moves,

- A data structure named the event point series, which holds the events to process.

The algorithm initialises the sweep status for the initial position of the imaginary object. Then the object jumps from one event to the next event; each event is handled by updating the status of the sweep.

A first typical application reported in [31] of the idea of sweep within the context of constraint programming is to aggregate several constraints that have two variables in common in order to perform more deduction. Let:

- X and Y be two distinct variables,
- \( C_1(V_{11}, \ldots, V_{1n_1}), \ldots, C_m(V_{m1}, \ldots, V_{mn_m}) \) be a set of \( m \) constraints such that all constraints mention \( X \) and \( Y \).
The sweep algorithm tries to adjust the minimum value of \( X \) wrt. the conjunction of the previous constraints by moving a sweep-line from the minimum value of \( X \) to its maximum value. It accumulates within the sweep-line status the values to be currently removed from the domain of \( Y \). If, for the current position \( \Delta \) of the sweep-line, all values of \( Y \) have to be removed, then the algorithm removes value \( \Delta \) from the domain of \( X \). The events to process correspond to the starts and ends of forbidden two-dimensional regions wrt. constraints \( C_1, \ldots, C_m \) and variables \( X \) and \( Y \). Forbidden regions are a way to represent constraints \( C_1, \ldots, C_m \) that is suited for this sweep algorithm. A forbidden region of the constraint \( C_i \) wrt. the variables \( X \) and \( Y \) is an ordered pair \(([F_x^-, F_x^+], [F_y^-, F_y^+])\) of intervals such that: \( \forall x \in [F_x^-, F_x^+], \forall y \in [F_y^-, F_y^+] : C_i(v_{i1}, \ldots, v_{im}) \) has no solution in which \( X = x \) and \( Y = y \).

Figure 3.66 shows five constraints and their respective forbidden regions (in pink) wrt. two given variables \( X \) and \( Y \) and their domains. The first constraint requires that \( X, Y \) and \( R \) be pairwise distinct. Constraints (B,C) are usual arithmetic constraints. Constraint (D) can be interpreted as requiring that two rectangles of respective origins \((X, Y)\) and \((T, U)\) and sizes \((2, 4)\) and \((3, 2)\) do not overlap. Finally, constraint (E) is a parity constraint of the sum of \( X \) and \( Y \).

We illustrate the use of the sweep algorithm on a concrete example. Assume that we want to find out the minimum value of variable \( X \) wrt. the conjunction of the five constraints that were introduced by Figure 3.66, that is versus the following conjunction

\[ \text{alldifferent(<X,Y,R>)} \]

\[ |X-Y|>2 \]

\[ S \text{ in } 1..6 \]

\[ X+2Y-1<S \]

\[ T \text{ in } 0..2, U \text{ in } 0..3 \]

\[ X+1<T \text{ or } T+2<X \text{ or } Y+3<U \text{ or } U+1<Y \]

\[ (X+Y) \mod 2 = 0 \]
of constraints:

\[
\begin{align*}
X &\in 0..4, Y \in 0..4, R \in 0..9, T \in 0..2, U \in 0..3 \\
\text{alldifferent}((X, Y, R)) &\quad (A) \\
|X - Y| &> 2 &\quad (B) \\
X + 2Y - 1 &< S &\quad (C) \\
X + 1 &< T \lor T + 2 < X \lor Y + 3 < U \lor U + 1 < Y &\quad (D) \\
(X + Y) \mod 2 &\equiv 2 &\quad (E)
\end{align*}
\]

Figure 3.67 shows the content of the sweep-line status (i.e., the forbidden values for \(Y\) according the current position of the sweep-line) for different positions of the sweep-line. More precisely, the sweep-line status can be viewed as an array (see the rightmost part of Figure 3.67) which records for each possible value of \(Y\) the number of forbidden regions that currently intersect the sweep-line (see the leftmost part of Figure 3.67 where these forbidden regions are coloured in red). The smallest possible value of \(X\) is 4, since this is the first position of the sweep-line where the sweep-line status contains a value of \(Y\) which is not forbidden (i.e., \(X = 4, Y = 0\) is not covered by any forbidden region).

A second similar application of the idea of sweep in the context of the cardinality operator [394], where all constraints have at least two variables in common, is reported in [30]. As before, each constraint \(C\) of the cardinality operator is defined by its forbidden regions wrt. a pair of variables \((X, Y)\) that occur in every constraint. In addition to that, a constraint \(C\) is also defined by its safe regions, where a safe region is the set of assignments to the pair \((X, Y)\) located in a rectangle such that the constraint always holds, no matter which values are taken by the other variables of \(C\). Then the extended sweep algorithm filters the pair of variables \((X, Y)\) right from the beginning according to the minimum and maximum number of constraints of the cardinality operator that have to hold.

A third typical application reported in [36] and in [93] of the idea of sweep within the context of multi-dimensional placement problems (see for instance the \textit{diffn} and the \textit{geost} constraints) for filtering each coordinate of the origin of an object \(o\) to place is as follows. To adjust the minimum (respectively maximum) value of a coordinate of the origin we perform a recursive traversal of the placement space in increasing (respectively decreasing) lexicographic order and skips infeasible points that are located in a multi-dimensional forbidden set. Each multi-dimensional forbidden set is computed from a constraint where object \(o\) occurs (for instance a non-overlapping constraint in the context of the \textit{diffn} and the \textit{geost} constraints).
3.7. **KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS**

![Figure 3.67: Sweep-line status while sweeping through the values of X](image)

<table>
<thead>
<tr>
<th>CONSTRAINTS</th>
<th>SWEEP-LINE STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A)</td>
<td>0</td>
</tr>
<tr>
<td>(B)</td>
<td>1</td>
</tr>
<tr>
<td>(C)</td>
<td>2</td>
</tr>
<tr>
<td>(D)</td>
<td>3</td>
</tr>
<tr>
<td>(E)</td>
<td>4</td>
</tr>
</tbody>
</table>

X=0

0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4

X=1

0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4

X=2

0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4

X=3

0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4

X=4

0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
3.7.249 **Symmetric** ➔ [9 CONS]

- all.differ.from.at.least.k.pos,
- all.incomparable,
- bipartite,
- clique,
- connect.points,
- connected,
- inverse.within.range,
- proper_forest,
- symmetric.

Denotes that a constraint is defined by a graph constraint for which the final graph is symmetric. A digraph is symmetric if and only if, if there is an arc from a vertex \( u \) to a vertex \( v \), there is also an arc from \( v \) to \( u \).

3.7.250 **Symmetry** ➔ [22 CONS]

- allperm,
- increasing.global.cardinality,
- increasing.nvalue,
- increasing.sum,
- int.value.precede,
- int.value.precede.chain,
- geost,
- lex2,
- lex.between,
- lex.chain.less,
- lex.chain.lesseq,
- lex.greater,
- lex.greatereq,
- lex.less,
- lex.lesseq,
- lex.lesseq.allperm,
- ordered.atleast.nvector,
- ordered.atmost.nvector,
- ordered.nvector,
- set.value.precede,
- strict_lex2,
- subgraph.isomorphism.

A constraint that can be used for breaking certain types of symmetries (i.e., allperm, int_value.precede, ..., strict_lex2) or for identifying certain symmetries (i.e., subgraph.isomorphism).
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.251 System of constraints

- all_differ_from_at_least_k_pos (system of differ_from_at_least_k_pos),
- all_incomparable (system of incomparable),
- alldifferent (system of neq),
- allperm (system of lex_less eq_allperm),
- among_seq (system of among low_up),
- colored_matrix (system of global_cardinality),
- elements (system of elem or of element sharing the same table),
- elements_sparse (system of element_sparse sharing the same table),
- global_cardinality (system of among),
- k_alldifferent (system of alldifferent),
- k_disjoint (system of disjoint),
- k_same (system of same),
- k_same_interval (system of same_interval),
- k_same_modulo (system of same_modulo),
- k_same_partition (system of same_partition),
- k_used_by (system of used_by),
- k_used_by_interval (system of used_by_interval),
- k_used_by_modulo (system of used_by_modulo),
- k_used_by_partition (system of used_by_partition),
- lex2 (system of lex_chain_less_eq),
- lex_between (system of lex_less_eq),
- lex_chain_less_eq (system of lex_less_eq),
- lex_chain_less (system of lex_less),
- lex_alldifferent (system of lex_alldifferent),
- sliding_distribution (system of global_cardinality_low_up),
- sliding_sum (system of sum_ctr),
- strict_lex2 (system of lex_chain_less).

Denotes that a constraint is defined as the conjunction of several identical global constraints that have some variables in common.
3.7.252  ▼Table ➤

- `elem`, `elem_from_to`, `element`, `elementn`, `element_greatereq`, `element_leasseq`, `element_product`, `element_sparse`, `elements`, `elements_alldifferent`, `elements_sparse`, `ith_pos_different_from_0`, `next_element`, `next_greater_element`, `stage_element`.

A constraint that allows for representing the access to an element of a table.

3.7.253  ▼Temporal constraint ➤


A constraint involving the notion of time.

3.7.254  ▼Ternary constraint ➤

- `distance`, `element_matrix`, `gcd`, `power`.

A constraint involving only three variables.
3.7.255 **Timetabling constraint** ➔

- change,
- change_continuity,
- change_pair,
- change_partition,
- circular_change,
- colored_matrix,
- cumulatives,
- cyclic_change,
- cyclic_change_joker,
- diffn,
- geost,
- geost_time,
- group,
- group_skip_isolated_item,
- interval_and_count,
- interval_and_sum,
- longest_change,
- pattern,
- period,
- period_except_0,
- shift,
- sliding_card_skip0,
- smooth,
- stretch_circuit,
- stretch_path,
- stretch_path_partition,
- symmetric_alldifferent,
- symmetric_alldifferent_except_0,
- symmetric_cardinality,
- symmetric_gcc,
- track.

A constraint that can occur in timetabling problems.

3.7.256 **Time window** ➔

- sliding_time_window_sum.

A constraint involving one or several date ranges.

3.7.257 **Touch** ➔

- orths_are_connected,
- two_orth_are_in_contact.

A constraint enforcing that some orthotopes touch each other (see **Contact**).
3.7.258 **Tree** ➤

- balance_path,
- balance_tree,
- binary_tree,
- path,
- proper_forest,
- stable_compatibility,
- tree,
- tree_range,
- tree_resource.

According to the context, the keyword **tree** has the following meaning:

- In the context of a **digraph**, a constraint that partitions the vertices of a given initial digraph and that keeps one single successor for each vertex so that each partition corresponds to one tree. Each vertex points to its father or to itself if it corresponds to the root of a tree.

- In the context of an **undirected graph** a constraint that partitions the vertices of a given initial undirected graph in a set of connected components with no cycles.

3.7.259 **Tuple** ➤

- in_relation,
- vec_eq_tuple.

A constraint involving a **tuple**. A **tuple** is an element of a **relation**, where a **relation** is a subset of the product of several finite sets.

3.7.260 **Two-dimensional orthogonal packing** ➤

- diffn,
- geost.

A constraint that can be used to model the **two-dimensional orthogonal packing problem**. Given a set of rectangles pack them into a rectangular placement space. Borders of the rectangles should be parallel to the borders of the placement space and rectangles should not overlap. Some variants of strip packing allow to rotate rectangles from 90 degrees. Benchmarks can be obtained from a generator described in the following paper [111].
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.261 Unary constraint

- in,
- in_interval,
- in_intervals,
- not_in,
- sum_free.

A constraint involving only one variable.

3.7.262 Undirected graph

- proper_forest,
- tour.

A constraint that deals with an undirected graph. An undirected graph is a graph whose edges consist of unordered pairs of vertices.

3.7.263 Value constraint

- all_equal,
- all_min_dist,
- alldifferent,
- alldifferent_cst,
- alldifferent_consecutive_values,
- alldifferent_except_0,
- alldifferent_interval,
- alldifferent_modulo,
- alldifferent_on_intersection,
- alldifferent_partition,
- among,
- among_diff_0,
- among_interval,
- among_low_up,
- arith,
- arith_or,
- atleast,
- atmost,
- balance,
- balance_interval,
- balance_modulo,
- balance_partition,
- cardinality_atleast,
- cardinality_atmost,
- cardinality_atmost_partition,
- consecutive_values,
- count,
- counts,
- differ_from_at_least_k_pos,
- discrepancy,
- disjoint,
- domain,
- exactly,
- global_cardinality,
- global_cardinality_low_up,
- global_cardinality_low_up_no_loop,
- global_cardinality_no_loop,
- in,
- in_interval,
- in_interval_reified,
- in_intervals,
A constraint that puts a restriction on how values can be assigned to usually one or several collections of variables, or possibly one or two variables. These variables usually correspond to domain variables but can sometimes be set variables.

3.7.264 ▼Value partitioning constraint ➾

- at least n value,
- at least n vector,
- at most n value,
- at most n vector,
- increasing n value,
- n class,
- n equivalence,
- n interval,
- n pair,
- n value,
- n values,
- n values except 0,
- n vector,
- n vectors.

A constraint involving a partitioning of values in its definition.
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.265 ▼ Value precedence ➤ [3 CONS]
- int_value_precede,
- int_value_precede_chain,
- set_value_precede.

A constraint that allows for expressing symmetries between values that are assigned to variables.

3.7.266 ▼ Variable-based violation measure ➤ [11 CONS]
- soft_alldifferent_var,
- soft_all_equal_max_var,
- soft_all_equal_min_var,
- soft_same_interval_var,
- soft_same_modulo_var,
- soft_same_partition_var,
- soft_same_var,
- soft_used_by_interval_var,
- soft_used_by_modulo_var,
- soft_used_by_partition_var,
- soft_used_by_var.

A soft constraint for which the violation cost is the minimum number of variables to unassign in order to get back to a solution.

3.7.267 ▼ Variable indexing ➤ [7 CONS]
- elem,
- elem_from_to,
- element,
- element_greatereq,
- element_lesseq,
- element_sparse,
- indexed_sum.

A constraint where one or several variables are used as an index into an array.

3.7.268 ▼ Variable subscript ➤ [7 CONS]
- elem,
- elem_from_to,
- element,
- element_greatereq,
- element_lesseq,
- element_product,
- indexed_sum.

A constraint that can be used to model one or several variables that have a variable subscript.
3.7.269 ▼ Vector ➔

- all_differ_from_at_least_k_pos,
- all_incomparable,
- allperm,
- atleast_nvector,
- atmost_nvector,
- change_vectors,
- cond_lex_cost,
- cond_lex_greater,
- cond_lex_greatereq,
- cond_lex_less,
- cond_lex_leq,
- differ_from_at_least_k_pos,
- incomparable,
- lex_alldifferent,
- lex_between,
- lex_chain_less,
- lex_chain_less_eq,
- lex_different,
- lex_equal,
- lex_greater,
- lex_greater_eq,
- lex_less,
- lex_less_eq,
- lex_less_eq_allperm,
- nvector,
- nvectors,
- ordered_atleast_nvector,
- ordered_atmost_nvector,
- ordered_nvectors,
- period_vectors.

Denotes that one (or more) argument of a constraint corresponds to a collection of vectors that all have the same number of components.

3.7.270 ▼ Vpartition ➔

- group.

Denotes that a constraint is defined by two graph constraints $C_1$ and $C_2$ such that:
- The two graph constraints have the same initial graph $G_i$,
- Each vertex of the initial graph $G_i$ belongs to exactly one of the final graphs associated with $C_1$ and $C_2$. 
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

3.7.271 ▼Weighted assignment ➔ [4 CONS]

- global_cardinality_with_costs,
- minimum_weight_alldifferent,
- sum_of_weights_of_distinct_values,
- weighted_partial_alldiff.

A constraint expressing an assignment problem such that a cost can be computed from each solution.

3.7.272 ▼Workload covering ➔ [1 CONS]

- cumulatives.

A constraint that can be used for modelling problems where a first set of tasks $T_1$ has to cover a second set of tasks $T_2$. Each task of $T_1$ and $T_2$ is defined by an origin, a duration and a height. At each point in time $t$ the sum of the heights of the tasks of the first set $T_1$ that overlap $t$ has to be greater than or equal to the sum of the heights of the tasks of the second set $T_2$ that also overlap $t$.

3.7.273 ▼Zebra puzzle ➔ [4 CONS]

- alldifferent,
- elem,
- element,
- inverse.

A constraint that can be used for modelling the zebra puzzle problem. Here is the first known publication of that puzzle quoted in italic from Life International, December 17, 1962:

1. There are five houses.
2. The Englishman lives in the red house.
3. The Spaniard owns the dog.
4. Coffee is drunk in the green house.
5. The Ukrainian drinks tea.
6. The green house is immediately to the right of the ivory house.
7. The Old Gold smoker owns snails.
8. Kools are smoked in the yellow house.
9. Milk is drunk in the middle house.

10. The Norwegian lives in the first house.

11. The man who smokes Chesterfields lives in the house next to the man with the fox.

12. Kools are smoked in the house next to the house where the horse is kept.

13. The Lucky Strike smoker drinks orange juice.


15. The Norwegian lives next to the blue house.

Now, who drinks water? Who owns the zebra?

In the interest of clarity, it must be added that each of the five houses is painted a different color, and their inhabitants are of different national extractions, own different pets, drink different beverages and smoke different brands of American cigarettes. In statement 6, right refers to the reader’s right.

A first model involves element constraints with variables in their tables (i.e., the table of an element constraint corresponds to its second argument). It consists of creating for each house \( i \) \((1 \leq i \leq 5)\) five variables \( C_i, N_i, A_i, D_i, B_i \) respectively corresponding to the color of house \( i \), the nationality of the person leaving in house \( i \), the preferred pet of the person leaving in house \( i \), the preferred beverage of the person leaving in house \( i \), the preferred brand of American cigarettes of the person leaving in house \( i \). We first state the following five alldifferent constraints on these variables for expressing that colors, nationalities, pets, beverages, and brands of American cigarettes are distinct:

- \( \text{alldifferent}(\langle C_1, C_2, C_3, C_4, C_5 \rangle) \),
- \( \text{alldifferent}(\langle N_1, N_2, N_3, N_4, N_5 \rangle) \),
- \( \text{alldifferent}(\langle A_1, A_2, A_3, A_4, A_5 \rangle) \),
- \( \text{alldifferent}(\langle D_1, D_2, D_3, D_4, D_5 \rangle) \),
- \( \text{alldifferent}(\langle B_1, B_2, B_3, B_4, B_5 \rangle) \).

Now observe that most statements link two specific attributes (e.g., The Englishman lives in the red house). Consequently, in order to ease the encoding of such statements in term of constraints, we will first create for each attribute a variable that indicates the house where an attribute occurs. For instance, for the statement The Englishman lives in the red house we will create two variables which respectively indicate in which house the Englishman lives and which house is red. We now create all the variables attached to each class of attributes.

For each possible color \( c \in \{ \text{red, green, ivory, yellow, blue} \} \) we create a variable \( I_c \) that corresponds to the index of the house having this color. For each variable \( I_c \), an element constraint links it to the variables \( C_1, C_2, C_3, C_4, C_5 \) giving the colour of each house:
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

- Red = 1, Green = 2, Ivory = 3, Yellow = 4, Blue = 5,
- \element(I_{red}, \langle C_1, C_2, C_3, C_4, C_5 \rangle, \text{Red}),
- \element(I_{green}, \langle C_1, C_2, C_3, C_4, C_5 \rangle, \text{Green}),
- \element(I_{ivory}, \langle C_1, C_2, C_3, C_4, C_5 \rangle, \text{Ivory}),
- \element(I_{yellow}, \langle C_1, C_2, C_3, C_4, C_5 \rangle, \text{Yellow}),
- \element(I_{blue}, \langle C_1, C_2, C_3, C_4, C_5 \rangle, \text{Blue}).

Note that we can replace the five previous \element constraints by the following \inverse constraint:

\begin{equation*}
\begin{pmatrix}
\text{index} - 1 & \text{succ} - C_1 & \text{pred} - I_{red}, \\
\text{index} - 2 & \text{succ} - C_2 & \text{pred} - I_{green}, \\
\text{index} - 3 & \text{succ} - C_3 & \text{pred} - I_{ivory}, \\
\text{index} - 4 & \text{succ} - C_4 & \text{pred} - I_{yellow}, \\
\text{index} - 5 & \text{succ} - C_5 & \text{pred} - I_{blue}
\end{pmatrix}
\end{equation*}

For each possible nationality \( n \in \{ \text{englishman, spaniard, ukrainian, norwegian, japanese} \} \) we create a variable \( I_n \) that corresponds to the index of the house where the person with this nationality lives. For each variable \( I_n \), an \element constraint links it to the variables \( N_1, N_2, N_3, N_4, N_5 \) giving the nationality associated with each house:

- Englishman = 1, Spaniard = 2, Ukrainian = 3, Norwegian = 4, Japanese = 5,
- \element(I_{englishman}, \langle N_1, N_2, N_3, N_4, N_5 \rangle, \text{Englishman}),
- \element(I_{spaniard}, \langle N_1, N_2, N_3, N_4, N_5 \rangle, \text{Spaniard}),
- \element(I_{ukrainian}, \langle N_1, N_2, N_3, N_4, N_5 \rangle, \text{Ukrainian}),
- \element(I_{norwegian}, \langle N_1, N_2, N_3, N_4, N_5 \rangle, \text{Norwegian}),
- \element(I_{japanese}, \langle N_1, N_2, N_3, N_4, N_5 \rangle, \text{Japanese}).

Again we can replace the five previous \element constraints by the following \inverse constraint:

\begin{equation*}
\begin{pmatrix}
\text{index} - 1 & \text{succ} - N_1 & \text{pred} - I_{englishman}, \\
\text{index} - 2 & \text{succ} - N_2 & \text{pred} - I_{spaniard}, \\
\text{index} - 3 & \text{succ} - N_3 & \text{pred} - I_{ukrainian}, \\
\text{index} - 4 & \text{succ} - N_4 & \text{pred} - I_{norwegian}, \\
\text{index} - 5 & \text{succ} - N_5 & \text{pred} - I_{japanese}
\end{pmatrix}
\end{equation*}

For each possible preferred pet \( a \in \{ \text{dog, snail, fox, horse, zebra} \} \) we create a variable \( I_a \) that corresponds to the index of the house where the person that prefers this pet lives. For each variable \( I_a \), an \element constraint links it to the variables \( A_1, A_2, A_3, A_4, A_5 \) giving the preferred pet of each house:
• Dog = 1, Snail = 2, Fox = 3, Horse = 4, Zebra = 5,

• element(\(I_{dog}, \langle A_1, A_2, A_3, A_4, A_5 \rangle, \text{Dog} \))

• element(\(I_{snail}, \langle A_1, A_2, A_3, A_4, A_5 \rangle, \text{Snail} \))

• element(\(I_{fox}, \langle A_1, A_2, A_3, A_4, A_5 \rangle, \text{Fox} \))

• element(\(I_{horse}, \langle A_1, A_2, A_3, A_4, A_5 \rangle, \text{Horse} \))

• element(\(I_{zebra}, \langle A_1, A_2, A_3, A_4, A_5 \rangle, \text{Zebra} \)).

Again we can replace the five previous \textit{element} constraints by the following \textit{inverse} constraint:

\[
\begin{pmatrix}
\text{index} - 1 & \text{succ} - A_1 & \text{pred} - I_{dog}, \\
\text{index} - 2 & \text{succ} - A_2 & \text{pred} - I_{snail}, \\
\text{index} - 3 & \text{succ} - A_3 & \text{pred} - I_{fox}, \\
\text{index} - 4 & \text{succ} - A_4 & \text{pred} - I_{horse}, \\
\text{index} - 5 & \text{succ} - A_5 & \text{pred} - I_{zebra}
\end{pmatrix}
\]

For each possible preferred beverage \(d \in \{\text{coffee, tea, milk, orange juice, water} \} \)
we create a variable \(I_d\) that corresponds to the index of the house where the person that
prefers this beverage lives. For each variable \(I_d\), an \textit{element} constraint links it to the
variables \(D_1, D_2, D_3, D_4, D_5\) giving the preferred beverage of each house:

• Coffee = 1, Tea = 2, Milk = 3, Orange juice = 4, Water = 5,

• element(\(I_{coffee}, \langle D_1, D_2, D_3, D_4, D_5 \rangle, \text{Coffee} \))

• element(\(I_{tea}, \langle D_1, D_2, D_3, D_4, D_5 \rangle, \text{Tea} \))

• element(\(I_{milk}, \langle D_1, D_2, D_3, D_4, D_5 \rangle, \text{Milk} \))

• element(\(I_{orange juice}, \langle D_1, D_2, D_3, D_4, D_5 \rangle, \text{Orange juice} \))

• element(\(I_{water}, \langle D_1, D_2, D_3, D_4, D_5 \rangle, \text{Water} \)).

Again we can replace the five previous \textit{element} constraints by the following \textit{inverse} constraint:

\[
\begin{pmatrix}
\text{index} - 1 & \text{succ} - D_1 & \text{pred} - I_{coffee}, \\
\text{index} - 2 & \text{succ} - D_2 & \text{pred} - I_{tea}, \\
\text{index} - 3 & \text{succ} - D_3 & \text{pred} - I_{milk}, \\
\text{index} - 4 & \text{succ} - D_4 & \text{pred} - I_{orange juice}, \\
\text{index} - 5 & \text{succ} - D_5 & \text{pred} - I_{water}
\end{pmatrix}
\]

For each possible preferred brand of American cigarettes \(b \in \{\text{old gold, kool, chesterfield, lucky strike, parliament} \} \)
we create a variable \(I_b\) that corresponds to the
index of the house where the person that prefers this brand lives. For each variable \(I_b\), an \textit{element} constraint links it to the variables \(B_1, B_2, B_3, B_4, B_5\) giving the preferred brand of American cigarettes of each house:
3.7. KEYWORDS ATTACHED TO THE GLOBAL CONSTRAINTS

- \( \text{element}(I_{old.gold}, \langle B_1, B_2, B_3, B_4, B_5 \rangle, \text{Old gold}) \),
- \( \text{element}(I_{kool}, \langle B_1, B_2, B_3, B_4, B_5 \rangle, \text{Kool}) \),
- \( \text{element}(I_{chesterfield}, \langle B_1, B_2, B_3, B_4, B_5 \rangle, \text{Chesterfield}) \),
- \( \text{element}(I_{lucky.strike}, \langle B_1, B_2, B_3, B_4, B_5 \rangle, \text{Lucky strike}) \),
- \( \text{element}(I_{parliament}, \langle B_1, B_2, B_3, B_4, B_5 \rangle, \text{Parliament}) \).

Again we can replace the five previous *element* constraints by the following *inverse* constraint:

\[
\begin{pmatrix}
\text{index} - 1 & \text{succ} - B_1 & \text{pred} - I_{old.gold}, \\
\text{index} - 2 & \text{succ} - B_2 & \text{pred} - I_{kool}, \\
\text{index} - 3 & \text{succ} - B_3 & \text{pred} - I_{chesterfield}, \\
\text{index} - 4 & \text{succ} - B_4 & \text{pred} - I_{lucky.strike}, \\
\text{index} - 5 & \text{succ} - B_5 & \text{pred} - I_{parliament}
\end{pmatrix}
\]

Finally we state one constraint for each statement from 2 to 15:

- \( I_{\text{englishman}} = I_{\text{red}} \) (the Englishman lives in the red house).
- \( I_{\text{spaniard}} = I_{\text{dog}} \) (the Spaniard owns the dog).
- \( I_{\text{coffee}} = I_{\text{green}} \) (coffee is drunk in the green house).
- \( I_{\text{ukrainian}} = I_{\text{tea}} \) (the Ukrainian drinks tea).
- \( I_{\text{green}} = I_{\text{ivory}} + 1 \) (the green house is immediately to the right of the ivory house).
- \( I_{\text{old.gold}} = I_{\text{snail}} \) (the Old Gold smoker owns snails).
- \( I_{\text{kool}} = I_{\text{yellow}} \) (kools are smoked in the yellow house).
- \( I_{\text{milk}} = 3 \) (milk is drunk in the middle house).
- \( I_{\text{norwegian}} = 1 \) (the Norwegian lives in the first house).
- \( |I_{\text{chesterfield}} - I_{\text{fox}}| = 1 \) (the man who smokes Chesterfields lives in the house next to the man with the fox).
- \( |I_{\text{kool}} - I_{\text{horse}}| = 1 \) (kools are smoked in the house next to the house where the horse is kept).
- \( I_{\text{lucky.strike}} = I_{\text{orange.juice}} \) (the Lucky Strike smoker drinks orange juice).
- \( I_{\text{japanese}} = I_{\text{parliament}} \) (the Japanese smokes Parliaments).
- \( |I_{\text{norwegian}} - I_{\text{blue}}| = 1 \) (the Norwegian lives next to the blue house).
Now note that variables $C_i$, $N_i$, $A_i$, $D_i$, $B_i$ ($1 \leq i \leq 5$) do not occur at all within the constraints encoding statements 2 to 15. Consequently they can be removed, as long as we replace the five `alldifferent` constraints on these variables by the following `alldifferent` constraints:

- `alldifferent(⟨I_red, I_green, I_ivory, I_yellow, I_blue⟩),`
- `alldifferent(⟨I_englishman, I_spaniard, I_ukrainian, I_norwegian, I_japanese⟩),`
- `alldifferent(⟨I_dog, I_snail, I_fox, I_horse, I_zebra⟩),`
- `alldifferent(⟨I_coffee, I_tea, I_milk, I_orange_juice, I_water⟩),`
- `alldifferent(⟨I_old_gold, I_kool, I_chesterfield, I_lucky_strike, I_parliament⟩).`

In our experience, when confronted for the first time to this puzzle, a lot of people come up with the model that associates to each house $i$ ($1 \leq i \leq 5$) five variables $C_i$, $N_i$, $A_i$, $D_i$, $B_i$ that describe the attributes of the person living in house $i$. However it is difficult to directly express the constraints according to these variables and the second model which associates to each attribute a variable that gives the corresponding house is more convenient for expressing the constraints.

3.7.274 **Zero-duration task**

A resource scheduling constraint that accepts tasks which can potentially have a duration equal to zero. Zero-duration tasks can be used for modelling over-constrained resource scheduling problems where, due to some resource limitations, some tasks have to be discarded. This can be expressed by creating for each task $i$ a duration variable $D_i$ with values 0 and $d_i$ in its initial domain, where $d_i$ is the effective duration of task $i$ when it is not discarded. Then, depending on the relaxation cost $C_i$ associated with the fact that task $i$ is not considered, a reified constraint of the form $D_i = 0 \iff C_i = \alpha_i$ ($\alpha_i > 0$) is created. The initial domain of the cost variable $C_i$ is set to 0 and $\alpha_i$, where $\alpha_i$ is the cost associated with the decision of discarding task $i$. Then all the relaxation costs associated with the different tasks have to be aggregated together, i.e., typically by taking the sum or the maximum of the relaxation costs of the different tasks. On the one hand, the overall advantage of the approach is that it does not require developing any specific algorithm. On the other hand, the disadvantage is the lack of bounds on the overall relaxation cost that can sometimes be compensated by a specific enumeration heuristics.
Chapter 4

Further Topics

Contents

4.1 Differences from the 2000 report .......................... 344
4.2 Differences from the 2005 report ............................... 346
4.3 Graph invariants .................................................. 347
  4.3.1 Graph classes .................................................. 347
  4.3.2 Format of an invariant ......................................... 348
  4.3.3 Using the database of invariants .............................. 349
  4.3.4 The database of graph invariants ............................ 350
Graph invariants involving one parameter of a final graph .......... 354
Graph invariants involving two parameters of a final graph .......... 356
Graph invariants involving three parameters of a final graph ........ 365
Graph invariants involving four parameters of a final graph ........ 379
Graph invariants involving five parameters of a final graph ........ 385
Graph invariants relating two parameters of two final graphs ....... 386
Graph invariants relating three parameters of two final graphs ...... 388
Graph invariants relating four parameters of two final graphs ...... 391
Graph invariants relating five parameters of two final graphs ...... 393
Graph invariants relating six parameters of two final graphs ...... 398
4.4 The electronic version of the catalogue ......................... 399
  4.4.1 Prolog facts describing a constraint .......................... 399
  4.4.2 XML schema associated with a global constraint ............ 404
Related work ......................................................... 404
Key features ......................................................... 404
Structure of schema ................................................ 405
Model ............................................................... 405
Variables ........................................................... 405
constraints ........................................................ 406
collection .......................................................... 406
4.1 Differences from the 2000 report

This section summarises the main differences with the SICS report [24] as well as of the corresponding article [25]. The main differences are listed below:

- We have both simplified and extended the way to generate the vertices of the initial graph and we have introduced a new way of defining set of vertices. We have also removed the CLIQUE(MAX) set of vertices generator since it cannot in general be evaluated in polynomial time. Therefore, we have modified the description of the constraints assign_and_counts, assign_and_nvalues, interval_and_count, interval_and_sum, bin_packing, cumulative, cumulatives, coloured_cumulative, coloured_cumulatives, cumulative_two_d, which all used this feature.

- We have introduced the new arc generators PATH_1 and PATH_N, which allow for specifying an n-ary constraint for which n is not fixed. The size_max_starting_seq_alldifferent and the size_max_seq_alldifferent are examples of global constraints that use these arc generators in order to generate a set of sliding alldifferent constraints.

- In addition to traditional domain variables we have introduced float, set and multiset variables as well as several global constraints mentioning float and set variables (see for instance the choquet [202] and the alldifferent_between_sets constraints). This decision was initially motivated by the fact that several constraint systems and articles mention global constraints dealing with these types of variables. Later on, we realised that set variables also greatly simplify the interface of existing global constraints. This was especially true for those global constraints that explicitly deal with a graph, like clique or cutset. In this context, using a set variable for catching the successors of a vertex is quite natural. This is especially true when a vertex of the final graph can have more than one successor since it allows for avoiding a lot of 0-1 variables.

- We have introduced the possibility of using more than one graph constraint for defining a given global constraint (see for instance the cumulative or the sort constraints). Therefore we have removed the notion of dual graph, which was initially introduced in the original report. In this context, we now use two graph constraints (see for instance change_continuity).
4.1. DIFFERENCES FROM THE 2000 REPORT

On the one hand, we have introduced the following new graph parameters:

- \texttt{MAX.DRG},
- \texttt{MAX.OD},
- \texttt{MIN.DRG},
- \texttt{MIN.ID},
- \texttt{MIN.OD},
- \texttt{NTREE},
- \texttt{PATH.FROM_TO},
- \texttt{PROD},
- \texttt{RANGE},
- \texttt{RANGE.DRG},
- \texttt{RANGE.NCC},
- \texttt{SUM},
- \texttt{SUM.WEIGHT_ARC}.

On the other hand, we have removed the following graph parameters:

- \texttt{NCC(COMP, val)},
- \texttt{NSCC(COMP, val)},
- \texttt{NTREE(ATTR, COMP, val)},
- \texttt{NSOURCE.EQ.NSINK},
- \texttt{NSOURCE.GREATEREQ.NSINK}.

Finally, \texttt{MAX_IN.DEGREE} has been renamed \texttt{MAX.ID}.

We have introduced an iterator over the items of a collection in order to specify in a generic way a set of similar elementary constraints or a set of similar graph properties. This was required for describing some global constraints such as \texttt{global.cardinality}, \texttt{cycle.resource} or \texttt{stretch}. All these global constraints mention a condition involving some limit depending on the specific values that are effectively used. For instance the \texttt{global.cardinality} constraint forces each value \( v \) to be respectively used at least \( \texttt{atleast} v \), and at most \( \texttt{atmost} v \) times. This iterator was also necessary in the context of graph covering constraints where one wants to cover a digraph with some patterns. Each pattern consists of one resource and several tasks. One can now attach specific constraints to the different resources. Both the \texttt{cycle.resource} and the \texttt{tree.resource} constraints illustrate this point.

We have added some standard existing global constraints that were obviously missing from the previous report. This was for instance the case of the \texttt{element} constraint.
• In order to make clear the notion of family of global constraints we have computed for each global constraint a signature, which summarises its structure. Each signature was inserted into the index so that one can retrieve all the global constraints sharing the same structure.

• We have generalised some existing global constraints. For instance the change_pair constraint extends the change constraint. Finally we have introduced some novel global constraints like disjoint_tasks or symmetric_gcc.

• We have defined the rules for specifying arc constraints.

4.2 Differences from the 2005 report

The second edition has more than 1300 pages of new content. The slots describing explicitly the meaning of a global constraint (e.g., the slots Graph model and Automaton) were moved to the last part of the description. This was motivated by the fact that most users want first to get the informal description of a global constraint (e.g., the slots Purpose and Example). Effort was not only devoted to the introduction of new constraints but also to a better description of multiple aspects like:

• The slot Symmetries describes a set of mapping that preserve the solution of a constraint (see Section 2.1.5).

• The slot Reformulation provides reformulation of a global constraint as a conjunction of constraints (see Section 2.4).

• The slot Systems gives links to concrete constraint systems.

• The slots See also and Keywords were redesigned in order to respectively indicate why we point to a given constraint (see Section 2.5) and to group together keywords by meta-keywords (see Section 3.6).

• In addition to the slots Graph model and Automaton that respectively describe the meaning of a global constraint in terms of graph properties and automaton, we have introduced the slot Logic in order to describe some geometrical constraints with first order formulae (see keyword Logic).

• Finally, an evaluator was provided for most global constraints.
4.3 Graph invariants

Within the scope of the graph-based description this section shows how to use implied constraints, which are systematically linked to the description of a global constraint. This usually occurs in the following context:

- Quite often, it happens that one wants to enforce the final graph to satisfy more than one graph property. In this context, these graph properties involve several graph parameters that cannot vary independently.

**EXAMPLE:** As a practical example, consider the group constraint and its first graph constraint. It involves the four graph parameters $NCC, \text{MIN}_NCC, \text{MAX}_NCC$ and $\text{NVERTEX}$, which respectively correspond to the number of connected components, the number of vertices of the smallest connected component, the number of vertices of the largest connected component and the number of vertices of the final graph. In this example the number of connected components of the final graph cannot vary independently from the size of the smallest connected component. The same remark applies also for the size of the largest connected component. Having a graph invariant that directly relates the four graph parameters can dramatically improve the propagation.

- Even if the description of a global constraint involves one single graph parameter $C$, we can introduce the number of vertices, $\text{NVERTEX}$, and the number of arcs, $\text{NARC}$, of the final digraph. In this context, we can take advantage of graph invariants linking $C$, $\text{NARC}$ and $\text{NVERTEX}$.

- It also happens that we enforce two graph constraints $GC_1$ and $GC_2$ that have the same initial graph $G$. In this context we consider the following situations:
  - Each arc of $G$ belongs to one of the final graphs associated with $GC_1$ or with $GC_2$ (but not to both). An example of such global constraint is the change continuity constraint. Within the graph invariants this situation is denoted by $\text{apartition}$.
  - Each vertex of $G$ belongs to one of the final graphs associated with $GC_1$ or with $GC_2$ (but not to both). An example of such global constraint is the group constraint. Within the graph invariants this situation is denoted by $\text{vpartition}$.

In these situations the graph properties associated with the two graph constraints are also not independent.

In practice the graphs associated with global constraints have a regular structure that comes from the initial graph or from the property of the arc constraints. So, in addition to graph invariants that hold for any graph, we want also tighter graph invariants that hold for specific graph classes. The next section introduces the graph classes we consider, while the two other sections give the graph invariants on one and two graphs.

4.3.1 Graph classes

By definition, a graph invariant has to hold for any digraph. For instance, we have the graph invariant $\text{NARC} \leq \text{NVERTEX}^2$, which relates the number of arcs and the
number of vertices of any digraph. This invariant is sharp since the equality is reached for a clique. However, by considering the structure of a digraph, we can get sharper invariants. For instance, if our digraph is a subset of an elementary path (e.g., we use the \textit{PATH} arc generator depicted by Figure 2.4) we have that \( N_{\text{ARC}} \leq N_{\text{VERTEX}} - 1 \), which is a tighter bound of the maximum number of arcs since \( N_{\text{VERTEX}} - 1 < \frac{N_{\text{VERTEX}}^2}{2} \). For this reason, we consider recurring graph classes that show up for different global constraints of the catalogue. Beside the graph classes that were introduced in Section 2.2.2 we also have the following classes relating several graph constraints:

- \textit{apartition}: constraint defined by two graph constraints having the same initial graph, where each arc of the initial graph belongs to one of the final graph (but not to both).

- \textit{vpartition}: constraint defined by two graph constraints having the same initial graph, where each vertex of the initial graph belongs to one of the final graph (but not to both).

In addition, we also consider graph constraints such that their final graphs is a subset of the graph generated by the arc generators:

- \textit{CHAIN},
- \textit{CIRCUIT},
- \textit{CLIQUE},
- \textit{CLIQUE}(Comparison)
- \textit{GRID},
- \textit{LOOP},
- \textit{PATH},
- \textit{PRODUCT},
- \textit{PRODUCT}(Comparison),
- \textit{SYMMETRIC\_PRODUCT},
- \textit{SYMMETRIC\_PRODUCT}(Comparison),

where \textit{Comparison} is one of the following comparison operators \( \leq, \geq, <, >, =, \neq \).

### 4.3.2 Format of an invariant

As we previously saw, we have graph invariants that hold for any digraph as well as tighter graph invariants for specific graph classes. As a consequence, we partition the database in groups of graph invariants. A \textit{group of graph invariants} corresponds to several invariants such that all invariants relate the same subset of graph parameters and such that all invariants are variations of the first invariant of the group taking into accounts the graph class. Therefore, the first invariant of a group has no precondition, while all other invariants have a non-empty precondition that characterises the graph class for which they hold.
4.3. GRAPH INVARIANTS

EXAMPLE: As a first example consider the group of invariants denoted by Proposition 68, which relate the number of arcs $NARC$ with the number of vertices of the smallest and largest connected component (i.e., $MIN_{NCC}$ and $MAX_{NCC}$).

$$MIN_{NCC} \neq MAX_{NCC} \Rightarrow NARC \geq MIN_{NCC} + MAX_{NCC} - 2 + (MIN_{NCC} = 1)$$

equivalence: $MIN_{NCC} \neq MAX_{NCC} \Rightarrow NARC \geq MIN_{NCC}^2 + MAX_{NCC}^2$

On the one hand, since the first rule has no precondition it corresponds to a general graph invariant. On the other hand the second rule specifies a tighter condition (since $MIN_{NCC}^2 + MAX_{NCC}^2$ is greater than or equal to $MIN_{NCC} + MAX_{NCC} - 2 + (MIN_{NCC} = 1)$), which only holds for a final graph that is reflexive, symmetric and transitive.

EXAMPLE: As a second example, consider the following group of invariants corresponding to Proposition 51, which relate the number of arcs $NARC$ to the number of vertices $NVERTEX$ according to the arc generator (see Figure 2.4) used for generating the initial digraph:

$$NARC \leq NVERTEX^2$$

$arc\_gen = CHAIN : NARC \leq 2 \cdot NVERTEX - 2$

$arc\_gen = CLIQUE(\leq) : NARC \leq \frac{NVERTEX \cdot (NVERTEX + 1)}{2}$

$arc\_gen = CLIQUE(\geq) : NARC \leq \frac{NVERTEX \cdot (NVERTEX + 1)}{2}$

$arc\_gen = CLIQUE(<) : NARC \leq \frac{NVERTEX \cdot (NVERTEX - 1)}{2}$

$arc\_gen = CLIQUE(>) : NARC \leq \frac{NVERTEX \cdot (NVERTEX - 1)}{2}$

$arc\_gen = CLIQUE(\neq) : NARC \leq NVERTEX^2 - NVERTEX$

$arc\_gen = CYCLE : NARC \leq 2 \cdot NVERTEX$

$arc\_gen = PATH : NARC \leq NVERTEX - 1$

4.3.3 Using the database of invariants

The purpose of this section is to provide a set of graph invariants, each invariant relating a given set of graph parameters. Once we have these graph invariants we can use them systematically by applying the following steps:

- For a given graph constraint we extract all the graph parameters occurring in its description. This can be done automatically by scanning the corresponding graph properties. Let $GP$ denote this subset of graph parameters. For each graph parameter $gp$ of $GP$ we check if we have a graph property of the form $gp = var$ where $var$ is a domain variable. If this is the case we record the pair $(gp, var)$; if not, we create a new domain variable $var$ and also record the pair $(gp, var)$.

- We then search for all groups of graph invariants involving a subset of the previous graph parameters $GP$. For each selected group we filter out those graph invariants for which the preconditions are not compatible with the graph class
of the graph constraint under consideration. In each group we finally keep those invariants that have the maximum number of preconditions (i.e., the most specialised graph invariants).

- Finally we state all the previous collected graph invariants as implied constraints. This is achieved by using the variables associated with each graph parameter.

**EXAMPLE:** We continue with the example of the group constraint and its first graph constraint. The steps for creating the implied constraints are:

- We first extract the graph parameters \textit{NCC}, \textit{MIN\_NCC}, \textit{MAX\_NCC} and \textit{NVERTEX} from the first graph constraint of the group constraint. Since all the graph properties attached to the previous graph parameters have the form \( gc = var \) we extract the corresponding domain variables and get the following pairs \((\textit{NCC}, \textit{NGROUP}), (\textit{MIN\_NCC}, \textit{MIN\_SIZE}), (\textit{MAX\_NCC}, \textit{MAX\_SIZE})\) and \((\textit{NVERTEX}, \textit{NVAL})\).

- We search for all groups of graph invariants involving the graph parameters \textit{NCC}, \textit{MIN\_NCC}, \textit{MAX\_NCC} and \textit{NVERTEX} and filter out the irrelevant graph invariants that cannot be applied on the graph class associated with the group constraint.

- We state all the previous invariants by substituting each graph parameter by its corresponding variable, which leads to a set of implied constraints.

### 4.3.4 The database of graph invariants

For each combination of graph parameters we give the number of graph invariants we currently have. The items are sorted first in increasing number of graph parameters of the invariant, second in alphabetic order on the name of the parameters. All graph invariants assume a digraph for which each vertex has at least one arc. For some propositions, a figure depicts the corresponding final graph, which minimises or maximises a given graph parameter. The propositions of this section and their corresponding proofs use the notations introduced in Section 2.2.2 on page 57.

- Graph invariants involving one graph parameter of a final graph:
  - \textit{MAX\_NCC}: 1 (see Proposition 1),
  - \textit{MAX\_NSCC}: 2 (see Propositions 2 and 3),
  - \textit{MIN\_NCC}: 1 (see Proposition 4),
  - \textit{MIN\_NSCC}: 2 (see Propositions 5 and 6),
  - \textit{NARC}: 1 (see Proposition 7),
  - \textit{NCC}: 2 (see Propositions 8 and 9),
  - \textit{NSCC}: 1 (see Proposition 10),
  - \textit{NSINK}: 1 (see Proposition 11),
  - \textit{NSOURCE}: 1 (see Proposition 12),
  - \textit{NVERTEX}: 1 (see Proposition 13).

- Graph invariants involving two graph parameters of a final graph:
4.3. GRAPH INVARIANTS

- MAX_NCC, MAX_NSCC: 2 (see Propositions 14 and 15),
- MAX_NCC, MIN_NCC: 2 (see Propositions 16 and 17),
- MAX_NCC, NARC: 2 (see Propositions 18 and 19),
- MAX_NCC, NSINK: 2 (see Propositions 20 and 21),
- MAX_NCC, NSOURCE: 2 (see Propositions 22 and 23),
- MAX_NCC, NVERTEX: 2 (see Propositions 24 and 25),
- MAX_NSCC, MIN_NSCC: 2 (see Propositions 26 and 27),
- MAX_NSCC, NARC: 2 (see Propositions 28 and 29),
- MAX_NSCC, NVERTEX: 2 (see Propositions 30 and 31),
- MIN_NCC, MIN_NSCC: 2 (see Propositions 32 and 33),
- MIN_NCC, NARC: 2 (see Propositions 34 and 35),
- MIN_NCC, NCC: 1 (see Proposition 36),
- MIN_NCC, NVERTEX: 3 (see Propositions 37, 38 and 39),
- MIN_NSCC, NARC: 2 (see Propositions 40 and 41),
- MIN_NSCC, NVERTEX: 2 (see Propositions 42 and 43),
- NARC, NCC: 2 (see Propositions 44 and 45),
- NARC, NSCC: 2 (see Propositions 46 and 47),
- NARC, NSINK: 1 (see Proposition 48),
- NARC, NSOURCE: 1 (see Proposition 49),
- NARC, NVERTEX: 4 (see Propositions 50, 51, 52 and 53),
- NCC, NSCC: 2 (see Propositions 54 and 55),
- NCC, NVERTEX: 3 (see Propositions 56 and 57 and 58),
- NSCC, NSINK: 1 (see Proposition 59),
- NSCC, NSOURCE: 1 (see Proposition 60),
- NSCC, NVERTEX: 3 (see Propositions 61, 62 and 63),
- NSINK, NVERTEX: 2 (see Propositions 64 and 65),
- NSOURCE, NVERTEX: 2 (see Propositions 66 and 67).

- Graph invariants involving three graph parameters of a final graph:
  - MAX_NCC, MIN_NCC, NARC: 1 (see Proposition 68),
  - MAX_NCC, MIN_NCC, NCC: 1 (see Proposition 69),
  - MAX_NCC, MIN_NCC, NVERTEX: 5 (see Propositions 70, 71, 72, 73 and 74),
  - MAX_NCC, NARC, NCC: 2 (see Propositions 75 and 76),
  - MAX_NCC, NARC, NVERTEX: 2 (see Propositions 77 and 78),
  - MAX_NCC, NCC, NSINK: 1 (see Proposition 79),
  - MAX_NCC, NCC, NSOURCE: 1 (see Proposition 80),
  - MAX_NCC, NCC, NVERTEX: 2 (see Propositions 81 and 82),
  - MAX_NSCC, MIN_NSCC, NARC: 1 (see Proposition 83),
  - MAX_NSCC, MIN_NSCC, NSCC: 1 (see Proposition 84),
  - MAX_NSCC, MIN_NSCC, NVERTEX: 2 (see Propositions 85 and 86),
  - MAX_NSCC, NCC, NVERTEX: 1 (see Proposition 87),
  - MAX_NSCC, NSCC, NVERTEX: 2 (see Propositions 88 and 89).
CHAPTER 4. FURTHER TOPICS

- MIN_NCC, NARC, NVERTEX: 2 (see Propositions 90 and 91),
- MIN_NCC, NCC, NVERTEX: 2 (see Propositions 92 and 93),
- MIN_NSCC, NARC, NVERTEX: 1 (see Proposition 94),
- MIN_NSCC, NCC, NVERTEX: 1 (see Proposition 95),
- MIN_NSCC, NSCC, NVERTEX: 2 (see Propositions 96 and 97),
- NARC, NCC, NVERTEX: 2 (see Propositions 98 and 99),
- NARC, NSCC, NVERTEX: 4 (see Propositions 100, 101, 102 and 103),
- NARC, NSINK, NVERTEX: 2 (see Propositions 104 and 105),
- NSINK, NSOURCE, NVERTEX: 1 (see Proposition 108),
- NSINK, NSOURCE, NVERTEX: 1 (see Proposition 109).

- Graph invariants involving four graph parameters of a final graph:
  - MAX_NCC, MIN_NCC, NARC, NCC: 2 (see Propositions 110 and 111),
  - MAX_NCC, MIN_NCC, NCC, NVERTEX: 2 (see Propositions 112 and 113),
  - MAX_NCC, NCC, NSINK, NSOURCE: 1 (see Proposition 114),
  - MAX_NSCC, MIN_NSCC, NARC, NSCC: 2 (see Propositions 115 and 116),
  - MAX_NSCC, MIN_NSCC, NSCC, NVERTEX: 2 (see Propositions 117 and 118),
  - MIN_NCC, NARC, NCC, NVERTEX: 1 (see Proposition 119),
  - NARC, NCC, NSCC, NVERTEX: 2 (see Propositions 120 and 121),
  - NARC, NSINK, NSOURCE, NVERTEX: 1 (see Proposition 122).

- Graph invariants involving five graph parameters of a final graph:
  - MAX_NCC, MIN_NCC, NARC, NCC, NVERTEX: 1 (see Proposition 123),
  - MIN_NCC, NARC, NCC, NSCC, NVERTEX: 1 (see Proposition 124).

- Graph invariants relating two parameters of two final graphs:
  - MAX_NCC1, MIN_NCC1: 1 (see Proposition 125),
  - MAX_NCC2, MIN_NCC2: 1 (see Proposition 126),
  - MAX_NCC1, NCC2: 1 (see Proposition 127),
  - MAX_NCC2, NCC1: 1 (see Proposition 128),
  - MIN_NCC1, NCC2: 1 (see Proposition 129),
  - MIN_NCC2, NCC1: 1 (see Proposition 130),
  - NARC1, NARC2: 1 (see Proposition 131),
  - NCC1, NCC2: 2 (see Propositions 132 and 133),
  - NVERTEX1, NVERTEX2: 1 (see Proposition 134).

- Graph invariants relating three parameters of two final graphs:
  - MAX_NCC1, MIN_NCC1, MIN_NCC2: 3 (see Propositions 135, 136 and 137),
  - MAX_NCC2, MIN_NCC2, MIN_NCC1: 3 (see Propositions 138, 139 and 140),
  - MAX_NCC1, MIN_NCC1, NVERTEX2: 1 (see Proposition 141),
  - MAX_NCC2, MIN_NCC2, NVERTEX1: 1 (see Proposition 142),
4.3. GRAPH INVARIANTS

- $\text{MIN\_NCC}_1, \text{NARC}_2, \text{NCC}_1$: 1 (see Proposition 143),
- $\text{MIN\_NCC}_2, \text{NARC}_1, \text{NCC}_2$: 1 (see Proposition 144).

• Graph invariants relating four parameters of two final graphs:
  - $\text{MAX\_NCC}_1, \text{MIN\_NCC}_1, \text{MIN\_NCC}_2, \text{NCC}_1$: 2 (see Propositions 145 and 146),
  - $\text{MAX\_NCC}_2, \text{MIN\_NCC}_2, \text{MIN\_NCC}_1, \text{NCC}_2$: 2 (see Propositions 147 and 148),
  - $\text{MAX\_NCC}_1, \text{MIN\_NCC}_1, \text{MIN\_NCC}_2, \text{NVERTEX}_2$: 1 (see Proposition 149),
  - $\text{MAX\_NCC}_2, \text{MIN\_NCC}_2, \text{MIN\_NCC}_1, \text{NVERTEX}_1$: 1 (see Proposition 150).

• Graph invariants relating five parameters of two final graphs:
  - $\text{MAX\_NCC}_1, \text{MAX\_NCC}_2, \text{MIN\_NCC}_1, \text{MIN\_NCC}_2, \text{NCC}_1$: 7 (see Propositions 151, 152, 153, 154, 155, 156 and 157),
  - $\text{MAX\_NCC}_1, \text{MAX\_NCC}_2, \text{MIN\_NCC}_1, \text{MIN\_NCC}_2, \text{NCC}_2$: 7 (see Propositions 158, 159, 160, 161, 162, 163 and 164).

• Graph invariants relating six parameters of two final graphs:
  - $\text{MAX\_NCC}_1, \text{MAX\_NCC}_2, \text{MIN\_NCC}_1, \text{MIN\_NCC}_2, \text{NCC}_1, \text{NCC}_2$: 2 (see Propositions 165 and 166).
Graph invariants involving one parameter of a final graph

**Proposition 1.**

\[
\text{no loop: } \text{MAX}_N \text{C} \neq 1 \tag{4.1}
\]

*Proof.* Since we do not have any loop, a non-empty connected component has at least two vertices.

**Proposition 2.**

\[
\text{acyclic: } \text{MAX}_N \text{SCC} \leq 1 \tag{4.2}
\]

*Proof.* Since we do not have any circuit, a non-empty strongly connected component consists of one single vertex.

**Proposition 3.**

\[
\text{no loop: } \text{MAX}_N \text{SCC} \neq 1 \tag{4.3}
\]

*Proof.* Since we do not have any loop, a non-empty strongly connected component has at least two vertices.

**Proposition 4.**

\[
\text{no loop: } \text{MIN}_N \text{C} \neq 1 \tag{4.4}
\]

*Proof.* Since we do not have any loop, a non-empty connected component has at least two vertices.

**Proposition 5.**

\[
\text{acyclic: } \text{MIN}_N \text{SCC} \leq 1 \tag{4.5}
\]

*Proof.* Since we do not have any circuit, a non-empty strongly connected component consists of one single vertex.

**Proposition 6.**

\[
\text{no loop: } \text{MIN}_N \text{SCC} \neq 1 \tag{4.6}
\]

*Proof.* Since we do not have any loop, a non-empty strongly connected component has at least two vertices.

**Proposition 7.**

\[
\text{one succ: } \text{NARC} = \text{NVERTEX}_{\text{INITIAL}} \tag{4.7}
\]

*Proof.* By definition of one succ.
4.3. GRAPH INVARIANTS

**Proposition 8.**  
o loop: \( 2 \cdot NCC \leq NVERTEX_{\text{Initial}} \)  
\( (4.8) \)

*Proof.* By definition of no loop, each connected component has at least two vertices.

**Proposition 9.**  
consecutive_loops_are_connected: \( 2 \cdot NCC \leq NVERTEX_{\text{Initial}} + 1 \)  
\( (4.9) \)

*Proof.* By definition of consecutive_loops_are_connected.

**Proposition 10.**  
no loop: \( 2 \cdot NSCC \leq NVERTEX_{\text{Initial}} \)  
\( (4.10) \)

*Proof.* By definition of no loop, each strongly connected component has at least two vertices.

**Proposition 11.**  
symmetric: \( NSINK = 0 \)  
\( (4.11) \)

*Proof.* Since we do not have any isolated vertex.

**Proposition 12.**  
symmetric: \( NSOURCE = 0 \)  
\( (4.12) \)

*Proof.* Since we do not have any isolated vertex.

**Proposition 13.**  
one_succ: \( NVERTEX = NVERTEX_{\text{Initial}} \)  
\( (4.13) \)

*Proof.* By definition of one_succ.
Graph invariants involving two parameters of a final graph

**MAX\_NCC, MAX\_NSCC**

**Proposition 14.**
\[
\text{MAX\_NCC} = 0 \iff \text{MAX\_NSCC} = 0
\] (4.14)

**Proof.** By definition of MAX\_NCC and of MAX\_NSCC.

**Proposition 15.**
\[
\text{MAX\_NSCC} \leq \text{MAX\_NCC}
\] (4.15)

**Proof.** MAX\_NSCC is a lower bound of the size of the largest connected component since the largest strongly connected component is for sure included within a connected component.

**MAX\_NCC, MIN\_NCC**

**Proposition 16.**
\[
\text{MAX\_NCC} = 0 \iff \text{MIN\_NCC} = 0
\] (4.16)

**Proof.** By definition of MAX\_NCC and of MIN\_NCC.

**Proposition 17.**
\[
\text{MIN\_NCC} \leq \text{MAX\_NCC}
\] (4.17)

**Proof.** By definition of MIN\_NCC and of MAX\_NCC.

**MAX\_NCC, NARC**

**Proposition 18.**
\[
\text{MAX\_NCC} = 0 \iff \text{NARC} = 0
\] (4.18)

**Proof.** By definition of MAX\_NCC and of NARC.

**Proposition 19.**
\[
\text{MAX\_NCC} > 0 \implies \text{NARC} \geq \max(1, \text{MAX\_NCC} - 1)
\] (4.19)

\[\text{symmetric: MAX\_NCC} > 0 \implies \text{NARC} \geq \max(1, 2 \cdot \text{MAX\_NCC} - 2)\] (4.20)

\[\text{equivalence: NARC} \geq \text{MAX\_NCC}^2\] (4.21)

\[\text{arc\_gen = PATH: NARC} \geq \text{MAX\_NCC} - 1\] (4.22)

**Proof.**
(4.19) \text{MAX\_NCC} – 1 arcs are needed to connect \text{MAX\_NCC} vertices that belong to a given connected component containing at least two vertices. And one arc is required for a connected component containing one single vertex.

(4.20) Similarly, when the graph is symmetric, \(2 \cdot \text{MAX\_NCC} – 2\) arcs are needed to connect \text{MAX\_NCC} vertices that belong to a given connected component containing at least two vertices.

(4.21) Finally, when the graph is reflexive, symmetric and transitive, \text{MAX\_NCC}^2\ arcs are needed to connect \text{MAX\_NCC} vertices that belong to a given connected component.

(4.22) When the initial graph corresponds to a path, the minimum number of arcs of a connected component involving \(n\) vertices is equal to \(n – 1\).
4.3. GRAPH INVARIANTS

**Proposition 20.**
\[ \text{MAX}_NCC = 0 \Rightarrow \text{NSINK} = 0 \quad (4.23) \]

**Proof.** By definition of \text{MAX}_NCC and of \text{NSINK}.

**Proposition 21.**
\[ \text{NSINK} \geq 1 \Rightarrow \text{MAX}_NCC \geq 2 \quad (4.24) \]

**Proof.** Since we do not have any isolated vertex a sink is connected to at least one other vertex. Therefore, if the graph has a sink, there exists at least one connected component with at least two vertices.

**Proposition 22.**
\[ \text{MAX}_NCC = 0 \Rightarrow \text{NSOURCE} = 0 \quad (4.25) \]

**Proof.** By definition of \text{MAX}_NCC and of \text{NSOURCE}.

**Proposition 23.**
\[ \text{NSOURCE} \geq 1 \Rightarrow \text{MAX}_NCC \geq 2 \quad (4.26) \]

**Proof.** Since we do not have any isolated vertex a source is connected to at least one other vertex. Therefore, if the graph has a source, there exists at least one connected component with at least two vertices.

**Proposition 24.**
\[ \text{MAX}_NCC = 0 \Leftrightarrow \text{NVERTEX} = 0 \quad (4.27) \]

**Proof.** By definition of \text{MAX}_NCC and of \text{NVERTEX}.

**Proposition 25.**
\[ \text{NVERTEX} \geq \text{MAX}_NCC \quad (4.28) \]

**Proof.** By definition of \text{MAX}_NCC.

**Proposition 26.**
\[ \text{MAX}_{NSCC} = 0 \Leftrightarrow \text{MIN}_{NSCC} = 0 \quad (4.29) \]

**Proof.** By definition of \text{MAX}_{NSCC} and of \text{MIN}_{NSCC}.

**Proposition 27.**
\[ \text{MIN}_{NSCC} \leq \text{MAX}_{NSCC} \quad (4.30) \]

**Proof.** By definition of \text{MIN}_{NSCC} and of \text{MAX}_{NSCC}.
**MAX_NSCC, NARC**

Proposition 28.
\[
\text{MAX_NSCC} = 0 \iff \text{NARC} = 0 \quad (4.31)
\]

*Proof.* By definition of \text{MAX_NSCC} and of \text{NARC}.

Proposition 29.
\[
\text{NARC} \geq \text{MAX_NSCC} \quad (4.32)
\]

*Proof.* Symmetric: \text{NARC} \geq 2 \cdot \text{MAX_NSCC} \quad (4.33)

*Proof.* Equivalence: \text{NARC} \geq \text{MAX_NSCC}^2 \quad (4.34)

*Proof.* (4.32) In a strongly connected component at least one arc has to leave each vertex. Since we have at least one strongly connected component of \text{MAX_NSCC} vertices this leads to the previous inequality.

**MAX_NSCC, NVERTEX**

Proposition 30.
\[
\text{MAX_NSCC} = 0 \iff \text{NVERTEX} = 0 \quad (4.35)
\]

*Proof.* By definition of \text{MAX_NSCC} and of \text{NVERTEX}.

Proposition 31.
\[
\text{NVERTEX} \geq \text{MAX_NSCC} \quad (4.36)
\]

*Proof.* By definition of \text{MAX_NSCC}.

**MIN_NCC, MIN_NSCC**

Proposition 32.
\[
\text{MIN_NCC} = 0 \iff \text{MIN_NSCC} = 0 \quad (4.37)
\]

*Proof.* By definition of \text{MIN_NCC} and of \text{MIN_NSCC}.

Proposition 33.
\[
\text{MIN_NCC} \geq \text{MIN_NSCC} \quad (4.38)
\]

*Proof.* By construction \text{MIN_NCC} is an upper bound of the number of vertices of the smallest strongly connected component.
4.3. GRAPH INVARIANTS

\[ **MIN\_NCC, NARC** \]

**Proposition 34.**
\[ MIN\_NCC = 0 \iff NARC = 0 \] (4.39)

*Proof.* By definition of \( MIN\_NCC \) and of \( NARC \).

\[ **MIN\_NCC, NARC** \]

**Proposition 35.**
\[ MIN\_NCC > 0 \Rightarrow NARC \geq \max(1, MIN\_NCC - 1) \] (4.40)

\[ \text{symmetric: } MIN\_NCC > 0 \Rightarrow NARC \geq \max(1, 2 \cdot MIN\_NCC - 2) \] (4.41)

\[ \text{equivalence: } NARC \geq MIN\_NCC^2 \] (4.42)

\[ \text{arc\_gen} = PATH: NARC \geq MIN\_NCC - 1 \] (4.43)

*Proof.* Similar to Proposition 19.

\[ **MIN\_NCC, NCC** \]

**Proposition 36.**
consecutive_loops_are_connected: \((MIN\_NCC + 1) \cdot NCC \leq NVERTEX\_INITIAL + 1\)

(4.44)

*Proof.* By definition of consecutive_loops_are_connected.

\[ **MIN\_NCC, NVERTEX** \]

**Proposition 37.**
\[ MIN\_NCC = 0 \iff NVERTEX = 0 \] (4.45)

*Proof.* By definition of \( MIN\_NCC \) and of \( NVERTEX \).

**Proposition 38.**
\[ NVERTEX \geq MIN\_NCC \] (4.46)

*Proof.* By definition of \( MIN\_NCC \).

**Proposition 39.**
\[ MIN\_NCC \notin \left[ \min \left( \left\lfloor \frac{NVERTEX}{2} \right\rfloor, \left\lfloor \frac{NVERTEX\_INITIAL - 1}{2} \right\rfloor \right) + 1, NVERTEX - 1 \right] \] (4.47)

*Proof.* On the one hand, if NCC \leq 1, we have that \( MIN\_NCC \geq NVERTEX \). On the other hand, if NCC > 1, we have that \( MIN\_NCC + MIN\_NCC \leq NVERTEX \) and that \( MIN\_NCC + MIN\_NCC + 1 \leq NVERTEX\_INITIAL \), which by isolating \( MIN\_NCC \) and by grouping the two inequalities leads to \( MIN\_NCC \leq \min \left( \left\lfloor \frac{NVERTEX}{2} \right\rfloor, \left\lfloor \frac{NVERTEX\_INITIAL - 1}{2} \right\rfloor \right) \). The result follows.
Proposition 40.

\[ \text{MIN}_{\text{NSCC}} = 0 \iff NARC = 0 \]  
(4.48)

*Proof.* By definition of \( \text{MIN}_{\text{NSCC}} \) and of \( NARC \).

Proposition 41.

\[ NARC \geq \text{MIN}_{\text{NSCC}} \]  
(4.49)

symmetric: \[ NARC \geq 2 \cdot \text{MIN}_{\text{NSCC}} \]  
(4.50)

equivalence: \[ NARC \geq \text{MIN}_{\text{NSCC}}^2 \]  
(4.51)

*Proof.* Similar to Proposition 29.

Proposition 42.

\[ \text{MIN}_{\text{NSCC}} = 0 \iff \text{NVERTEX} = 0 \]  
(4.52)

*Proof.* By definition of \( \text{MIN}_{\text{NSCC}} \) and of \( \text{NVERTEX} \).

Proposition 43.

\[ \text{NVERTEX} \geq \text{MIN}_{\text{NSCC}} \]  
(4.53)

*Proof.* By definition of \( \text{MIN}_{\text{NSCC}} \).

Proposition 44.

\[ NARC = 0 \iff \text{NCC} = 0 \]  
(4.54)

*Proof.* By definition of \( NARC \) and of \( \text{NCC} \).

Proposition 45.

\[ NARC \geq \text{NCC} \]  
(4.55)

*Proof.* Each connected component contains at least one arc (since, by hypothesis, each vertex has at least one arc).

Proposition 46.

\[ NARC = 0 \iff \text{NSCC} = 0 \]  
(4.56)

*Proof.* By definition of \( NARC \) and of \( \text{NSCC} \).

Proposition 47.

\[ NARC \geq \text{NSCC} \]  
(4.57)

no loop: \[ NARC \geq 2 \cdot \text{NSCC} \]  
(4.58)

*Proof.* 4.57 (respectively 4.58) holds since each strongly connected component contains at least one (respectively two) arc(s).


4.3. GRAPH INVARIANTS

Proposition 48.

\[ \text{NARC} \geq \text{NSINK} \quad (4.59) \]

Proof. Since isolated vertices are not allowed, each sink has a distinct ingoing arc.

Proposition 49.

\[ \text{NARC} \geq \text{NSOURCE} \quad (4.60) \]

Proof. Since isolated vertices are not allowed, each source has a distinct outgoing arc.

Proposition 50.

\[ \text{NARC} = 0 \iff \text{NVERTEX} = 0 \quad (4.61) \]

Proof. By definition of \text{NARC} and of \text{NVERTEX}.

Proposition 51.

\[ \begin{align*}
\text{arc\_gen} &= \text{CIRCUIT} : \text{NARC} \leq \text{NVERTEX} & (4.62) \\
\text{arc\_gen} &= \text{CHAIN} : \text{NARC} \leq 2 \cdot \text{NVERTEX} - 2 & (4.63) \\
\text{arc\_gen} &= \text{CLIQUE}(\leq) : \text{NARC} \leq \frac{\text{NVERTEX} \cdot (\text{NVERTEX} + 1)}{2} & (4.64) \\
\text{arc\_gen} &= \text{CLIQUE}(\geq) : \text{NARC} \leq \frac{\text{NVERTEX} \cdot (\text{NVERTEX} + 1)}{2} & (4.65) \\
\text{arc\_gen} &= \text{CLIQUE}(\langle) : \text{NARC} \leq \frac{\text{NVERTEX} \cdot (\text{NVERTEX} - 1)}{2} & (4.66) \\
\text{arc\_gen} &= \text{CLIQUE}(\rangle) : \text{NARC} \leq \frac{\text{NVERTEX} \cdot (\text{NVERTEX} - 1)}{2} & (4.67) \\
\text{arc\_gen} &= \text{CLIQUE}(\neq) : \text{NARC} \leq \text{NVERTEX}^2 - \text{NVERTEX} & (4.68) \\
\text{arc\_gen} &= \text{CYCLE} : \text{NARC} \leq 2 \cdot \text{NVERTEX} & (4.69) \\
\text{arc\_gen} &= \text{PATH} : \text{NARC} \leq \text{NVERTEX} - 1 & (4.70) \\
\end{align*} \]

Proof. 4.62 holds since each vertex of a digraph can have at most \text{NVERTEX} successors. The next items correspond to the maximum number of arcs that can be achieved according to a specific arc generator.

Note that, when the equality is reached in 4.62, the corresponding extreme graph is in fact the graph initially generated. The same observation holds for inequalities 4.63 to 4.71. As a consequence all \text{U}-arcs have to be turned into \text{T}-arcs.

Proposition 52.

\[ 2 \cdot \text{NARC} \geq \text{NVERTEX} \quad (4.72) \]

Proof. By induction on the number of vertices of a graph \( G \):

1. If \( \text{NVERTEX}(G) \) is equal to 1 or 2 Proposition 52 holds.
2. Assume that $\text{NVERTEX}(G) \geq 3$.

- Assume there exists a vertex $v$ such that, if we remove $v$, we do not create any isolated vertex in the remaining graph. We have $\text{NARC}(G) \geq \text{NARC}(G - v) + 1$. Thus $2 \cdot \text{NARC}(G) \geq 2 \cdot \text{NARC}(G - v) + 1$. Since by induction hypothesis $2 \cdot \text{NARC}(G - v) \geq \text{NVERTEX}(G - v) = \text{NVERTEX}(G) - 1$ the result holds.

- Otherwise, all the connected components of $G$ are reduced to two elements with only one arc. We remove one of such connected component $(v, w)$.

Thus $\text{NARC}(G) = \text{NARC}(G - \{v, w\}) + 1$. As by induction hypothesis, $2 \cdot \text{NARC}(G - \{v, w\}) \geq \text{NVERTEX}(G - \{v, w\}) = \text{NVERTEX}(G) - 2$ the result holds.

Note that, when the equality is reached in 52, the corresponding extreme graph is in fact a perfect matching of the graph. As a consequence all $U$-arcs that do not belong to any perfect matching have to be turned into $F$-arcs.

**Proposition 53.**

\[ \text{arc.gen} = \text{LOOP} : \text{NARC} = \text{NVERTEX} \quad (4.73) \]

**Proof.** From the definition of $\text{LOOP}$. ☐

**Proposition 54.**

\[ \text{NCC} = 0 \Leftrightarrow \text{NSCC} = 0 \quad (4.74) \]

**Proof.** By definition of $\text{NCC}$ and of $\text{NSCC}$. ☐

**Proposition 55.**

\[ \text{NCC} \leq \text{NSCC} \quad (4.75) \]

**Proof.** Holds since each connected component contains at least one strongly connected component. ☐

Note that, when the equality is reached in 55, each connected component of the corresponding extreme graph is strongly connected. As a consequence all sink vertices of the graph induced by the $T$-vertices and the $T$-arcs should have at least one successor.

**Proposition 56.**

\[ \text{NCC} = 0 \Leftrightarrow \text{NVERTEX} = 0 \quad (4.76) \]

**Proof.** By definition of $\text{NCC}$ and of $\text{NVERTEX}$. ☐

**Proposition 57.**

\[ \text{NCC} \leq \text{NVERTEX} \quad (4.77) \]

\[ \text{no loop} : 2 \cdot \text{NCC} \leq \text{NVERTEX} \quad (4.78) \]

**Proof.** 4.77 (respectively 4.78) holds since each connected component contains at least one (respectively two) vertex. ☐
Note that, when the equality is reached in 4.77, the corresponding extreme graph does not contain any arc between two distinct vertices. As a consequence any $U$-arc between two distinct vertices is turned into a $F$-vertex.

**Proposition 58.**

\[
\text{vpartition} \land \text{consecutive_loops_are_connected} : \quad N\text{VERTEX} \leq N\text{VERTEX}_{\text{INITIAL}} - (N\text{CC} - 1) \tag{4.79}
\]

**Proof.** Holds since between two “consecutive” connected components of the initial graph there is at least one vertex that is missing. \qed

**Proposition 59.**

\[
N\text{SCC} \geq N\text{INK} + 1 \tag{4.80}
\]

**Proof.** Since each sink cannot belong to a circuit and since no isolated vertex is allowed at least one extra non-sink vertex is required the result follows. \qed

**Proposition 60.**

\[
N\text{SCC} \geq N\text{SOURCE} + 1 \tag{4.81}
\]

**Proof.** Since each source cannot belong to a circuit and since no isolated vertex is allowed at least one extra non-source vertex is required the result follows. \qed

**Proposition 61.**

\[
N\text{SCC} = 0 \iff N\text{VERTEX} = 0 \tag{4.82}
\]

**Proof.** By definition of $N\text{SCC}$ and of $N\text{VERTEX}$. \qed

**Proposition 62.**

\[
N\text{SCC} \leq N\text{VERTEX} \tag{4.83}
\]

**Proof.** Proposition 62 holds since each strongly connected component contains at least one vertex. \qed

**Proposition 63.**

\[
\text{acyclic} : N\text{SCC} = N\text{VERTEX} \tag{4.84}
\]

**Proof.** In a directed acyclic graph we have that each vertex corresponds to a strongly connected component involving only that vertex. \qed
Proposition 64.

\[ N_{\text{VERTEX}} = 0 \Rightarrow N_{\text{SINK}} = 0 \]  \hspace{1cm} (4.85)

Proof. By definition of \( N_{\text{VERTEX}} \) and of \( N_{\text{SINK}} \).

Proposition 65.

\[ N_{\text{VERTEX}} > 0 \Rightarrow N_{\text{SINK}} < N_{\text{VERTEX}} \]  \hspace{1cm} (4.86)

Proof. Holds since each sink must have a predecessor that cannot be a sink and since each vertex has at least one arc.

Proposition 66.

\[ N_{\text{VERTEX}} = 0 \Rightarrow N_{\text{SOURCE}} = 0 \]  \hspace{1cm} (4.87)

Proof. By definition of \( N_{\text{VERTEX}} \) and of \( N_{\text{SOURCE}} \).

Proposition 67.

\[ N_{\text{VERTEX}} > 0 \Rightarrow N_{\text{SOURCE}} < N_{\text{VERTEX}} \]  \hspace{1cm} (4.88)

Proof. Holds since each source must have a successor that cannot be a source and since each vertex has at least one arc.
4.3. GRAPH INVARIANTS

Graph invariants involving three parameters of a final graph

\textbf{MAX\_NCC, MIN\_NCC, NARC}

Proposition 68.

\begin{equation}
\text{MIN\_NCC} \neq \text{MAX\_NCC} \Rightarrow \\
\text{NARC} \geq \text{MIN\_NCC} + \text{MAX\_NCC} - 2 + (\text{MIN\_NCC} = 1)
\end{equation}

\text{equivalence:}

\begin{equation}
\text{MIN\_NCC} \neq \text{MAX\_NCC} \Rightarrow \\
\text{NARC} \geq \text{MIN\_NCC}^2 + \text{MAX\_NCC}^2
\end{equation}

\textbf{Proof.} (4.89) \( n - 1 \) arcs are needed to connect \( n \) \( (n > 1) \) vertices that all belong to a given connected component. Since we have two connected components, which respectively have \text{MIN\_NCC} and \text{MAX\_NCC} vertices, this leads to the previous inequality. When \text{MIN\_NCC} is equal to one we need an extra arc. \( \Box \)

\textbf{MAX\_NCC, MIN\_NCC, NCC}

Proposition 69.

\begin{equation}
\text{MIN\_NCC} \neq \text{MAX\_NCC} \Rightarrow \text{NCC} \geq 2
\end{equation}

\textbf{Proof.} If \text{MIN\_NCC} and \text{MAX\_NCC} are different then they correspond for sure to at least two distinct connected components. \( \Box \)

\textbf{MAX\_NCC, MIN\_NCC, NVERTEX}

Proposition 70.

\begin{equation}
\text{MIN\_NCC} \neq \text{MAX\_NCC} \Rightarrow \text{NVERTEX} \geq \text{MIN\_NCC} + \text{MAX\_NCC}
\end{equation}

\textbf{Proof.} Since we have at least two distinct connected components, which respectively have \text{MIN\_NCC} and \text{MAX\_NCC} vertices, this leads to the previous inequality. \( \Box \)

Proposition 71.

\begin{equation}
\text{MAX\_NCC} \leq \max(\text{MIN\_NCC}, \text{NVERTEX} - \max(1, \text{MIN\_NCC}))
\end{equation}

\textbf{Proof.} On the one hand, if \text{NCC} \leq 1, we have that \text{MAX\_NCC} \leq \text{MIN\_NCC}. On the other hand, if \text{NCC} > 1, we have that \text{NVERTEX} \geq \max(1, \text{MIN\_NCC}) + \text{MAX\_NCC} (i.e., \text{MAX\_NCC} \leq \text{NVERTEX} - \max(1, \text{MIN\_NCC})). The result is obtained by taking the maximum value of the right-hand sides of the two inequalities. \( \Box \)

Proposition 72.

\begin{equation}
\text{MIN\_NCC} \notin [\text{NVERTEX} - \max(1, \text{MAX\_NCC}) + 1, \text{NVERTEX} - 1]
\end{equation}

\textbf{Proof.} On the one hand, if \text{NCC} \leq 1, we have that \text{MIN\_NCC} \geq \text{NVERTEX}. On the other hand, if \text{NCC} > 1, we have that \text{MIN\_NCC} + \max(1, \text{MAX\_NCC}) \leq \text{NVERTEX} (i.e., \text{MIN\_NCC} \leq \text{NVERTEX} - \max(1, \text{MAX\_NCC})). The result follows. \( \Box \)

Proposition 73.

\begin{equation}
\text{NVERTEX} \notin [\text{MIN\_NCC} + 1, \text{MIN\_NCC} + \text{MAX\_NCC} - 1]
\end{equation}
CHAPTER 4. FURTHER TOPICS

Proof. On the one hand, if $NCC \leq 1$, we have that $N\text{VERTEX} \leq \text{MIN}_NCC$. On the other hand, if $NCC > 1$, we have that $N\text{VERTEX} \geq \text{MIN}_NCC + \text{MAX}_NCC$. Since $\text{MIN}_NCC \leq \text{MIN}_NCC + \text{MAX}_NCC$ the result follows.

\[\text{Proposition 74.}\]

if $\text{MIN}_NCC > 0$

then $k_{inf} = \left\lfloor \frac{N\text{VERTEX} + \text{MIN}_NCC}{\text{MIN}_NCC} \right\rfloor$ else $k_{inf} = 1$

if $\text{MAX}_NCC > 0$

then $k_{sup} = \left\lfloor \frac{N\text{VERTEX} - 1}{\text{MAX}_NCC} \right\rfloor$ else $k_{sup} = N\text{VERTEX}$

if $\text{MAX}_NCC < \text{MIN}_NCC$

then $k_{sup} = \left\lfloor \frac{\text{MIN}_NCC - 2}{\text{MAX}_NCC - \text{MIN}_NCC} \right\rfloor$ else $k_{sup} = N\text{VERTEX}$

\[\forall k \in [k_{inf}, k_{sup}] : N\text{VERTEX} \notin [k \cdot \text{MAX}_NCC + 1, (k + 1) \cdot \text{MIN}_NCC - 1] \quad (4.96)\]

Proof. We make the proof for $k \in \mathbb{N}$ (the interval $[k_{inf}, k_{sup}]$ is only used for restricting the number of intervals to check). We have that $N\text{VERTEX} \in [k \cdot \text{MIN}_NCC, k \cdot \text{MAX}_NCC]$. A forbidden interval $[k \cdot \text{MAX}_NCC + 1, (k + 1) \cdot \text{MIN}_NCC - 1]$ corresponds to an interval between the end of interval $[k \cdot \text{MIN}_NCC, k \cdot \text{MAX}_NCC]$ and the start of the next interval $[(k + 1) \cdot \text{MIN}_NCC, (k + 1) \cdot \text{MAX}_NCC]$. Since all intervals $[i \cdot \text{MIN}_NCC, i \cdot \text{MAX}_NCC]$ ($i < k$) end before $k \cdot \text{MAX}_NCC$ and since all intervals $[j \cdot \text{MIN}_NCC, j \cdot \text{MAX}_NCC]$ ($j > k$) start after $(k + 1) \cdot \text{MIN}_NCC$, they do not use any value in $[k \cdot \text{MAX}_NCC + 1, (k + 1) \cdot \text{MIN}_NCC - 1]$.

\[\text{Proposition 75.}\]

\[N\text{ARC} \leq \text{NCC} \cdot \text{MAX}_NCC^2 \quad (4.97)\]

\[\text{arc}\_\text{gen} = \text{PATH} : N\text{ARC} \leq \text{NCC} \cdot (\text{MAX}_NCC - 1) \quad (4.98)\]

Proof. On the one hand, (4.97) holds since the maximum number of arcs is achieved by taking $\text{NCC}$ connected components where each connected component is a clique involving $\text{MAX}_NCC$ vertices. On the other hand, (4.98) holds since a tree of $n$ vertices has $n - 1$ arcs.

\[\text{Proposition 76.}\]

\[N\text{ARC} \geq \text{MAX}_NCC + \text{NCC} - 2 \quad (4.99)\]

Proof. The minimum number of arcs is achieved by taking one connected component with $\text{MAX}_NCC$ vertices and $\text{MAX}_NCC - 1$ arcs as well as $\text{NCC} - 1$ connected components with one single vertex and a loop.
Proposition 77. \[ \text{NARC} \leq \text{MAX}_\text{NCC}^2 \frac{\text{NVERTEX}}{\max(1, \text{MAX}_\text{NCC})} + (\text{NVERTEX} \mod \max(1, \text{MAX}_\text{NCC}))^2 \]

\[(4.100)\]

Figure 4.1: Illustration of Proposition 77. A graph that achieves the maximum number of arcs according to the size of the largest connected component as well as a fixed number of vertices (MAX_NCC = 3, NVERTEX = 11, NARC = 3^2 \cdot \left\lfloor \frac{11}{\max(1, 3)} \right\rfloor + (11 \mod \max(1, 3))^2 = 31)

Proof. If MAX_NCC = 0 we get NARC \leq 0 which holds since the set of vertices is empty. We now assume that MAX_NCC > 0. We first begin with the following claim:

Let \( G \) be a graph such that \( V(G) - \text{NCC}(G, \text{MAX}_\text{NCC}(G)) * \text{MAX}_\text{NCC}(G) \geq \text{MAX}_\text{NCC}(G) \), then there exists a graph \( G' \) such that \( V(G') = V(G) \), \( \text{MAX}_\text{NCC}(G') = \text{MAX}_\text{NCC}(G) \), \( \text{NCC}(G', \text{MAX}_\text{NCC}(G')) = \text{NCC}(G, \text{MAX}_\text{NCC}(G)) + 1 \) and \( |E(G)| \leq |E(G')| \).

Proof of the claim

Let \( (C_i)_{i \in [n]} \) be the connected components of \( G \) on less than MAX_NCC(G) vertices and such that \( |C_i| \geq |C_{i+1}| \). By hypothesis there exists \( k \leq n \) such that \( \bigcup_{i=1}^{k-1} C_i < \text{MAX}_\text{NCC}(G) \) and \( |\bigcup_{i=1}^{k} C_i| \geq \text{MAX}_\text{NCC}(G) \).

- Either \( |\bigcup_{i=1}^{k} C_i| = \text{MAX}_\text{NCC}(G) \), and then with \( G' \) such that \( G' \) restricted to the \( \bigcup_{i=1}^{k} C_i \) be a complete graph and \( G' \) restricted to \( V(G) - \bigcup_{i=1}^{k} C_i \) being exactly \( G \) restricted to \( V(G) - \bigcup_{i=1}^{k} C_i \) we obtain the claim.

- Or \( |\bigcup_{i=1}^{k} C_i| > \text{MAX}_\text{NCC}(G) \). Then \( C_k = C_k^1 \cup C_k^2 \) such that \( |\bigcup_{i=1}^{k-1} C_i \cup C_k^1| = \text{MAX}_\text{NCC}(G) \) and \( |C_k^2| < |C_1| \) (notice that \( k \geq 2 \)). Then with \( G' \) such that \( G' \) restricted to \( \bigcup_{i=1}^{k-1} C_i \cup C_k^1 \) is a complete graph and \( G' \) restricted to \( V(G) - (\bigcup_{i=1}^{k-1} C_i \cup C_k^1) \) is exactly \( G \) restricted to \( V(G) - (\bigcup_{i=1}^{k-1} C_i \cup C_k^1) \) we obtain the claim.

End of proof of the claim
We prove by induction on \( r(G) = \left\lfloor \frac{\text{NVERTEX}(G)}{\text{MAX}_\text{NCC}(G)} \right\rfloor - \text{NCC}(G, \text{MAX}_\text{NCC}(G)) \), where \( G \) is any graph. For \( r(G) = 0 \) the result holds (see Prop 44). Otherwise, since \( r(G) > 0 \) we have that \( V(G) - \text{NCC}(G, \text{MAX}_\text{NCC}(G)) - \text{MAX}_\text{NCC}(G) \geq \text{MAX}_\text{NCC}(G) \), by the previous claim there exists \( G' \) with the same number of vertices and the same number of vertices in the largest connected component, such that \( r(G') = r(G) - 1 \). Consequently the result holds by induction.

\[ \text{Proposition 78.} \]

\[
\text{NARC} \geq \text{MAX}_\text{NCC} - 1 + \left\lfloor \frac{\text{NVERTEX} - \text{MAX}_\text{NCC} + 1}{2} \right\rfloor \]  \tag{4.101}

\[ \text{Proof.} \] Let \( G \) be a graph, let \( X \) be a maximal size connected component of \( G \), then we have \( G = G[X] \oplus G[V(G) - X] \). On the one hand, as \( G[X] \) is connected, by setting \( \text{NCC} = 1 \) in 4.143 of Proposition 99, we have \( |E(G[X])| \geq |X| - 1 \), on the other hand, by Proposition 52, \( |E(G[V(G) - X])| \geq \left\lceil \frac{|V(G) - X|}{2} \right\rceil \). Thus the result follows.

\[ \text{Proposition 79.} \]

\[ \text{NSINK} \leq \text{NCC} \cdot \max(0, \text{MAX}_\text{NCC} - 1) \]  \tag{4.102}

\[ \text{Proof.} \] Since a connected component contains at most \( \text{MAX}_\text{NCC} \) vertices and since it does not contain any isolated vertex a connected component involves at most \( \text{MAX}_\text{NCC} - 1 \) sinks. Thus the result follows.

\[ \text{Proposition 80.} \]

\[ \text{NSOURCE} \leq \text{NCC} \cdot \max(0, \text{MAX}_\text{NCC} - 1) \]  \tag{4.103}

\[ \text{Proof.} \] Similar to Proposition 79.

\[ \text{Proposition 81.} \]

\[ \text{NVERTEX} \leq \text{NCC} \cdot \text{MAX}_\text{NCC} \]  \tag{4.104}

\[ \text{Proof.} \] The number of vertices is less than or equal to the number of connected components multiplied by the largest number of vertices in a connected component.

\[ \text{Proposition 82.} \]

\[ \text{NVERTEX} \geq \text{MAX}_\text{NCC} + \max(0, \text{NCC} - 1) \]  \tag{4.105}

\[ \text{no loop} : \text{NVERTEX} \geq \text{MAX}_\text{NCC} + \max(0, 2 \cdot \text{NCC} - 2) \]  \tag{4.106}

\[ \text{Proof.} \] (4.105) The minimum number of vertices according to a fixed number of connected components \( \text{NCC} \) such that one of the connected component contains \( \text{MAX}_\text{NCC} \) vertices is obtained as follows: we get \( \text{MAX}_\text{NCC} \) vertices from the connected component involving \( \text{MAX}_\text{NCC} \) vertices and one vertex for each remaining connected component.
4.3. GRAPH INVARIANTS

**Proposition 83.**

\[
\text{MIN}_\text{NSCC} \neq \text{MAX}_\text{NSCC} \Rightarrow \text{NARC} \geq \text{MIN}_\text{NSCC} + \text{MAX}_\text{NSCC} \quad (4.107)
\]

equivalence: \( \text{MIN}_\text{NSCC} \neq \text{MAX}_\text{NSCC} \Rightarrow \text{NARC} \geq \text{MIN}_\text{NSCC}^2 + \text{MAX}_\text{NSCC}^2 \quad (4.108) \)

**Proof.** (4.107) In a strongly connected component at least one arc has to leave each arc. Since we have two strongly connected components, which respectively have \text{MIN}_\text{NSCC} and \text{MAX}_\text{NSCC} vertices, this leads to the previous inequality.

**Proposition 84.**

\[
\text{MIN}_\text{NSCC} \neq \text{MAX}_\text{NSCC} \Rightarrow \text{NSCC} \geq 2 \quad (4.109)
\]

**Proof.** Follows from the definitions of \text{MIN}_\text{NSCC} and of \text{MAX}_\text{NSCC}.

**Proposition 85.**

\[
\text{MIN}_\text{NSCC} \neq \text{MAX}_\text{NSCC} \Rightarrow \text{NVERTEX} \geq \text{MIN}_\text{NSCC} + \text{MAX}_\text{NSCC} \quad (4.110)
\]

**Proof.** Since we have at least two distinct strongly connected components, which respectively have \text{MIN}_\text{NSCC} and \text{MAX}_\text{NSCC} vertices, this leads to the previous inequality.

**Proposition 86.**

\[
\text{if } \text{MIN}_\text{NSCC} > 0
\]

then \( k_{\text{inf}} = \left\lfloor \frac{\text{NVERTEX} + \text{MIN}_\text{NSCC}}{\text{MIN}_\text{NSCC}} \right\rfloor \) else \( k_{\text{inf}} = 1 \)

\[
\text{if } \text{MAX}_\text{NSCC} > 0
\]

then \( k_{\text{sup}1} = \left\lfloor \frac{\text{NVERTEX} - 1}{\text{MAX}_\text{NSCC}} \right\rfloor \) else \( k_{\text{sup}1} = \text{NVERTEX} \)

\[
\text{if } \text{MAX}_\text{NSCC} < \text{MIN}_\text{NSCC}
\]

then \( k_{\text{sup}2} = \left\lfloor \frac{\text{MIN}_\text{NSCC} - 2}{\text{MAX}_\text{NSCC} - \text{MIN}_\text{NSCC}} \right\rfloor \) else \( k_{\text{sup}2} = \text{NVERTEX} \)

\[
k_{\text{sup}} = \min(k_{\text{sup}1}, k_{\text{sup}2})
\]

\[
\forall k \in [k_{\text{inf}}, k_{\text{sup}}] : \text{NVERTEX} \notin [k \cdot \text{MAX}_\text{NSCC} + 1, (k + 1) \cdot \text{MIN}_\text{NSCC} - 1] \quad (4.111)
\]

**Proof.** Similar to Proposition 74.
Proposition 87.
\[ \text{NVERTEX} \leq \text{NCC} \cdot \text{NSCC} \] \hspace{1cm} (4.112)

Proof. The largest number of vertices is obtained by putting within each connected component the number of vertices of the largest strongly connected component.

Proposition 88.
\[ \text{NVERTEX} \leq \text{NSCC} \cdot \text{MAX\_NSCC} \] \hspace{1cm} (4.113)

Proof. Since each strongly connected component contains at most \text{MAX\_NSCC} vertices the total number of vertices is less than or equal to \text{NSCC} \cdot \text{MAX\_NSCC}.

Proposition 89.
\[ \text{NVERTEX} \geq \text{MAX\_NSCC} + \max(0, \text{NSCC} - 1) \] \hspace{1cm} (4.114)

Proof. \((4.114)\) The minimum number of vertices according to a fixed number of strongly connected components \text{NSCC} such that one of them contains \text{MAX\_NSCC} vertices is equal to \text{MAX\_NSCC} + \max(0, \text{NSCC} - 1).

Proposition 90.
\[ \text{NARC} \leq \text{MIN\_NCC}^2 + (\text{NVERTEX} - \text{MIN\_NCC})^2 \] \hspace{1cm} (4.116)

arc_gen = \text{CIRCUIT} : \text{NARC} \leq \text{NVERTEX} - 2 \cdot (\text{MIN\_NCC} < \text{NVERTEX}) \hspace{1cm} (4.117)

arc_gen = \text{CHAIN} : \text{NARC} \leq \text{NVERTEX} - 2 \cdot (\text{MIN\_NCC} < \text{NVERTEX}) \hspace{1cm} (4.118)

arc_gen = \text{CLIQUE(\leq)} : \text{NARC} \leq \frac{\text{MIN\_NCC} \cdot (\text{MIN\_NCC} + 1)}{2} + \frac{(\text{NVERTEX} - \text{MIN\_NCC}) \cdot (\text{NVERTEX} - \text{MIN\_NCC} + 1)}{2} \hspace{1cm} (4.119)

arc_gen = \text{CLIQUE(\geq)} : \text{NARC} \leq \frac{\text{MIN\_NCC} \cdot (\text{MIN\_NCC} + 1)}{2} + \frac{(\text{NVERTEX} - \text{MIN\_NCC}) \cdot (\text{NVERTEX} - \text{MIN\_NCC} + 1)}{2} \hspace{1cm} (4.120)

arc_gen = \text{CLIQUE(<)} : \text{NARC} \leq \frac{\text{MIN\_NCC} \cdot (\text{MIN\_NCC} - 1)}{2} + \frac{(\text{NVERTEX} - \text{MIN\_NCC}) \cdot (\text{NVERTEX} - \text{MIN\_NCC} - 1)}{2} \hspace{1cm} (4.121)

arc_gen = \text{CLIQUE(>)} : \text{NARC} \leq \frac{\text{MIN\_NCC} \cdot (\text{MIN\_NCC} - 1)}{2} + \frac{(\text{NVERTEX} - \text{MIN\_NCC}) \cdot (\text{NVERTEX} - \text{MIN\_NCC} - 1)}{2} \hspace{1cm} (4.122)
4.3. GRAPH INVARIANTS

\[ \text{arc\_gen} = CLIQUE(\neq) : \text{NARC} \leq \text{MIN}\_NCC^2 - \text{MIN}\_NCC + (\text{NVERTEX} - \text{MIN}\_NCC)^2 - (\text{NVERTEX} - \text{MIN}\_NCC) \]  
\[ (4.123) \]

\[ \text{arc\_gen} = CYCLE : \text{NARC} \leq \text{NVERTEX} - 4 \cdot (\text{MIN}\_NCC < \text{NVERTEX}) \]  
\[ (4.124) \]

\[ \text{arc\_gen} = PATH : \text{NARC} \leq \max(0, \text{MIN}\_NCC - 1) + \max(0, \text{NVERTEX} - \text{MIN}\_NCC - 1) \]  
\[ (4.125) \]

**Proof.** (4.116) The maximum number of vertices according to a fixed number of vertices \( \text{NVERTEX} \) and to the fact that there is a connected component with \( \text{MIN}\_NCC \) vertices is obtained by:

- Building a connected component with \( \text{MIN}\_NCC \) vertices and creating an arc between each pair of vertices.
- Building a connected component with all the \( \text{NVERTEX} - \text{MIN}\_NCC \) remaining vertices and creating an arc between each pair of vertices.

\[ \Box \]

**Proposition 91.**

\[ \text{MIN}\_NCC > 1 \Rightarrow \text{NARC} \geq \left\lfloor \frac{\text{NVERTEX}}{\text{MIN}\_NCC} \right\rfloor \cdot (\text{MIN}\_NCC - 1) + \text{NVERTEX} \mod \text{MIN}\_NCC \]  
\[ (4.126) \]

**Proof.** Achieving the minimum number of arcs with a fixed number of vertices and with a minimum number of vertices greater than or equal to one in each connected component is achieved in the following way:

- Since the minimum number of arcs of a connected component of \( n \) vertices is \( n - 1 \), splitting a connected component into \( k \) parts that all have more than one vertex saves \( k - 1 \) arcs. Therefore we build a maximum number of connected components. Since each connected component has at least \( \text{MIN}\_NCC \) vertices we get \( \left\lfloor \frac{\text{NVERTEX}}{\text{MIN}\_NCC} \right\rfloor \) connected components.
- Since we cannot build a connected component with the rest of the vertices (i.e., \( \text{NVERTEX} \mod \text{MIN}\_NCC \) vertices left) we have to incorporate them in the previous connected components and this costs one arc for each vertex.

\[ \Box \]

When \( \text{MIN}\_NCC = 1 \), note that Proposition 52 provides a lower bound on the number of arcs.
Proposition 92.
\[ \text{NVERTEX} \geq \text{NCC} \cdot \text{MIN\_NCC} \quad (4.127) \]

*Proof.* The smallest number of vertices is obtained by taking all connected components to their minimum number of vertices MIN\_NCC.

Proposition 93.
\[ \text{NVERTEX} > \text{MIN\_NCC} \Rightarrow \text{NCC} \geq 2 \quad (4.128) \]

*Proof.* If all vertices do not fit within the smallest connected component then we have at least two connected components.

Proposition 94.
\[ \text{NARC} \leq \text{NVERTEX}^2 + \text{MIN\_NSCC}^2 - \text{NVERTEX} \cdot \text{MIN\_NSCC} \quad (4.129) \]

*Proof.* Achieving the maximum number of arcs, provided that we have at least one strongly connected component with MIN\_NSCC vertices, is done by:

- Building a first strongly connected component \( C_1 \) with MIN\_NSCC vertices and adding an arc between each pair of vertices of \( C_1 \).
- Building a second strongly connected component \( C_2 \) with NVERTEX – MIN\_NSCC vertices and adding an arc between each pair of vertices of \( C_2 \).

Finally, we add an arc from every vertex of \( C_1 \) to every vertex of \( C_2 \). This leads to a total number of arcs of MIN\_NSCC\(^2\) + (NVERTEX – MIN\_NSCC)\(^2\) + MIN\_NSCC \cdot (NVERTEX – MIN\_NSCC).

Proposition 95.
\[ \text{NVERTEX} \geq \text{NCC} \cdot \text{MIN\_NSCC} \quad (4.130) \]

*Proof.* The smallest number of vertices is obtained by putting within each connected component the number of vertices of the smallest strongly connected component.

Proposition 96.
\[ \text{NVERTEX} \geq \text{NSCC} \cdot \text{MIN\_NSCC} \quad (4.131) \]

*Proof.* Since each strongly connected component contains at least MIN\_NSCC vertices the total number of vertices is greater than or equal to NSCC \cdot MIN\_NSCC.

Proposition 97.
\[ \text{NVERTEX} > \text{MIN\_NSCC} \Rightarrow \text{NSCC} \geq 2 \quad (4.132) \]

*Proof.* If all vertices do not fit within the smallest strongly connected component then we have at least two strongly connected components.
4.3. GRAPH INVARIANTS

NARC, NCC, NVERTEX

Proposition 98.

\[
\text{NARC} \leq (\text{NVERTEX} - \text{NCC} + 1)^2 + \text{NCC} - 1 \tag{4.133}
\]

\[
\text{arc} \_ \text{gen} = \text{CIRCUIT} : \text{NARC} \leq \text{NVERTEX} - \text{NCC} + 1 - (\text{NCC} \neq 1) \tag{4.134}
\]

\[
\text{arc} \_ \text{gen} = \text{CHAIN} : \text{NARC} \leq 2 \cdot \text{NVERTEX} - 2 \cdot \text{NCC} \tag{4.135}
\]

\[
\text{arc} \_ \text{gen} = \text{CLIQUE}(\leq) : \text{NARC} \leq \text{NCC} - 1 + \\
(\text{NVERTEX} - \text{NCC} + 1) \cdot (\text{NVERTEX} - \text{NCC} + 2) \tag{4.136}
\]

\[
\text{arc} \_ \text{gen} = \text{CLIQUE}(\geq) : \text{NARC} \leq \text{NCC} - 1 + \\
(\text{NVERTEX} - \text{NCC} + 1) \cdot (\text{NVERTEX} - \text{NCC} + 2) \tag{4.137}
\]

\[
\text{arc} \_ \text{gen} = \text{CLIQUE}(\langle) : \text{NARC} \leq \text{NCC} - 1 + \\
(\text{NVERTEX} - \text{NCC} + 1) \cdot (\text{NVERTEX} - \text{NCC}) \tag{4.138}
\]

\[
\text{arc} \_ \text{gen} = \text{CLIQUE}(\rangle) : \text{NARC} \leq \text{NCC} - 1 + \\
(\text{NVERTEX} - \text{NCC} + 1) \cdot (\text{NVERTEX} - \text{NCC}) \tag{4.139}
\]

\[
\text{arc} \_ \text{gen} = \text{CLIQUE}(\neq) : \text{NARC} \leq \max(0, \text{NCC} - 1) + \\
(\text{NVERTEX} - \text{NCC} + 1)^2 - (\text{NVERTEX} - \text{NCC} + 1) \tag{4.140}
\]

\[
\text{arc} \_ \text{gen} = \text{CYCLE} : \text{NARC} \leq 2 \cdot \text{NVERTEX} - 2 \cdot \text{NCC} + 2 \cdot (\text{NCC} = 1) \tag{4.141}
\]

\[
\text{arc} \_ \text{gen} = \text{PATH} = \text{NARC} = \text{NVERTEX} - \text{NCC} \tag{4.142}
\]

Figure 4.2: Illustration of Proposition 98. A graph that achieves the maximum number of arcs according to a fixed number of connected components as well as to a fixed number of vertices (NCC = 5, NVERTEX = 7, NARC = (7 - 5 + 1)^2 + 5 - 1 = 13)

Proof. (4.133) We proceed by induction on \( T(G) = \text{NVERTEX}(G) - |X| - (\text{NCC}(G) - 1) \), where \( X \) is any connected component of \( G \) of maximum cardinality. For \( T(G) = 0 \) then either \( \text{NCC}(G) = 1 \) and thus the formula is clearly true, or all the connected components of \( G \), but possibly \( X \), are reduced to one element. Since isolated vertices are not allowed, the formula holds.

Assume that \( T(G) \geq 1 \). Then there exists \( Y \), a connected component of \( G \) distinct from \( X \), with more than one vertex. Let \( y \in Y \) and let \( G' \) be the graph such that \( V(G') = V(G) \) and \( E(G') \) is defined by:
For all $Z$ connected components of $G$ distinct from $X$ and $Y$ we have $G[Z] = G[Z]$. 

With $X' = X \cup \{y\}$ and $Y' = Y - \{y\}$, we have $G'[Y'] = G[Y']$ and $E(G'[X']) = E(G[X]) \cup \{(x, y), (y, x)\}$. 

Clearly $|E(G')| - |E(G)| \geq 2 \cdot |X| + 1 - (2 \cdot |Y| - 1)$ and since $X$ is of maximal cardinality the difference is strictly positive. Now as $N_{\text{VERTEX}}(G') = N_{\text{VERTEX}}(G), N_{\text{CC}}(G') = N_{\text{CC}}(G)$ and as $T(G') = T(G) - 1$ the result holds by induction hypothesis.

**Proposition 99.**

$$N_{\text{ARC}} \geq N_{\text{VERTEX}} - N_{\text{CC}}$$  (4.143)

**Equivalence:** $N_{\text{CC}} > 0 \Rightarrow$

$$N_{\text{ARC}} \geq (N_{\text{VERTEX}} \mod N_{\text{CC}}) \cdot \left(\left\lceil \frac{N_{\text{VERTEX}}}{N_{\text{CC}}} \right\rceil + 1\right)^2 + (N_{\text{CC}} - N_{\text{VERTEX}} \mod N_{\text{CC}}) \cdot \left\lceil \frac{N_{\text{VERTEX}}}{N_{\text{CC}}} \right\rceil^2$$  (4.144)

**Proof.** (4.143) By induction of the number of vertices. The formula holds for one vertex. Let $G$ a graph with $n + 1$ vertices ($n \geq 1$). First assume there exists $x$ in $G$ such that $G - x$ has the same number of connected components than $G$. Since $N_{\text{ARC}}(G) \geq N_{\text{ARC}}(G - x) + 1$, and by induction hypothesis $N_{\text{ARC}}(G - x) \geq N_{\text{VERTEX}}(G - x) - N_{\text{CC}}(G - x)$ the result holds. Otherwise all connected components of $G$ are reduced to one vertex and the formula holds.

**Proposition 100.**

$$N_{\text{ARC}} \leq (N_{\text{VERTEX}} - N_{\text{SCC}} + 1) \cdot N_{\text{VERTEX}} + \frac{N_{\text{SCC}} \cdot (N_{\text{SCC}} - 1)}{2}$$  (4.145)

**Equivalence:** $N_{\text{ARC}} \leq N_{\text{SCC}} - 1 + (N_{\text{VERTEX}} - N_{\text{SCC}} + 1)^2$  (4.146)

**Proof.** 4.145, it is easier to rewrite the formula as $N_{\text{ARC}} \leq (N_{\text{VERTEX}} - (N_{\text{SCC}} - 1))^2 + \frac{(N_{\text{SCC}} - 1) \cdot (N_{\text{VERTEX}} - (N_{\text{SCC}} - 1)) + N_{\text{SCC}} \cdot (N_{\text{SCC}} - 1)}{2}$. We proceed by induction on $T(G) = N_{\text{VERTEX}}(G) - |X| - (N_{\text{CC}}(G) - 1)$, where $X$ is any strongly connected component of $G$ of maximum cardinality.
For \( T(G) = 0 \) then either \( \text{NSCC}(G) = 1 \) and thus the formula is clearly true, or all the strongly connected components of \( G \), but possibly \( X \), are reduced to one element. Since the maximum number of arcs in a directed acyclic graph of \( n \) vertices is \( \frac{n(n+1)}{2} \), and as the subgraph of \( G \) induced by all the strongly connected components of \( G \) excepted \( X \) is acyclic, the formula clearly holds.

Assume that \( T(G) \geq 1 \), let \(( X_i )_{i \in I} \) be the family of strongly connected components of \( G \), and let \( G_r \) be the reduced graph of \( G \) induced by \(( X_i )_{i \in I} \) (that is \( V(G_r) = I \) and \( \forall i_1, i_2 \in I, (i_1, i_2) \in E(G_r) \) if and only if \( \exists x_1 \in X_{i_1}, \exists x_2 \in X_{i_2} \) such that \( (x_1, x_2) \in E \)). Consider \( G' \) such that \( V(G') = V(G) \) and \( E(G') \) is defined by:

- For all strongly connected components \( Z \) of \( G \) we have \( G'[Z] = G[Z] \).
- For \( \sigma \) be any topological sort of \( G_r \), \( \forall x_i \in X_i, \forall x_j \in X_j, (x_i, x_j) \in E(G') \) whenever \( i \) is less than \( j \) with respect to \( \sigma \).

Notice that \( G' \) satisfies the following properties: \( T(G') = T(G) \), \( V(G') = V(G) \), \( \text{NSCC}(G') = \text{NSCC}(G) \), \( E(G) \subseteq E(G') \). \(( X_i )_{i \in I} \) is still the family of strongly connected components of \( G' \), and moreover, for every \( i \in I \) and every \( x_i \in X_i \) we have that \( x_i \) is connected to any vertex outside \( X_i \), that is the number of arcs incident to \( x_i \) and incident to vertices outside \( X_i \) is exactly \( |V(G')| - |X_i| \).

Now, as \( T(G') \geq 1 \), there exists \( Y \), a strongly connected component of \( G' \) distinct from \( X \), with more than one vertex. Let \( y \in Y \) and let \( G'' \) be the graph such that \( V(G'') = V(G') \) and \( E(G'') \) is defined by:

- \( G''[V(G) - \{ y \}] = G'[V(G) - \{ y \}] \).
- With \( X' = X \cup \{ y \} \), we have \( G''[X'] = G'[X'] \) and \( E(G''[X']) = E(G'[X]) \cup (\cup_{x \in X} \{ (x, y), (y, x) \}) \).
- Assume that \( X = X_j \) for \( j \in I \). Then \( \forall i \in I - \{ j \}, \forall x_i \in X_i, (x_i, y) \in E(G'') \) whenever \( i \) is less than \( j \) with respect to \( \sigma \) and \( (y, x_i) \in E(G'') \) whenever \( j \) is less than \( i \) with respect to \( \sigma \).

Clearly \( |E(G'')| - |E(G')| \geq 2|X| + 1 + |V(G')| - |X| - (2 \cdot |Y| - 1 + |V(G')| - |Y|) = |X| - |Y| + 2 \) and since \( X \) is of maximal cardinality the difference is strictly positive. As \( E(G) \subseteq E(G') \), \( |E(G')| - |E(G)| \) is also strictly positive. Now as \( \text{NVERTEX}(G') = \text{NVERTEX}(G), \text{NSCC}(G'') = \text{NSCC}(G) \) and as \( T(G'') = T(G') - 1 = T(G) - 1 \) the result holds by induction hypothesis.

**Proposition 101.**

\[
\text{NARC} \geq \text{NVERTEX} - \left\lfloor \frac{\text{NSCC} - 1}{2} \right\rfloor \tag{4.147}
\]

\[
\text{equivalence: } \text{NSCC} > 0 \Rightarrow \text{NARC} \geq (\text{NVERTEX} \mod \text{NSCC}) \cdot \left( \left\lfloor \frac{\text{NVERTEX}}{\text{NSCC}} \right\rfloor + 1 \right)^2 + (\text{NSCC} - \text{NVERTEX} \mod \text{NSCC}) \cdot \left\lfloor \frac{\text{NVERTEX}}{\text{NSCC}} \right\rfloor^2 \tag{4.148}
\]

**Proof.** For proving part 4.147 of Proposition 101 we proceed by induction on \( \text{NSCC}(G) \). If \( \text{NSCC}(G) = 1 \) then, we have \( \text{NARC}(G) \geq \text{NVERTEX}(G) \) (i.e., for one vertex this is true since every vertex has at least one arc, otherwise every vertex \( v \) has an arc arriving on it as well as an arc starting from it, thus we have \( \text{NARC} \geq \frac{2 \text{NVERTEX}}{2} \)). If \( \text{NSCC}(G) > 1 \) let \( X \) be a strongly connected component of \( G \). Then \( \text{NARC}(G) \geq \text{NARC}(G[V(G) -
Figure 4.4: Illustration of Proposition 4.147. A graph that achieves the minimum number of arcs according to a fixed number of strongly connected components as well as to a fixed number of vertices ($\text{NSCC} = 7$, $\text{NVERTEX} = 10$, $\text{NARC} = 10 - \left\lceil \frac{7}{2} \right\rceil = 7$)

\[ X] \right\rceil \) + $\text{NARC}(G[X])$. By induction hypothesis $\text{NARC}(G[V(G) - X]) \geq |V(G) - X| - \left\lceil \frac{\text{NSCC}(G[V(G) - X]) - 1}{2} \right\rceil$, thus $\text{NARC}(G[V(G) - X]) \geq |V(G) - X| - \left\lceil \frac{\text{NSCC}(G) - 1}{2} \right\rceil$.

Since $\text{NARC}(G[X]) \geq |X|$ we obtain $\text{NARC}(G) \geq |V(G)| - \left\lceil \frac{\text{NSCC}(G)}{2} \right\rceil$, and thus the result holds.

**Proposition 102.**

\[
\text{equivalence: NVERTEX} > 0 \Rightarrow \text{NSCC} \geq \left\lceil \frac{\text{NVERTEX}^2}{\text{NARC}} \right\rceil \tag{4.149}
\]

**Proof.** As shown in [58], a lower bound for the minimum number of equivalence classes (e.g., strongly connected components) is the independence number of the graph and the right-hand side of Proposition 102 corresponds to a lower bound of the independence number proposed by Turán [385].

**Proposition 103.**

\[
\text{equivalence: NVERTEX} > 0 \Rightarrow \text{NSCC} \geq \left\lceil 2 \cdot \left( \frac{\text{NARC} - \text{NSINK}}{\text{NARC} \cdot \text{NVERTEX}} \right) \right\rceil + 1 \tag{4.150}
\]

**Proof.** See [185] and [150].

**Proposition 104.**

\[
\text{NARC} \leq (\text{NVERTEX} - \text{NSINK}) \cdot \text{NVERTEX} \tag{4.151}
\]

**Proof.** The maximum number of arcs is achieved by the following pattern: for all non-sink vertices we have an arc to all vertices.

**Proposition 105.**

\[
\text{NARC} \geq \text{NSINK} + \max(0, \text{NVERTEX} - 2 \cdot \text{NSINK}) \tag{4.152}
\]
Figure 4.5: Illustration of Proposition 105. Graphs that achieve the minimum number of arcs according to a fixed number of sinks as well as to a fixed number of vertices (A : $\text{NSINK} = 3, \text{NVERTEX} = 5, \text{NARC} = 3 + \max(0, 5 - 2 \cdot 3) = 3$; B : $\text{NSINK} = 3, \text{NVERTEX} = 9, \text{NARC} = 3 + \max(0, 9 - 2 \cdot 3) = 6$)

Proof. Recall that for $x \in V(G)$, we have that $d^+_G(x) + d^-_G(x) \geq 1$. If $x$ is a sink then $d^-_G(x) \geq 1$, consequently $\text{NARC}(G) \geq \text{NSINK}(G)$. If $x$ is not a sink then $d^+_G(x) \geq 1$, consequently $\text{NARC}(G) \geq |V(G)| - \text{NSINK}(G)$. □

NARC, NSOURCE, NVERTEX

Proposition 106.

\[ \text{NARC} \leq (\text{NVERTEX} - \text{NSOURCE}) \cdot \text{NVERTEX} \] (4.153)

Proof. The maximum number of arcs is achieved by the following pattern: for all non-source vertices we have an arc from all vertices. □

Proposition 107.

\[ \text{NARC} \geq \text{NSOURCE} + \max(0, \text{NVERTEX} - 2 \cdot \text{NSOURCE}) \] (4.154)

Proof. Similar to Proposition 105. □

NSCC, NSINK, NSOURCE

Proposition 108.

\[ \text{NSCC} \geq \text{NSINK} + \text{NSOURCE} \] (4.155)

Proof. Since sinks and sources cannot belong to a circuit and since they cannot coincide (i.e., because isolated vertices are not allowed) the result follows. □
NSINK, NSOURCE, NVERTEX

Proposition 109.

\[ \text{NVERTEX} \geq \text{NSINK} + \text{NSOURCE} \]  \hspace{1cm} (4.156)

\textit{Proof.} No vertex can be both a source and a sink (isolated vertices are removed). \qed
4.3. GRAPH INVARIANTS

Graph invariants involving four parameters of a final graph

Proposition 110. Let \( \alpha \) denote \( \max(0, \text{NCC} - 1) \).

\[
\text{NARC} \leq \alpha \cdot \text{MAX}_N \text{NCC}^2 + \text{MIN}_N \text{NCC}^2
\]  
(4.157)

\[
\text{arc_gen} = \text{CIRCUIT} : \text{NARC} \leq \alpha \cdot \text{MAX}_N \text{NCC} + \text{MIN}_N \text{NCC}
\]  
(4.158)

\[
\text{arc_gen} = \text{CHAIN} : \text{NARC} \leq \alpha \cdot (2 \cdot \text{MAX}_N \text{NCC} - 2) + 2 \cdot \text{MIN}_N \text{NCC} - 2
\]  
(4.159)

\[
\text{arc_gen} \in \{ \text{CLIQUE}(\leq) , \text{CLIQUE}(\geq) \} : \text{NARC} \leq
\alpha \cdot \frac{\text{MAX}_N \text{NCC} \cdot (\text{MAX}_N \text{NCC} + 1)}{2} + \frac{\text{MIN}_N \text{NCC} \cdot (\text{MIN}_N \text{NCC} + 1)}{2}
\]  
(4.160)

\[
\text{arc_gen} \in \{ \text{CLIQUE}(\leq) , \text{CLIQUE}(\geq) \} : \text{NARC} \leq
\alpha \cdot \frac{\text{MAX}_N \text{NCC} \cdot (\text{MAX}_N \text{NCC} - 1)}{2} + \frac{\text{MIN}_N \text{NCC} \cdot (\text{MIN}_N \text{NCC} - 1)}{2}
\]  
(4.161)

\[
\text{arc_gen} = \text{CLIQUE}(\neq) : \text{NARC} \leq \text{MIN}_N \text{NCC}^2 - \text{MIN}_N \text{NCC} +
\alpha \cdot (\text{MAX}_N \text{NCC}^2 - \text{MAX}_N \text{NCC})
\]  
(4.162)

\[
\text{arc_gen} = \text{CYCLE} : \text{NARC} \leq 2 \cdot \alpha \cdot \text{MAX}_N \text{NCC} + 2 \cdot \text{MIN}_N \text{NCC}
\]  
(4.163)

\[
\text{arc_gen} = \text{PATH} : \text{NARC} \leq \alpha \cdot (\text{MAX}_N \text{NCC} - 1) + \text{MIN}_N \text{NCC} - 1
\]  
(4.164)

Proof. We construct \( \text{NCC} - 1 \) connected components with \( \text{MAX}_N \text{NCC} \) vertices and one connected component with \( \text{MIN}_N \text{NCC} \) vertices. \( n^2 \) corresponds to the maximum number of arcs in a connected component. \( n, 2 \cdot n - 2, \frac{n \cdot (n+1)}{2}, \frac{n \cdot (n+1)}{2}, \frac{n \cdot (n-1)}{2}, \frac{n \cdot (n-1)}{2}, n^2 - n, 2 \cdot n \) and \( n - 1 \) respectively correspond to the maximum number of arcs in a connected component of \( n \) vertices according to the fact that we use the arc generator \( \text{CIRCUIT}, \text{CHAIN}, \text{CLIQUE}(\leq) \text{CLIQUE}(\geq) \text{CLIQUE}(\leq) \text{CLIQUE}(\geq) \text{CLIQUE}(\neq) \text{CYCLE} \text{or PATH} \).

Proposition 111.

\( \text{NCC} > 0 \Rightarrow \text{NARC} \geq (\text{NCC} - 1) \cdot \max(1, \text{MIN}_N \text{NCC} - 1) + \max(1, \text{MAX}_N \text{NCC} - 1)
\]  
(4.165)

\[
\text{arc_gen} = \text{PATH} : \text{NARC} \geq \max(0, \text{NCC} - 1) \cdot (\text{MIN}_N \text{NCC} - 1) + \text{MAX}_N \text{NCC} - 1
\]  
(4.166)

Proof. (4.165) We construct \( \text{NCC} - 1 \) connected components with \( \text{MIN}_N \text{NCC} \) vertices and one connected component with \( \text{MAX}_N \text{NCC} \) vertices. The quantity \( \max(1, n - 1) \) corresponds to the minimum number of arcs in a connected component of \( n \) \( (n > 0) \) vertices.
CHAPTER 4. FURTHER TOPICS

**MAX\_NCC, MIN\_NCC, NCC, NVERTEX**

Proposition 112.

\[
N\text{VERTEX} \leq \max(0, \ NCC - 1) \cdot \text{MAX\_NCC} + \text{MIN\_NCC}
\]  (4.167)

Proof. Derived from the definitions of \text{MIN\_NCC} and \text{MAX\_NCC}. \qed

Proposition 113.

\[
N\text{VERTEX} \geq \max(0, \ NCC - 1) \cdot \text{MIN\_NCC} + \text{MAX\_NCC}
\]  (4.168)

Proof. Derived from the definitions of \text{MIN\_NCC} and \text{MAX\_NCC}. \qed

**MAX\_NCC, NARC, NSOURCE, NVERTEX**

Proposition 114.

\[
\text{NSINK} + \text{NSOURCE} \leq \text{NCC} \cdot \max(0, \text{MAX\_NCC} - 1)
\]  (4.169)

Proof. Since a connected component contains at most \text{MAX\_NCC} vertices and since it does not contain any isolated vertex and since a same vertex cannot be both a sink and a source a connected component involves at most \text{MAX\_NCC} − 1 sinks and sources altogether. Thus the result follows. \qed

**MAX\_NSCC, MIN\_NSCC, NARC, NSCC**

Proposition 115.

\[
\text{NARC} \leq \max(0, \text{NSCC} - 1) \cdot \text{MAX\_NSCC}^2 + \text{MIN\_NSCC}^2 + \max(0, \text{NSCC} - 1) \cdot \text{MIN\_NSCC} \cdot \text{MAX\_NSCC} + \text{MAX\_NSCC}^2 \cdot \frac{\max(0, \text{NSCC} - 2) \cdot \max(0, \text{NSCC} - 1)}{2}
\]  (4.170)

Proof. We assume that we have at least two strongly connected components (the case with one being obvious). Let \((\text{SCC}_i)_{i \in [\text{NCC}(G)]}\) be the family of strongly connected components of G. Then \(|E(G)| \leq \sum_{i \in [\text{NCC}(G)]} |E(G[\text{SCC}_i])| + k\), where k is the number of arcs between the distinct strongly connected components of G. For any strongly connected component \text{SCC}_i the number of arcs it has with the other strongly connected components is bounded by \(|\text{SCC}_i| \cdot (|V(G)| - |\text{SCC}_i|)\). Consequently, \(k \leq \frac{1}{2} \cdot \sum_{i \in [\text{NCC}(G)]} |\text{SCC}_i| \cdot (|V(G)| - |\text{SCC}_i|)\); W.l.o.g. we assume \(|\text{SCC}_1| = \text{MIN\_NCC}\). Then we get \(k \leq \frac{1}{2} \cdot (\text{MIN\_NCC} \cdot (\text{NCC} - 1) \cdot \text{MAX\_NCC} + \text{MAX\_NCC} \cdot ((\text{NCC} - 2) \cdot \text{MAX\_NCC} + \text{MIN\_NCC}))\). \qed

Proposition 116.

\[
\text{NARC} \geq \max(0, \text{NSCC} - 1) \cdot \text{MIN\_NSCC} + \text{MAX\_NSCC}
\]  (4.171)

Proof. Let \((\text{SCC}_i)_{i \in [\text{NCC}(G)]}\) be the family of strongly connected components of G, as \(|E(G)| \geq \sum_{i \in [\text{NCC}(G)]} |E(G[\text{SCC}_i])|\), we obtain the result since in a strongly connected graph the number of edges is at least its number of vertices. \qed
4.3. GRAPH INVARIANTS

**Proposition 117.**

\[ \text{NVERTEX} \leq \max(0, \text{NSCC} - 1) \cdot \text{MAX}_\text{NSCC} + \text{MIN}_\text{NSCC} \quad (4.172) \]

*Proof.* Derived from the definitions of \( \text{MIN}_\text{NSCC} \) and \( \text{MAX}_\text{NSCC} \).

**Proposition 118.**

\[ \text{NVERTEX} \geq \max(0, \text{NSCC} - 1) \cdot \text{MIN}_\text{NSCC} + \text{MAX}_\text{NSCC} \quad (4.173) \]

*Proof.* Derived from the definitions of \( \text{MIN}_\text{NSCC} \) and \( \text{MAX}_\text{NSCC} \).

**Proposition 119.** Let \( \alpha, \beta \) and \( \gamma \) respectively denote \( \max(0, \text{NCC} - 1) \), \( \text{NVERTEX} - \alpha \cdot \text{MIN}_\text{NCC} \) and \( \text{MIN}_\text{NCC} \).

\[ \text{NARC} \leq \alpha \cdot \gamma^2 + \beta^2 \quad (4.174) \]

\[ \text{arc}_\text{gen} \in \{ \text{CLIQUE}(\leq), \text{CLIQUE}(\geq) \} : \text{NARC} \leq \alpha \cdot \frac{\gamma \cdot (\gamma + 1)}{2} + \frac{\beta \cdot (\beta + 1)}{2} \quad (4.175) \]

\[ \text{arc}_\text{gen} \in \{ \text{CLIQUE}(<), \text{CLIQUE}(>) \} : \text{NARC} \leq \alpha \cdot \frac{\gamma \cdot (\gamma - 1)}{2} + \frac{\beta \cdot (\beta - 1)}{2} \quad (4.176) \]

\[ \text{arc}_\text{gen} = \text{CLIQUE}(\neq) : \text{NARC} \leq \alpha \cdot \gamma \cdot (\gamma - 1) + \beta \cdot (\beta - 1) \quad (4.177) \]

![Figure 4.7](image-url) Illustration of Proposition 119(4.174). Graphs that achieve the maximum number of arcs according to a minimum number of vertices in a connected component, to a number of connected components, as well as to a fixed number of vertices (\( \text{MIN}_\text{NCC} = 2 \), \( \text{NCC} = 5 \), \( \text{NVERTEX} = 11 \), \( \text{NARC} = (11 - (5 - 1) \cdot 2)^2 + (5 - 1) \cdot 2^2 = 25 \)).

*Proof.* For proving inequality 4.174 we proceed by induction on the number of vertices of \( G \). First note that if all the connected components are reduced to one element the result is obvious. Thus we assume that the number of vertices in the maximal sized connected component of \( G \) is at least 2. Let \( x \) be an element of the maximal sized connected component of \( G \). Then, \( G - x \) satisfies \( \alpha(G - x) = \alpha(G) \), \( \gamma(G - x) = \gamma(G) \) and \( \beta(G - x) = \beta(G) - 1 \). Since by induction hypothesis \( |E(G - x)| \leq \alpha(G - x) \cdot \gamma(G - x)^2 + \beta(G - x)^2 \), and since the number of arcs of \( G \) incident to \( x \) is at most \( 2 \cdot (\beta(G) - 1) + 1 \), we have that \( |E(G)| \leq \alpha(G) \cdot \gamma(G)^2 + (\beta(G) - 1)^2 + 2 \cdot (\beta(G) - 1) + 1 \). And thus the result follows.
Proposition 120.
\[
\text{NARC} \leq \text{NCC} - 1 + (\text{NVERTEX} - \text{NSCC} + 1) \cdot (\text{NVERTEX} - \text{NCC} + 1)
+ \frac{(\text{NSCC} - \text{NCC} + 1) \cdot (\text{NSCC} - \text{NCC})}{2}
\]

Figure 4.8: Illustration of Proposition 120. A graph that achieves the maximum number of arcs according to a fixed number of connected components, to a fixed number of strongly connected components as well as to a fixed number of vertices (NSCC = 3, NCC = 6, NVERTEX = 7, NARC = 3 − 1 + (7 − 6 + 1) · (7 − 3 + 1) + \frac{(6−3+1)(6−3)}{2} = 18)

Proof. We proceed by induction on \(T(G) = \text{NVERTEX}(G) - |X| - (\text{NCC}(G) - 1)\), where \(X\) is any connected component of \(G\) of maximum cardinality. For \(T(G) = 0\) then either \(\text{NCC}(G) = 1\) and thus the formula is clearly true, by Proposition 4.145 or all the connected components of \(G\), but possibly \(X\), are reduced to one element. Since isolated vertices are not allowed, again by Proposition 4.145 applied on \(G[X]\), the formula holds indeed \(\text{NVERTEX}(G[X]) = \text{NVERTEX}(G) - (\text{NCC}(G) - 1)\) and \(\text{NSCC}(G[X]) = \text{NSCC}(G) - (\text{NCC}(G) - 1)\).

Assume that \(T(G) \geq 1\). Then there exists \(Y\), a connected component of \(G\) distinct from \(X\), with more than one vertex.

- Firstly assume that \(G[Y]\) is strongly connected. Let \(y \in Y\) and let \(G'\) be the graph such that \(V(G') = V(G)\) and \(E(G')\) is defined by:
  - For all \(Z\) connected components of \(G\) distinct from \(X\) and \(Y\) we have \(G'[Z] = G[Z]\).
  - With \(X' = X \cup \{y\}\) and \(Y' = \{y\}\), we have \(E(G'[Y']) = \{(y, y)\}\), \(E(G'[X']) = E(G[X]) \cup \{(z, x) : z \in Y - \{y\}, x \in X\} \cup \{(z, t) : z, t \in Y - \{y\}\}\).

Clearly we have that \(|E(G')| - |E(G)| \geq (|Y| - 1) \cdot |X| - 2 \cdot (|Y| - 1)\) and since \(|X| \geq |Y| \geq 2\), the difference is positive or null. Now as \(\text{NVERTEX}(G') = \text{NVERTEX}(G)\), \(\text{NCC}(G') = \text{NCC}(G)\), \(\text{NSCC}(G') = \text{NSCC}(G)\) (since \(G'[Y - \{y\}]\) is strongly connected because \(E(G'[Y - \{y\}]) = \{(z, t) : z, t \in Y - \{y\}\}\)) and since the reduced graph of the strongly connected components of \(G'[X']\) is exactly the reduced graph of the strongly connected components of \(G[X]\) to which a unique source has been added) and as \(T(G') \leq T(G) - 1\), the result holds by induction hypothesis.
4.3. GRAPH INVARIANTS

- Secondly assume that \( G[Y] \) is not strongly connected. Let \( Z \subset Y \) such that \( Z \) is a strongly connected component of \( G[Y] \) corresponding to a source in the reduced graph of the strongly connected components of \( G[Y] \). Let \( G' \) be the graph such that \( V(G') = V(G) \) and \( E(G') \) is defined by:
  
  - For all \( W \) connected components of \( G \) distinct from \( X \) and \( Y \) we have \( G'[W] = G[W] \).
  
  - With \( X' = X \cup Z \) and \( Y' = Y - Z \), we have \( E(G'[Y']) = E(G[Y']) \) if \( |Y'| > 1 \) and \( E(G'[Y']) = \{(y, y) \} \) if \( Y' = \{y\} \). \( E(G'[X']) = E(G[X]) \cup \{(z, x) : z \in Z, x \in X\} \).

Clearly we have that \(|E(G')| - |E(G)| \geq |Z| \cdot |X| - |Z| \cdot (|Y| - |Z|)\) and since \(|X| > |Y| - |Z|\), the difference is strictly positive. Now as \( \text{NVERTEX}(G') = \text{NVERTEX}(G), \text{NCC}(G') = \text{NCC}(G), \text{NSCC}(G') = \text{NSCC}(G) \) and as \( T(G') \leq T(G) - 1\), the result holds by induction hypothesis.

\[ \text{NARC} \geq \text{NVERTEX} = \max(0, \min(\text{NCC}, \text{NSCC} - \text{NCC})) \tag{4.179} \]

**Proof.** We prove that the invariant is valid for any digraph \( G \). First notice that for an operational behaviour, since we cannot assume that Proposition 55 (i.e., \( \text{NCC}(G) \leq \text{NSCC}(G) \)) was already triggered, we use the max operator. But since any strongly connected component is connected, then \( \text{NSCC}(G) - \text{NCC}(G) \) is never negative. Consequently we only show by induction on \( \text{NSCC}(G) \) that \( \text{NARC}(G) \geq \text{NVERTEX}(G) - \min(\text{NCC}(G), \text{NSCC}(G) - \text{NCC}(G)) \). To begin notice that if \( X \) is a strongly (non void) connected component then either \( \text{NARC}(G'[X]) \geq |X| \) or \( \text{NARC}(G[X]) = 0 \) and in this latter case we have that both \(|X| = 1\) and \( X \) is strictly included in a connected component of \( G \) (recall that isolated vertices are not allowed). Thus we can directly assume that \( \text{NSCC}(G) = k > 1 \).

First, consider that there exists a connected component of \( G \), say \( X \), which is also strongly connected. Let \( G' = G - X \), consequently we have \( \text{NSCC}(G') = \text{NSCC}(G) - 1, \text{NCC}(G') = \text{NCC}(G) - 1, \text{NVERTEX}(G') = \text{NVERTEX}(G) - |X|, \) and \( \text{NARC}(G) \geq |X| + \text{NARC}(G') \). Then \( \text{NARC}(G) \geq |X| + \text{NVERTEX}(G') - \min(\text{NCC}(G'), \text{NSCC}(G') - \text{NCC}(G')) \) and thus \( \text{NARC}(G) \geq \text{NVERTEX}(G') - \min(\text{NCC}(G') - 1, \text{NSCC}(G) - \text{NCC}(G)) \), which immediately gives the result.

Second consider that any strongly connected component is strictly included in a connected component of \( G \). Then, either there exists a strongly connected component \( X \) such that \(|X| \geq 2\). Let \( G' = G - X \), consequently we have \( \text{NSCC}(G') = \text{NSCC}(G) - 1, \text{NCC}(G') = \text{NCC}(G), \text{NVERTEX}(G') = \text{NVERTEX}(G) - |X|, \) and \( \text{NARC}(G) \geq |X| + 1 + \text{NARC}(G') \). Then \( \text{NARC}(G) \geq |X| + 1 + \text{NVERTEX}(G') - \min(\text{NCC}(G'), \text{NSCC}(G') - \text{NCC}(G')) \) and thus \( \text{NARC}(G) \geq \text{NVERTEX}(G) + 1 - \min(\text{NCC}(G'), \text{NSCC}(G) - \text{NCC}(G)) + 1 \), which immediately gives the result. Or, all the strongly connected components are reduced to one element, so we have \( \text{NSCC}(G) = \text{NVERTEX}(G) \), and thus we obtain that \( \text{NVERTEX}(G) - \min(\text{NCC}(G), \text{NSCC}(G) - \text{NCC}(G)) = \min(\text{NCC}(G), \text{NVERTEX}(G) - \text{NCC}(G)) \), which gives the result by for example Proposition 99 (4.143).

\[ \text{NARC} \geq \text{NVERTEX} = \max(0, \min(\text{NCC}, \text{NSCC} - \text{NCC})) \tag{4.179} \]

This bound is tight: take for example any circuit.
**Proposition 122.**

\[
\text{NARC} \leq \text{NVERTEX}^2 - \text{NVERTEX} \cdot \text{NSOURCE} - \text{NVERTEX} \cdot \text{NSINK} + \text{NSOURCE} \cdot \text{NSINK}
\]

(4.180)

**Proof.** Since the maximum number of arcs of a digraph is \(\text{NVERTEX}^2\), and since:

- No vertex can have a source as a successor we lose \(\text{NVERTEX} \cdot \text{NSOURCE}\) arcs,
- No sink can have a successor we lose \(\text{NVERTEX} \cdot \text{NSINK}\) arcs.

In these two sets of arcs we count twice the arcs from the sinks to the sources, so we finally get a maximum number of arcs corresponding to the right-hand side of the inequality to prove. \(\square\)
Graph invariants involving five parameters of a final graph

**Proposition 123.**

Let:

- \( \Delta = \text{NVERTEX} - \text{NCC} \cdot \text{MIN_NCC} \).
- \( \delta = \lfloor \frac{\Delta}{\max(1, \text{MAX_NCC} - \text{MIN_NCC})} \rfloor \).
- \( r = \Delta \mod \max(1, \text{MAX_NCC} - \text{MIN_NCC}) \).
- \( \epsilon = (r > 0) \).

\[
\Delta = 0 \lor (\text{MAX_NCC} \neq \text{MIN_NCC} \land \delta + \epsilon \leq \text{NCC}) \quad (4.181)
\]

\[
\text{NARC} \leq (\text{NCC} - \delta - \epsilon) \cdot \text{MIN_NCC}^2 + \epsilon \cdot (\text{MIN_NCC} + r)^2 + \delta \cdot \text{MAX_NCC}^2 
\quad (4.182)
\]

**Proposition 123 is currently a conjecture.**

**Proposition 124.**

\[
\text{NARC} \leq (\text{NCC} - 1) \cdot \max(1, (\text{MIN_NCC} - 1)) + (\text{NVERTEX} - \text{NSCC} + 1) \cdot (\text{NVERTEX} - \text{NCC} + 1) + (\text{NSCC} - \text{NCC} + 1) \cdot (\text{NSCC} - \text{NCC}) \quad (4.183)
\]

**Proposition 124 is currently a conjecture.**
CHAPTER 4. FURTHER TOPICS

Graph invariants relating two parameters of two final graphs

\[ \text{MAX}_1, \text{MIN}_1 \]

Proposition 125.

\[ \text{vpartition} \land \text{consecutive} \land \text{loops} \land \text{are} \land \text{connected} \land \text{MIN}_1 \notin [\text{NVERTEX}_{\text{INITIAL}} - \text{MAX}_1, \text{MAX}_1 - 1] \tag{4.184} \]

Proof. We show that the conjunction \( \text{MIN}_1 \geq \text{NVERTEX}_{\text{INITIAL}} - \text{MAX}_1 \) and \( \text{MIN}_1 \leq \text{MAX}_1 - 1 \) leads to a contradiction.

Since \( \text{MIN}_1 \leq \text{MAX}_1 - 1 \) we have that \( \text{MIN}_1 \neq \text{MAX}_1 \) and the minimum required size for the different groups is \( \text{MIN}_1 + 1 + \text{MAX}_1 \). This minimum required size should not exceed the number of vertices \( \text{NVERTEX}_{\text{INITIAL}} \) of the initial graph. But since, by hypothesis, \( \text{MIN}_1 \geq \text{NVERTEX}_{\text{INITIAL}} - \text{MAX}_1 \), this is impossible. \( \square \)

\[ \text{MAX}_2, \text{MIN}_2 \]

Proposition 126.

\[ \text{vpartition} \land \text{consecutive} \land \text{loops} \land \text{are} \land \text{connected} \land \text{MIN}_2 \notin [\text{NVERTEX}_{\text{INITIAL}} - \text{MAX}_2, \text{MAX}_2 - 1] \tag{4.185} \]

Proof. Similar to Proposition 125. \( \square \)

\[ \text{MAX}_1, \text{NCC}_2 \]

Proposition 127.

\[ \text{vpartition} : \text{MAX}_1 < \text{NVERTEX}_{\text{INITIAL}} \Leftrightarrow \text{NCC}_2 > 0 \tag{4.186} \]

\[ \text{apartition} : \text{MAX}_1 < \text{NVERTEX}_{\text{INITIAL}} \Leftrightarrow \text{NCC}_2 > 0 \tag{4.187} \]

Proof. (4.186) Since we have the precondition \( \text{vpartition} \), we know that each vertex of the initial graph belongs to the first or to the second final graphs (but not to both).

1. On the one hand, if the largest connected component of the first final graph cannot contain all the vertices of the initial graph, then the second final graph has at least one connected component.

2. On the other hand, if the second final graph has at least one connected component then the largest connected component of the first final graph cannot be equal to the initial graph.

(4.187) holds for a similar reason. \( \square \)

\[ \text{MAX}_2, \text{NCC}_1 \]

Proposition 128.

\[ \text{vpartition} : \text{MAX}_2 < \text{NVERTEX}_{\text{INITIAL}} \Leftrightarrow \text{NCC}_1 > 0 \tag{4.188} \]

\[ \text{apartition} : \text{MAX}_2 < \text{NVERTEX}_{\text{INITIAL}} \Leftrightarrow \text{NCC}_1 > 0 \tag{4.189} \]

Proof. Similar to Proposition 127. \( \square \)
4.3. GRAPH INVARIANTS

**MIN\_NCC\_1, NCC\_2**

**Proposition 129.**

\[ \text{vpartition} : \text{MIN}_{NCC\_1} < \text{NVERTEX}\_{INITIAL} \iff \text{NCC\_2} > 0 \] (4.190)

**Proof.** Since we have the precondition \text{vpartition}, we know that each vertex of the initial graph belongs to the first or to the second final graphs (but not to both).

1. On the one hand, if the smallest connected component of the first final graph cannot contain all the vertices of the initial graph, then the second final graph has at least one connected component.
2. On the other hand, if the second final graph has at least one connected component then the smallest connected component of the first final graph cannot be equal to the initial graph.

**MIN\_NCC\_2, NCC\_1**

**Proposition 130.**

\[ \text{vpartition} : \text{MIN}_{NCC\_2} < \text{NVERTEX}\_{INITIAL} \iff \text{NCC\_1} > 0 \] (4.191)

**Proof.** Similar to Proposition 129.

**NARC\_1, NARC\_2**

**Proposition 131.**

\[ \text{apartition} \land \text{arc}\_\text{gen} = \text{PATH} : \text{NARC}\_1 + \text{NARC}\_2 = \text{NVERTEX}\_{INITIAL} - 1 \] (4.192)

**Proof.** Holds since each arc of the initial graph belongs to one of the two final graphs and since the initial graph has \text{NVERTEX}\_{INITIAL} - 1 arcs.

**NCC\_1, NCC\_2**

**Proposition 132.**

\[ \text{apartition} \land \text{arc}\_\text{gen} = \text{PATH} : |\text{NCC}\_1 - \text{NCC}\_2| \leq 1 \] (4.193)

\[ \text{vpartition} \land \text{consecutive}\_\text{loops}\_\text{are}\_\text{connected} : |\text{NCC}\_1 - \text{NCC}\_2| \leq 1 \] (4.194)

**Proof.** Holds because the two initial graphs correspond to a path and because consecutive connected components do not come from the same graph constraint.

**Proposition 133.**

\[ \text{apartition} \land \text{arc}\_\text{gen} = \text{PATH} : \text{NCC}\_1 + \text{NCC}\_2 < \text{NVERTEX}\_{INITIAL} \] (4.195)

**Proof.** Holds because the initial graph is a path.

**NVERTEX\_1, NVERTEX\_2**

**Proposition 134.**

\[ \text{vpartition} : \text{NVERTEX}\_1 + \text{NVERTEX}\_2 = \text{NVERTEX}\_{INITIAL} \] (4.196)

**Proof.** By definition of \text{vpartition} each vertex of the initial graph belongs to one of the two final graphs (but not to both).
Graph invariants relating three parameters of two final graphs

\[ \text{MIN}_1, \text{MIN}_1, \text{MIN}_2 \]

**Proposition 135.**

\[ \text{apartition} \land \text{arc_gen} = \text{PATH} : \]

\[
\max(2, \text{MIN}_1) + \max(3, \text{MIN}_1 + 1, \text{MAX}_1) + \\
\max(2, \text{MIN}_2) - 2 > \text{NVERTEX}_{\text{INITIAL}} \Rightarrow \text{MIN}_1 = \text{MAX}_1
\]

(4.197)

**Proof.** The quantity \(\max(2, \text{MIN}_1) + \max(3, \text{MIN}_1 + 1, \text{MAX}_1) + \max(2, \text{MIN}_2) - 2\) corresponds to the minimum number of variables needed for building two non-empty connected components of respective size \(\text{MIN}_1\) and \(\text{MAX}_1\) such that \(\text{MAX}_1\) is strictly greater than \(\text{MIN}_1\). If this quantity is greater than the total number of variables we have that \(\text{MIN}_1 = \text{MAX}_1\). \(\Box\)

**Proposition 136.**

\[ \text{vpartition} \land \text{consecutive_loops_are_connected} : \]

\[
\max(1, \text{MIN}_1) + \max(2, \text{MIN}_1 + 1, \text{MAX}_1) + \\
\max(1, \text{MIN}_2) > \text{NVERTEX}_{\text{INITIAL}} \Rightarrow \text{MIN}_1 = \text{MAX}_1
\]

(4.198)

**Proof.** The quantity \(\max(1, \text{MIN}_1) + \max(2, \text{MIN}_1 + 1, \text{MAX}_1) + \max(1, \text{MIN}_2)\) corresponds to the minimum number of variables needed for building two non-empty connected components of respective size \(\text{MIN}_1\) and \(\text{MAX}_1\) such that \(\text{MAX}_1\) is strictly greater than \(\text{MIN}_1\). If this quantity is greater than the total number of variables we have that \(\text{MIN}_1 = \text{MAX}_1\). \(\Box\)

**Proposition 137.**

\[ \text{vpartition} \land \text{consecutive_loops_are_connected} : \]

\[
\text{MIN}_2 \notin \left[ \max \left( \text{NVERTEX}_{\text{INITIAL}} - \text{MAX}_1 - \text{MIN}_1 + 1, \right. \\
\left. \left( \text{NVERTEX}_{\text{INITIAL}} - \text{MAX}_1 + 2 \right) / 2 \right), \\
\text{NVERTEX}_{\text{INITIAL}} - \text{MAX}_1 - 1 \right]
\]

(4.199)

**Proof.** A value \(v\) is not a possible number of vertices for the smallest connected component of type 2 if the following two conditions hold:

- \(v + \text{MAX}_1\) does not allow to cover all the vertices of the initial graph: we need at least one extra connected component of type 1 or 2.
- If we add an additional connected component of type 1 or 2 we exceed the number of vertices of the initial graph.

\(\Box\)
4.3. **GRAPH INVARIANTS**

**Proposition 138.**

\[ \text{apartition} \land \text{arc-gen} = \text{PATH} : \]
\[ \max(2, \text{MIN}_2) + \max(3, \text{MIN}_2 + 1, \text{MAX}_2) + \]
\[ \max(2, \text{MIN}_1) - 2 > \text{NVERTEX}_{\text{INITIAL}} \Rightarrow \text{MIN}_2 = \text{MAX}_2 \]

(4.200)

**Proof.** Similar to Proposition 135.

**Proposition 139.**

\[ \text{vpartition} \land \text{consecutive_loops_are_connected} : \]
\[ \max(1, \text{MIN}_2) + \max(2, \text{MIN}_2 + 1, \text{MAX}_2) + \]
\[ \max(1, \text{MIN}_1) > \text{NVERTEX}_{\text{INITIAL}} \Rightarrow \text{MIN}_2 = \text{MAX}_2 \]

(4.201)

**Proof.** Similar to Proposition 136.

**Proposition 140.**

\[ \text{vpartition} \land \text{consecutive_loops_are_connected} : \]
\[ \text{MIN}_1 \notin \left[ \max \left( \text{NVERTEX}_{\text{INITIAL}} - \text{MAX}_2 - \text{MIN}_2 + 1, \right. \]
\[ \left. \frac{\text{NVERTEX}_{\text{INITIAL}} - \text{MAX}_2 + 2}{2} \right] \right. \]
\[ \text{NVERTEX}_{\text{INITIAL}} - \text{MAX}_2 - 1 \]

(4.202)

**Proof.** Similar to Proposition 137.

**Proposition 141.**

\[ \text{vpartition} : \text{MIN}_1 = \text{MAX}_1 \land \text{MIN}_1 \mod 2 = 0 \Rightarrow \]
\[ \text{NVERTEX}_2 \mod 2 = \text{NVERTEX}_{\text{INITIAL}} \mod 2 \]

(4.203)

**Proof.** If the number of vertices of the first graph is even then the number of vertices of the second graph has the same parity as the number of vertices of the initial graph (since a vertex of the initial graph belongs either to the first graph, either to the second graph (but not to both).

**Proposition 142.**

\[ \text{vpartition} : \text{MIN}_2 = \text{MAX}_2 \land \text{MIN}_2 \mod 2 = 0 \Rightarrow \]
\[ \text{NVERTEX}_1 \mod 2 = \text{NVERTEX}_{\text{INITIAL}} \mod 2 \]

(4.204)

**Proof.** Similar to Proposition 141.
Proposition 143.

apartition $\land$ arc_gen $= PATH \land NVERTEX_{\text{INITIAL}} > 0 :$

\begin{align*}
NCC_1 &= 1 \iff \text{MIN}_{NCC_1} + \text{NARC}_2 = NVERTEX_{\text{INITIAL}} \\
\end{align*}

(4.205)

Proof. When $\text{MIN}_{NCC_1} + \text{NARC}_2 = NVERTEX_{\text{INITIAL}}$ there is no more room for an extra connected component for the first final graph.

Proposition 144.

apartition $\land$ arc_gen $= PATH \land NVERTEX_{\text{INITIAL}} > 0 :$

\begin{align*}
NCC_2 &= 1 \iff \text{MIN}_{NCC_2} + \text{NARC}_1 = NVERTEX_{\text{INITIAL}} \\
\end{align*}

(4.206)

Proof. Similar to Proposition 143.
Graph invariants relating four parameters of two final graphs

**Proposition 145.**

\[
\text{apartition} \land \text{arc.gen} = \text{PATH} : \\
\max(2, \text{MIN.NCC}_1) + \max(2, \text{MAX.NCC}_1) + \max(2, \text{MIN.NCC}_2) - 2 > \\
\text{NVERTEX}_\text{INITIAL} \Rightarrow \text{NCC}_1 \leq 1 \\
\] (4.207)

**Proof.** The quantity \(\max(2, \text{MIN.NCC}_1) + \max(2, \text{MAX.NCC}_1) + \max(2, \text{MIN.NCC}_2) - 2\) corresponds to the minimum number of variables needed for building two non-empty connected components of respective size \(\text{MIN.NCC}_1\) and \(\text{MAX.NCC}_1\). If this quantity is greater than the total number of variables we have that \(\text{NCC}_1 \leq 1\).

**Proposition 146.**

\[
\text{vpartition} \land \text{consecutive_loops.\ are.\ connected} : \\
\max(1, \text{MIN.NCC}_1) + \max(1, \text{MAX.NCC}_1) + \max(1, \text{MIN.NCC}_2) > \\
\text{NVERTEX}_\text{INITIAL} \Rightarrow \text{NCC}_1 \leq 1 \\
\] (4.208)

**Proof.** The quantity \(\max(1, \text{MIN.NCC}_1) + \max(1, \text{MAX.NCC}_1) + \max(1, \text{MIN.NCC}_2)\) corresponds to the minimum number of variables needed for building two non-empty connected components of respective size \(\text{MIN.NCC}_1\) and \(\text{MAX.NCC}_1\). If this quantity is greater than the total number of variables we have that \(\text{NCC}_1 \leq 1\).

**Proposition 147.**

\[
\text{apartition} \land \text{arc.gen} = \text{PATH} : \\
\max(2, \text{MIN.NCC}_2) + \max(2, \text{MAX.NCC}_2) + \max(2, \text{MIN.NCC}_1) - 2 > \\
\text{NVERTEX}_\text{INITIAL} \Rightarrow \text{NCC}_2 \leq 1 \\
\] (4.209)

**Proof.** Similar to Proposition 145.

**Proposition 148.**

\[
\text{vpartition} \land \text{consecutive_loops.\ are.\ connected} : \\
\max(1, \text{MIN.NCC}_2) + \max(1, \text{MAX.NCC}_2) + \max(1, \text{MIN.NCC}_1) > \\
\text{NVERTEX}_\text{INITIAL} \Rightarrow \text{NCC}_2 \leq 1 \\
\] (4.210)

**Proof.** Similar to Proposition 146.
Proposition 149.

\[ \min \text{NCC}_2 \notin \left[ \left\lfloor \frac{\text{NVERTEX}_2}{2} \right\rfloor + 1, \text{NVERTEX}_{\text{INITIAL}} - \min \text{NCC}_1 - \max \text{NCC}_1 - 1 \right] \]

(4.211)

Proof. First, note that, when \( \text{NCC}_2 > 1 \), we have that \( \min \text{NCC}_2 \leq \left\lfloor \frac{\text{NVERTEX}_2}{2} \right\rfloor \).
Second, note that, when \( \text{NCC}_2 \leq 1 \), we have that \( \min \text{NCC}_2 \geq \text{NVERTEX}_{\text{INITIAL}} - \min \text{NCC}_1 - \max \text{NCC}_1 \). Since \( \text{NCC}_2 \) has to have at least one value the result follows.

Proposition 150.

\[ \min \text{NCC}_1 \notin \left[ \left\lfloor \frac{\text{NVERTEX}_1}{2} \right\rfloor + 1, \text{NVERTEX}_{\text{INITIAL}} - \min \text{NCC}_2 - \max \text{NCC}_2 - 1 \right] \]

(4.212)

Proof. Similar to Proposition 149.
4.3. GRAPH INVARIANTS

Graph invariants relating five parameters of two final graphs

\[ \text{MAX\_NCC}_1, \text{MAX\_NCC}_2, \text{MIN\_NCC}_1, \text{MIN\_NCC}_2, \text{NCC} \]

**Proposition 151.**

\[ \text{vpartition} \land \text{consecutive\_loops\_are\_connected} : \]

\[ \text{MIN\_NCC}_1 \cdot \max(0, \text{NCC}_1 - 1) + \text{MAX\_NCC}_1 + \]

\[ \text{MIN\_NCC}_2 \cdot \max(0, \text{NCC}_1 - 2) + \text{MAX\_NCC}_2 \leq \text{NVERTEX\_INITIAL} \tag{4.213} \]

**Proof.** The left-hand side of 151 corresponds to the minimum number of vertices of the two final graphs provided that we build the smallest possible connected components.

**Proposition 152.**

\[ \text{vpartition} \land \text{consecutive\_loops\_are\_connected} : \]

\[ \text{NCC}_1 \leq (\text{MAX\_NCC}_2 > 0) + \left\lceil \frac{\alpha}{\beta} \right\rceil + (\alpha \mod \beta \geq \max(1, \text{MIN\_NCC}_1)) \]

\[ \begin{cases} 
\alpha = \max(0, \text{NVERTEX\_INITIAL} - \max(1, \text{MAX\_NCC}_1) - \max(1, \text{MAX\_NCC}_2)), \\
\beta = \max(1, \text{MIN\_NCC}_1) + \max(1, \text{MIN\_NCC}_2). 
\end{cases} \tag{4.214} \]

**Proof.** The maximum number of connected components is achieved by building non-empty groups as small as possible, except for two groups of respective size \(\max(1, \text{MAX\_NCC}_1)\) and \(\max(1, \text{MAX\_NCC}_2)\), which have to be built.

**Proposition 153.**

\[ \text{vpartition} \land \text{consecutive\_loops\_are\_connected} : \]

\[ \text{MAX\_NCC}_1 \cdot \max(0, \text{NCC}_1 - 1) + \text{MIN\_NCC}_1 + \]

\[ \text{MAX\_NCC}_2 \cdot \text{NCC}_1 + \text{MIN\_NCC}_2 \geq \text{NVERTEX\_INITIAL} \tag{4.215} \]

**Proof.** The left-hand side of 153 corresponds to the maximum number of vertices of the two final graphs provided that we build the largest possible connected components.

**Proposition 154.**

\[ \text{vpartition} \land \text{consecutive\_loops\_are\_connected} : \]

\[ \text{NCC}_1 \geq (\text{MAX\_NCC}_2 < \text{NVERTEX\_INITIAL}) + \left\lceil \frac{\alpha}{\beta} \right\rceil + (\alpha \mod \beta > \text{MAX\_NCC}_2) \]

\[ \begin{cases} 
\alpha = \min(0, \text{NVERTEX\_INITIAL} - \text{MIN\_NCC}_1 - \text{MIN\_NCC}_2), \\
\beta = \max(1, \text{MAX\_NCC}_1) + \max(1, \text{MAX\_NCC}_2). 
\end{cases} \tag{4.216} \]

**Proof.** The minimum number of connected components is achieved by taking the groups as large as possible except for two groups of respective size \(\text{MIN\_NCC}_2\) and \(\text{MIN\_NCC}_1\), which have to be built.
Proposition 155.

\[ v\text{partition} \land \text{consecutive\_loops\_are\_connected} : \]
\[ \text{MAX\_NCC}_2 \leq \max(\text{MIN\_NCC}_2, \text{NVERTEX}_{\text{INITIAL}} - \alpha), \text{with} : \]
\[ \bullet \alpha = \text{MIN\_NCC}_1 \cdot \max(0, \text{NCC}_1 - 1) + \text{MAX\_NCC}_1 \]
\[ \text{MIN\_NCC}_2 + \text{MIN\_NCC}_2 \cdot \max(0, \text{NCC}_1 - 3) \]
\[ (4.217) \]

**Proof.** If \( \text{NCC}_1 \leq 1 \) we have that \( \text{MAX\_NCC}_2 \leq \text{MIN\_NCC}_2 \). Otherwise, when \( \text{NCC}_1 > 1 \), we have that \( \text{MIN\_NCC}_1 \cdot \max(0, \text{NCC}_1 - 1) + \text{MAX\_NCC}_1 + \text{MIN\_NCC}_2 + \text{MAX\_NCC}_2 + \text{MIN\_NCC}_2 \cdot \max(0, \text{NCC}_1 - 3) \leq \text{NVERTEX}_{\text{INITIAL}}. \)
\( \text{NCC}_1 - 3 \) comes from the fact that we build the minimum number of connected components in the second final graph (i.e., \( \text{NCC}_1 - 1 \) connected components) and that we have already built two connected components of respective size \( \text{MIN\_NCC}_2 \) and \( \text{MAX\_NCC}_2 \). By isolating \( \text{MAX\_NCC}_2 \) in the previous expression and by grouping the two inequalities the result follows.

Proposition 156.

\[ \text{apartition} \land \text{arc\_gen} = \text{PATH} \land \text{MIN\_NCC}_1 > 1 \land \text{MIN\_NCC}_2 > 1 : \]
\[ \text{NCC}_1 \leq (\text{MAX\_NCC}_1 > 0) + \left\lceil \frac{\alpha}{\beta} \right\rceil + ((\alpha \mod \beta) + 1) \geq \text{MIN\_NCC}_1, \text{with} : \]
\[ \begin{cases} 
\bullet \alpha = \max(0, \text{NVERTEX}_{\text{INITIAL}} - \text{MAX\_NCC}_1 - \text{MAX\_NCC}_2 + 1), \\
\bullet \beta = \text{MIN\_NCC}_1 + \text{MIN\_NCC}_2 - 2.
\end{cases} \]
\[ (4.218) \]

Figure 4.9: Illustration of Proposition 156. Configuration achieving the maximum number of connected components for \( G_1 \) according to the size of the smallest and largest connected components of \( G_1 \) and \( G_2 \) and to an initial number of vertices \( \text{MAX\_NCC}_1 = 4, \text{MAX\_NCC}_2 = 5, \text{MIN\_NCC}_1 = 3, \text{MIN\_NCC}_2 = 4, \text{NVERTEX}_{\text{INITIAL}} = 14, \alpha = \max(0, 14 - 4 - 5 + 1) = 6, \beta = \max(2, 3 + 4 - 2) = 5, \text{NCC}_1 = (4 > 0) + \left\lceil \frac{6}{5} \right\rceil + ((6 \mod 5) + 1) \geq 3) = 2 \)

**Proof.** The maximum number of connected components of \( G_1 \) is achieved by:

- Building a first connected component of \( G_1 \) involving \( \text{MAX\_NCC}_1 \) vertices,
- Building a first connected component of \( G_2 \) involving \( \text{MAX\_NCC}_2 \) vertices,
- Building alternatively a connected component of \( G_1 \) and a connected component of \( G_2 \) involving respectively \( \text{MIN\_NCC}_1 \) and \( \text{MIN\_NCC}_2 \) vertices,
Finally, if this is possible, building a connected component of \( G_1 \) involving \( \text{MIN}_N\text{CC}_1 \) vertices.

**Proposition 157.**

\[
\text{apartition} \land \text{arc_gen} = \text{PATH} \land \text{MIN}_N\text{CC}_1 > 1 \land \text{MIN}_N\text{CC}_2 > 1:
\]

\[
\text{NCC}_1 \geq (\text{MIN}_N\text{CC}_1 > 0) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + ((\alpha \mod \beta) + 1 > \text{MAX}_N\text{CC}_2), \text{ with }:
\]

\[
\begin{cases} 
\alpha = \max(0, \text{NVERTEX}_{\text{INITIAL}} - \text{MIN}_N\text{CC}_1 - \text{MIN}_N\text{CC}_2 + 1), \\
\beta = \text{MAX}_N\text{CC}_1 + \text{MAX}_N\text{CC}_2 - 2.
\end{cases}
\]

(4.219)

![Graphs](image)

Figure 4.10: Illustration of Proposition 157. Configuration achieving the minimum number of connected components for \( G_1 \) according to the size of the smallest and largest connected components of \( G_1 \) and \( G_2 \) and to an initial number of vertices (\( \text{MAX}_N\text{CC}_1 = 4, \text{MAX}_N\text{CC}_2 = 5, \text{MIN}_N\text{CC}_1 = 3, \text{MIN}_N\text{CC}_2 = 4, \text{NVERTEX}_{\text{INITIAL}} = 18, \alpha = \max(0, 18 - 3 - 4 + 1) = 12, \beta = \max(2, 4 + 5 - 2) = 7, \text{NCC}_1 = (3 > 0) + \left\lfloor \frac{12}{7} \right\rfloor + (((12 \mod 7) + 1) > 5) = 3)\)

**Proof.** The minimum number of connected components of \( G_1 \) is achieved by:

- Building a first connected component of \( G_2 \) involving \( \text{MIN}_N\text{CC}_2 \) vertices,
- Building a first connected component of \( G_1 \) involving \( \text{MIN}_N\text{CC}_1 \) vertices,
- Building alternatively a connected component of \( G_2 \) and a connected component of \( G_1 \) involving respectively \( \text{MAX}_N\text{CC}_2 \) and \( \text{MAX}_N\text{CC}_1 \) vertices,
- Finally, if this is possible, building a connected component of \( G_2 \) involving \( \text{MAX}_N\text{CC}_2 \) vertices and a connected component of \( G_1 \) with the remaining vertices.

Note that these remaining vertices cannot be incorporated in the connected components previously built.

\[\square\]
Proposition 158.

\[\text{min} \cdot \max(0, \text{NCC}_2 - 1) + \text{max} \cdot \max(0, \text{NCC}_2 - 2) + \text{MAX} \cdot \text{MIN} \leq \text{NVERTEX}_{\text{INITIAL}} \]

**Proof.** Similar to Proposition 151.

Proposition 159.

\[\text{NCC}_2 \leq (\text{MAX} > 0) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + (\alpha \mod \beta > \text{MAX})\]

\[
\begin{cases}
\alpha &= \max(0, \text{NVERTEX}_{\text{INITIAL}} - \max(1, \text{MAX})), \\
\beta &= \max(1, \text{MIN} + \max(1, \text{MIN})).
\end{cases}
\]

**Proof.** Similar to Proposition 152.

Proposition 160.

\[\text{MAX} \cdot \max(0, \text{NCC}_2 - 1) + \text{MIN} \cdot \text{MIN} + \text{MAX} \cdot \text{MIN} \geq \text{NVERTEX}_{\text{INITIAL}} \]

**Proof.** Similar to Proposition 153.

Proposition 161.

\[\text{NCC}_2 \geq (\text{MAX} < \text{NVERTEX}_{\text{INITIAL}}) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + (\alpha \mod \beta > \text{MAX})\]

\[
\begin{cases}
\alpha &= \max(0, \text{NVERTEX}_{\text{INITIAL}} - \min - \text{MIN} + \max(1, \text{MAX})), \\
\beta &= \max(1, \text{MIN} + \max(1, \text{MIN})).
\end{cases}
\]

**Proof.** Similar to Proposition 154.

Proposition 162.

\[\text{MAX} \leq \max(\text{MIN}, \text{NVERTEX}_{\text{INITIAL}} - \alpha)\]

\[
\begin{cases}
\alpha &= \text{MIN} \cdot \max(0, \text{NCC}_2 - 1) + \text{MAX} + \text{MIN} \cdot \max(0, \text{NCC}_2 - 3)
\end{cases}
\]

**Proof.** Similar to Proposition 155.
4.3. GRAPH INVARIANTS

Proposition 163.

\[
\text{apartition} \land \text{arc.gen} = \text{PATH} \land \text{MIN.NCC}_1 > 1 \land \text{MIN.NCC}_2 > 1 : \\
\text{NCC}_2 \leq (\text{MAX.NCC}_2 > 0) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + (\alpha \mod \beta + 1 \geq \text{MIN.NCC}_2), \text{ with:}
\]
\[
\begin{align*}
\bullet & \quad \alpha = \max(0, \text{NVERTEX}_\text{INITIAL} - \text{MAX.NCC}_1 - \text{MAX.NCC}_2 + 1), \\
\bullet & \quad \beta = \text{MIN.NCC}_1 + \text{MIN.NCC}_2 - 2.
\end{align*}
\]

\textit{Proof.} Similar to Proposition 156.

Proposition 164.

\[
\text{apartition} \land \text{arc.gen} = \text{PATH} \land \text{MIN.NCC}_1 > 1 \land \text{MIN.NCC}_2 > 1 : \\
\text{NCC}_2 \geq (\text{MIN.NCC}_2 > 0) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + (\alpha \mod \beta + 1 \geq \text{MAX.NCC}_1), \text{ with:}
\]
\[
\begin{align*}
\bullet & \quad \alpha = \max(0, \text{NVERTEX}_\text{INITIAL} - \text{MIN.NCC}_1 - \text{MIN.NCC}_2 + 1), \\
\bullet & \quad \beta = \text{MAX.NCC}_1 + \text{MAX.NCC}_2 - 2.
\end{align*}
\]

\textit{Proof.} Similar to Proposition 157.
Graph invariants relating six parameters of two final graphs

\[ \text{MAX}_1, \text{MAX}_2, \text{MIN}_1, \text{MIN}_2, \text{NCC}_1, \text{NCC}_2 \]

Proposition 165.

apartition \( \land \) arc_gen = PATH \( \land \) NVERTEX\_INITIAL > 0:

\[
\begin{align*}
\alpha \cdot \text{MIN}_1 + \text{MAX}_1 + \\
\beta \cdot \text{MIN}_2 + \text{MAX}_2 \leq N\text{VERTEX}\_\text{INITIAL} + \text{NCC}_1 + \text{NCC}_2 - 1, \text{ with }:
\end{align*}
\]

\[
\begin{align*}
\bullet & \quad \alpha = \max(0, \text{NCC}_1 - 1), \\
\bullet & \quad \beta = \max(0, \text{NCC}_2 - 1). 
\end{align*}
\]

(4.227)

Proof. Let \( CC(G_1) = \{ CC_1^a : a \in [\text{NCC}_1] \} \) and \( CC(G_2) = \{ CC_2^a : a \in [\text{NCC}_2] \} \) be respectively the set of connected components of the first and the second final graphs. Since the initial graph is a path, and since each arc of the initial graph belongs to the first or to the second final graphs (but not to both), there exists \( (A_i)_{i \in [\text{NCC}_1 + \text{NCC}_2]} \) and there exists \( j \in [2] \) such that \( A_i \in CC(G_1 + (j \mod 2)) \), for \( i \mod 2 = 0 \) and \( A_i \in CC(G_1 + ((j+1) \mod 2)) \) for \( i \mod 2 = 1 \) and \( A_i \cap A_{i+1} \neq \emptyset \) for every well defined \( i \).

By inclusion-exclusion principle, since \( A_i \cap A_j = \emptyset \) whenever \( j \neq i + 1 \), we obtain \( N\text{VERTEX}\_\text{INITIAL} = \sum_{a \in [\text{NCC}_1]} |CC_1^a| + \sum_{a \in [\text{NCC}_2]} |CC_2^a| - \sum_{a \in [\text{NCC}_1 + \text{NCC}_2]} |A_i \cap A_{i+1}|. \) Since \( |A_i \cap A_{i+1}| \) is equal to 1 for every well defined \( i \), we obtain \( \sum_{a \in [\text{NCC}_1]} |CC_1^a| + \sum_{a \in [\text{NCC}_2]} |CC_2^a| = N\text{VERTEX}\_\text{INITIAL} + \text{NCC}_1 + \text{NCC}_2 - 1. \)

Since \( \alpha \cdot \text{MIN}_1 + \text{MAX}_1 + \beta \cdot \text{MIN}_2 + \text{MAX}_2 \leq \sum_{a \in [\text{NCC}_1]} |CC_1^a| + \sum_{a \in [\text{NCC}_2]} |CC_2^a| \) the result follows.

\( \square \)

Proposition 166.

apartition \( \land \) arc_gen = PATH \( \land \) NVERTEX\_INITIAL > 0:

\[
\begin{align*}
\alpha \cdot \text{MAX}_1 + \text{MIN}_1 + \\
\beta \cdot \text{MIN}_2 + \text{MAX}_2 \geq N\text{VERTEX}\_\text{INITIAL} + \text{NCC}_1 + \text{NCC}_2 - 1, \text{ with }:
\end{align*}
\]

\[
\begin{align*}
\bullet & \quad \alpha = \max(0, \text{NCC}_1 - 1), \\
\bullet & \quad \beta = \max(0, \text{NCC}_2 - 1). 
\end{align*}
\]

(4.228)

Proof. Similar to Proposition 165.

\( \square \)
4.4. The electronic version of the catalogue

4.4.1 Prolog facts describing a constraint

An electronic version of the catalogue containing every global constraint of the catalogue is given in Appendix B. In addition, the entry “Utilities” contains a set of shared utilities used for evaluating the constraints. This electronic version was used for generating the \LaTeX\ file of this catalogue, the figures associated with the graph-based description and a filtering algorithm for some of the constraints that use the automaton-based description. Within the electronic version, each constraint is described in terms of meta-data. A typical entry is:
CHAPTER 4. FURTHER TOPICS

\begin{verbatim}
ctr_date(minimum, ['20000128','20030820','20040530','20041230','20060811','20090416']).
ctr_origin(minimum, '\index{CHIP|indexuse}CHIP', []).
ctr_arguments(minimum, ['MIN'-dvar, 'VARIABLES'-collection{var-dvar}]).
ctr_exchangeable(minimum, [items('VARIABLES',all),
                      vals(['VARIABLES'°var],int,=,all,in),
                      translate(['MIN','VARIABLES'°var])]).
ctr_synonyms(minimum, [min]).
ctr_restrictions(minimum, [size('VARIABLES') > 0, required('VARIABLES',var)]).
ctr_graph(minimum,
          ['VARIABLES'],
          2,
          ['CLIQUE'>>collection(variables1,variables2)],
          [variables1'°key = variables2'°key \/ variables1°var < variables2°var],
          ['ORDER'(0,'MAXINT',var) = 'MIN'],
          []).
ctr_example(minimum, minimum(2,[[var-3],[var-2],[var-7],[var-2],[var-6]])).
ctr_see_also(minimum,
             [link('generalisation', minimum_modulo,
                 '°e replaced by °e', [variable, variable mod constant]),
                 link('specialisation', min_n,
                 'minimum or order °e replaced by absolute minimum', [n]),
                 link('comparison swapped', maxint, '', []),
                 link('common keyword', maxint, '', []),
                 link('soft variant', open_minimum, '°k', ['open constraint']),
                 link('soft variant', minimum_except_0, '°e is ignored', [0]),
                 link('implies', between_min_max, '', []),
                 link('implies', and, '', []),
                 link('implied by', or, '', [])]).
ctr_key_words(minimum,['order constraint' ,
                      'minimum' ,
                      'maxint' ,
                      'automaton' ,
                      'automaton without counters' ,
                      'reified automaton constraint' ,
                      'centered cyclic(1) constraint network(1)' ,
                      'arc-consistency' ]).
ctr_persons(minimum,['Beldiceanu N.']).
ctr_eval(minimum, [builtin(minimum_b), automaton(minimum_a)]).
minimum_b(MIN, VARIABLES) :-
  check_type(dvar, MIN),
  collection(VARIABLES, [dvar]),
  length(VARIABLES, N),
  N > 0,
  get_attr1(VARIABLES, VARS),
  minimum(MIN, VARS).
minimum_a(MIN, VARIABLES) :- % 0: MIN<VAR, 1: MIN=VAR, 2: MIN>VAR
  minimum_signature(VARIABLES, SIGNATURE, MIN),
  automaton(SIGNATURE, _, SIGNATURE,
            [source(s),sink(t)],
            [arc(s,0,s),arc(s,1,t),arc(t,1,t),arc(t,0,t)],
            [],[],[]).
minimum_signature([], [], _).
minimum_signature([[var-VAR]|VARS], [$|SS], MIN) :-
  S in 0..2,
  MIN #< VAR #<=> S = 0, MIN #= VAR #<=> S = 1, MIN #> VAR #<=> S = 2,
  minimum_signature(VARS, SS, MIN).
\end{verbatim}
and consists of the following Prolog facts, where \texttt{CONSTRAINT\_NAME} is the name of the constraint under consideration. The facts are organised in the following 15 items:

- Items 1, 2, 3, 4, 12 and 13 provide general information about a global constraint,
- Items 5, 6 and 7 describe the arguments of a global constraint.
- Items 9 and 10 describes the meaning of a global constraint in terms of a graph-based representation.
- Item 11 provides a ground instance which holds.
- Item 14 gives the list of available evaluators of a global constraint.
- Item 15 describes the meaning of a global constraint in terms of a set of first order logic formulae.

Items 1, 2, 6 and 11 are mandatory, while all other items are optional. We now give the different items:

1. \texttt{ctr\_date( CONSTRAINT\_NAME, LIST\_OF\_DATES\_OF\_MODIFICATIONS )}
   - \texttt{LIST\_OF\_DATES\_OF\_MODIFICATIONS} is a list of dates when the description of the constraint was modified.

2. \texttt{ctr\_origin( CONSTRAINT\_NAME, STRING, LIST\_OF\_CONSTRAINTS\_NAMES )}
   - \texttt{STRING} is a string denoting the origin of the constraint. \texttt{LIST\_OF\_CONSTRAINTS\_NAMES} is a possibly empty list of constraint names related to the origin of the constraint.

3. \texttt{ctr\_usual\_name( CONSTRAINT\_NAME, USUAL\_NAME )}
   - When, for some reason, the constraint name used in the catalogue does not correspond to the usual name of the constraint, \texttt{USUAL\_NAME} provides the usual name of the constraint. This stems from the fact that each entry of the catalogue should have a distinct name. This is for instance the case for the \texttt{stretch\_path} and the \texttt{stretch\_circuit} constraints which are both usually called \texttt{stretch}.

4. \texttt{ctr\_synonyms( CONSTRAINT\_NAME, LIST\_OF\_SYNONYMS )}
   - \texttt{LIST\_OF\_SYNONYMS} is a list of synonyms for the constraint. This stems from the fact that, quite often, different authors use a different name for the same constraint. This is for instance the case for the \texttt{alldifferent} and the \texttt{symmetric\_alldifferent} constraints.
5. **ctr_types** (CONSTRAINT NAME, LIST_OF_TYPES_DECLARATIONS)
   - LIST_OF_TYPES_DECLARATIONS is a list of elements of the form name-type, where name is the name of a new type and type the type itself (usually a collection). Basic and compound data types were respectively introduced in sections 2.1.1 and 2.1.2 on page 6. This field is only used when we need to declare a new type that will be used for specifying the type of the arguments of the constraint. This is for instance the case when one argument of the constraint is a collection for which the type of one attribute is also a collection. This is for instance the case for the `diffn` constraint where the unique argument ORTHOTOPES is a collection of ORTHOTOPE; ORTHOTOPE refers to a new type declared in LIST_OF_TYPES_DECLARATIONS.

6. **ctr_arguments** (CONSTRAINT NAME, LIST_OF_ARGUMENTS_DECLARATIONS)
   - LIST_OF_ARGUMENTS_DECLARATIONS is a list of elements of the form arg-type, where arg is the name of an argument of the constraint and type the type of the argument. Basic and compound data types were respectively introduced in sections 2.1.1 and 2.1.2 on page 6.

7. **ctr_restrictions** (CONSTRAINT NAME, LIST_OF_RESTRICTIONS)
   - LIST_OF_RESTRICTIONS is a list of restrictions on the different argument of the constraint. Possible restrictions were described in Section 2.1.3 on page 9.

8. **ctr_exchangeable** (CONSTRAINT_NAME, LIST_OF_SYMMETRIES)
   - LIST_OF_SYMMETRIES is a list of mappings preserving the solutions of the constraint. Possible mappings were described in Section 2.1.5 on page 18.

9. **ctr-derived_collections** (CONSTRAINT_NAME, LIST_OF_DERIVED_Collections)
    - LIST_OF_DERIVED_Collections is a list of derived collections. Derived collections are collections that are computed from the arguments of the constraint and are used in the graph-based description. Derived collections were described in Section 2.2.2 on page 42.

10. **ctr-graph** (CONSTRAINT_NAME, LIST_OF_ARC_INPUT, ARC_ARITY, ARC_GENERATORS, ARC_CONSTRAINTS, GRAPH_PROPERTIES)
    - LIST_OF_ARC_INPUT is a list of collections used for creating the vertices of the initial graph. This was described at page 70 of Section 2.2.3.
    - ARC_ARITY is the number of vertices of an arc. Arc arity was explained at page 72 of Section 2.2.3.
    - ARC_GENERATORS is a list of arc generators. Arc generators were introduced at page 71 of Section 2.2.3.
    - ARC_CONSTRAINTS is a list of arc constraints. Arc constraints were defined in Section 2.2.2 on page 48.
    - GRAPH_PROPERTIES is a list of graph properties. Graph properties were described in Section 2.2.2 on page 57.
4.4. THE ELECTRONIC VERSION OF THE CATALOGUE

11. ctr_example(CONSTRAINT_NAME, LIST_OF_EXAMPLES)

- LIST_OF_EXAMPLES is a list of examples (usually one). Each example corresponds to a ground instance for which the constraint holds.

12. ctr_see_also(CONSTRAINT_NAME, LIST_OF_CONSTRAINTS)

- LIST_OF_CONSTRAINTS is a list of constraints that are related in some way to the constraint. Each element of the list is a fact of the form link(TYPE_OF_LINK, CONSTRAINT, STRING, SYMBOLS), where:
  - TYPE_OF_LINK is a semantic link that explains why we refer to CONSTRAINT.
  - CONSTRAINT is the name of the constraint that is linked to CONSTRAINT_NAME.
  - STRING is a string providing contextual explanation.
  - SYMBOLS is a list of symbols (e.g., keywords, constraint names, mathematical expressions) that are inserted in STRING.

13. ctr_keywords(CONSTRAINT_NAME, LIST_OF_KEYWORDS)

- LIST_OF_KEYWORDS is a list of keywords associated with the constraint. Keywords may be linked to the meaning of the constraint, to a typical pattern where the constraint can be applied or to a specific problem where the constraint is useful. All keywords used in the catalogue are listed in alphabetic order in Section 3.7 on page 147. Each keyword has an entry explaining its meaning and providing the list of global constraints using that keyword.

14. ctr_eval(CONSTRAINT_NAME, LIST_OF_EVALUATORS)

- For many of the constraints of the catalogue one or several evaluators are provided. Each evaluator is explicitly described in LIST_OF_EVALUATORS by an element of the form method(predicate_name), where predicate_name is the name of the Prolog predicate to call in order to evaluate the constraint,¹ and method can be one of the following keywords:
  - builtin when the corresponding evaluator uses a SICStus built-in. This is for instance the case for the alldifferent constraint.
  - reformulation when the corresponding evaluator reformulates the constraints in terms of a conjunction of constraints of the catalogue and/or in term of a conjunction of reified constraints. This is for instance the case for the tree constraint.
  - automaton when the corresponding evaluator is based on an automaton that describes the set of solutions accepted by the constraint. The evaluator corresponds to the Prolog code that creates the signature constraints as well as the automata (usually one) associated with the constraint. A fact of the form automaton/9 lists the states and the transitions of the automata used for describing the set of solutions accepted by the constraint. It follows the description provided in Section 2.3.2 on page 82. The pattern constraint is an example of constraint for which an automaton is provided.

¹Note that this predicate name should be different from existing SICStus built-ins.
4.4.2 XML schema associated with a global constraint

In this section we describe an XML schema associated with the global constraint catalogue. We present the motivation for this schema, how it integrates with the description of the constraint in the catalogue, and how the schema information is updated when the catalogue is modified.

Related work

There have been a number of approaches to defining an exchange format for constraint models.

The seminal OPL language [392] provides a modelling language for constraint programs, which is linked to Ilog’s solver products. Its use an exchange format is limited by its proprietary background. MiniZinc [273] is a subset of the Zinc modelling language intended to be compiled to multiple solver implementations. First, a model in FlatZinc is generated from the MiniZinc model, removing all iteration (respectively recursion). The flat model can then be compiled into different solver implementations, currently Mercury, ECLiPSe and Gecode. The development of new back-ends is facilitated by the co-development of the Cadmium [375] term-rewriting system, which can parse and transform FlatZinc code.

The work most closely related to our format probably is the XML format used for the CSP solver competitions [389]. We reviewed an earlier draft version before generating our own schema for the catalogue, the 2007 version (for the 2008 competition) is described in [276]. It is intended as a solver independent format, which can be used by all participants of the competition. As a design choice, the authors decided not to fully structure the format, e.g. to use string values to hold structured information. In order to understand the actual meaning of the model, these strings need to be parsed and analysed as well. This may have size advantages for CSP data given in extensional form, but makes it more complex to check validity of a data file.

Key features

The following list summarizes the core features of our XML format and the associated schema:
4.4. THE ELECTRONIC VERSION OF THE CATALOGUE

language independent  The underlying description of the constraint in the catalogue is provided as Prolog facts. These may be difficult/tedious to read in other programming languages. The use of XML as an exchange format allows use with most programming languages via provided XML parsers.

machine readable, precise format  The format is precisely defined, using XML schema data types throughout, so that validity of a model can be checked with standard XML tools.

one-to-one match with the data format used for the catalog  The internal structure of the schema follows the data format for the constraints in the rest of the catalogue. This minimises the need for relearning, once the basic format of the catalogue description has been understood.

detailed description of the allowed format for arguments  For each global constraint, the allowed format of the argument is specified in great detail. As the complexity of global constraints increases, this becomes more and more important to simplify the generation of valid problem files.

automated generation of schema from the catalogue data files  The schema is automatically generated from the catalogue data files by the simple generator program. This keeps the schema up-to-date with changes of the catalogue, and reduces the task of schema maintenance.

generation of examples for each constraint  Example XML files based on the examples in the catalogue can be generated automatically, so that a link to these examples can be added to each catalogue entry.

generation of diagrams describing schema for each constraint  At the same time, graph structures of the schema for each constraint can be automatically generated using the graphviz [168] tool. This can help a human user to produce XML data for a particular constraint without reading the details of the schema.

Structure of schema

Model  The top-level element for the schema is model, which contains an optional variables element and a required constraints element.

Variables  The variables element consists of a non-empty sequence of variable elements, each describing a single variable which may occur in some of the constraints. Each variable has some attributes, an required id, an optional name and a required external. The id is an XML schema ID used to refer to the variable in the constraints of the model, the name is a string which describes the variable to the user, and external is a "yes"/"no" string which states if the variable is visible outside the model.

The domains of variables are not described as part of the variables section, special unary constraints (e.g., in_interval) are used in the constraint section instead.
**constraints** The *constraints* element consists of one or more elements representing constraints in the catalogue. The constraints can be stated in any order, with the understanding that the order may influence the sequence in which they are introduced to the solver.

For each constraint in the catalogue, a specific element with the same name is described in the schema. This imposes restrictions on the names of constraints in the catalogue, only alphanumerical names (with underscores) should be used.

Each constraint has attributes id (type ID), a name (type string) and an optional description (type string). The name and description can be used to include user-readable information about the constraint for example for debugging or explanations.

For each introduced element, a sequence of arguments is defined to define the arguments of the constraints in the same order as described in the catalogue. Each of the arguments has a specific type, which is defined in accordance with the catalogue definition. The argument names can be reused throughout the catalogue, as long as they are unique within each constraint. Arguments can have atomic values (i.e., consist of a single value), or they may be collection elements.

**collection** Roughly, collections correspond to lists in Prolog. Collections can be empty, or must contain entries of the same type. Collections can be nested as required.

**item** Items correspond to terms in Prolog. Items have named arguments, for which the same rules apply as for the arguments of constraints. The different arguments of an item can be of different type.

**Generating schema from the catalogue**

There are two programs which can be used to build the schema description from the data describing the catalogue. They should be run whenever a description of a constraint in the catalogue has been changed.

**schema.ecl** The *ECLiPSe* [9] program *schema.ecl* can be used to re-generate the schema when the catalogue description has been modified. The query *schema.* produces the schema from descriptions in the *src* directory, the query *top.* produces example files for each constraint in the *xml* directory.

The predicate *handle_table* defines which of the restrictions in the constraint description are included as part of the schema information. Many of the more complex rules cannot be easily checked by the schema, an entry in *handle_table* says to ignore the restriction for the moment.

**schema.dot.ecl** The program *schema.dot.ecl* can be used to generate *graphviz dot* files from the schema file *schema.xsd*. The generated files are placed in the *images* directory, and a *dot* command to produce *.png* and *.eps* output is run in the same directory. The pixel based png files are intended for use in web pages, the scalable eps files can be used in *LaTeX* files producing postscript or pdf documents.
There are some predicates in `schema_dot.ecl` which control the format of the generated graph. They are:

- The predicate `range_style` controls the display of range information, and optional/required choices for attributes.
- The predicate `type_shape` defines the shape and color of the different elements in the schema for a constraint.
- The predicate `match_builtin` provides an abbreviated element name for some of the predefined element types in the schema. This is required as the graphs should not become too big to fit onto a single A4 page in the output.

**Conclusion**

We have described the rationale and details for an XML schema attached to the global constraint catalogue. It allows to describe models using the constraints of the catalogue as flat XML files, which are a good exchange format for generating and/or parsing constraint data.
# Chapter 5

## Global Constraint Catalogue

<table>
<thead>
<tr>
<th>Section</th>
<th>Constraint</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>abs_value</td>
<td>420</td>
</tr>
<tr>
<td>5.2</td>
<td>all_differ_from_at_least_k_pos</td>
<td>422</td>
</tr>
<tr>
<td>5.3</td>
<td>all_equal</td>
<td>426</td>
</tr>
<tr>
<td>5.4</td>
<td>all_incomparable</td>
<td>428</td>
</tr>
<tr>
<td>5.5</td>
<td>all_min_dist</td>
<td>430</td>
</tr>
<tr>
<td>5.6</td>
<td>alldifferent</td>
<td>434</td>
</tr>
<tr>
<td>5.7</td>
<td>alldifferent_between_sets</td>
<td>442</td>
</tr>
<tr>
<td>5.8</td>
<td>alldifferent_consecutive_values</td>
<td>444</td>
</tr>
<tr>
<td>5.9</td>
<td>alldifferent_cst</td>
<td>446</td>
</tr>
<tr>
<td>5.10</td>
<td>alldifferent_except_0</td>
<td>450</td>
</tr>
<tr>
<td>5.11</td>
<td>alldifferent_interval</td>
<td>454</td>
</tr>
<tr>
<td>5.12</td>
<td>alldifferent_modulo</td>
<td>458</td>
</tr>
<tr>
<td>5.13</td>
<td>alldifferent_on_intersection</td>
<td>462</td>
</tr>
<tr>
<td>5.14</td>
<td>alldifferent_partition</td>
<td>466</td>
</tr>
<tr>
<td>5.15</td>
<td>alldifferent_same_value</td>
<td>470</td>
</tr>
<tr>
<td>5.16</td>
<td>allperm</td>
<td>474</td>
</tr>
<tr>
<td>5.17</td>
<td>among</td>
<td>478</td>
</tr>
<tr>
<td>5.18</td>
<td>among_diff_0</td>
<td>486</td>
</tr>
<tr>
<td>5.19</td>
<td>among_interval</td>
<td>490</td>
</tr>
<tr>
<td>5.20</td>
<td>among_low_up</td>
<td>494</td>
</tr>
<tr>
<td>5.21</td>
<td>among_modulo</td>
<td>498</td>
</tr>
<tr>
<td>5.22</td>
<td>among_seq</td>
<td>502</td>
</tr>
<tr>
<td>5.23</td>
<td>among_var</td>
<td>506</td>
</tr>
<tr>
<td>5.24</td>
<td>and</td>
<td>510</td>
</tr>
<tr>
<td>5.25</td>
<td>arith</td>
<td>514</td>
</tr>
<tr>
<td>5.26</td>
<td>arith_or</td>
<td>518</td>
</tr>
<tr>
<td>5.27</td>
<td>arith_sliding</td>
<td>522</td>
</tr>
<tr>
<td>Section</td>
<td>Constraint Name</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.28</td>
<td>assign_and_counts</td>
<td>526</td>
</tr>
<tr>
<td>5.29</td>
<td>assign_and_nvalues</td>
<td>530</td>
</tr>
<tr>
<td>5.30</td>
<td>atleast</td>
<td>534</td>
</tr>
<tr>
<td>5.31</td>
<td>atleast_nvalue</td>
<td>538</td>
</tr>
<tr>
<td>5.32</td>
<td>atleast_nvector</td>
<td>542</td>
</tr>
<tr>
<td>5.33</td>
<td>atmost</td>
<td>546</td>
</tr>
<tr>
<td>5.34</td>
<td>atmost1</td>
<td>550</td>
</tr>
<tr>
<td>5.35</td>
<td>atmost_nvalue</td>
<td>552</td>
</tr>
<tr>
<td>5.36</td>
<td>atmost_nvector</td>
<td>556</td>
</tr>
<tr>
<td>5.37</td>
<td>balance</td>
<td>560</td>
</tr>
<tr>
<td>5.38</td>
<td>balance_cycle</td>
<td>566</td>
</tr>
<tr>
<td>5.39</td>
<td>balance_interval</td>
<td>570</td>
</tr>
<tr>
<td>5.40</td>
<td>balance_modulo</td>
<td>574</td>
</tr>
<tr>
<td>5.41</td>
<td>balance_partition</td>
<td>578</td>
</tr>
<tr>
<td>5.42</td>
<td>balance_path</td>
<td>582</td>
</tr>
<tr>
<td>5.43</td>
<td>balance_tree</td>
<td>586</td>
</tr>
<tr>
<td>5.44</td>
<td>between_min_max</td>
<td>590</td>
</tr>
<tr>
<td>5.45</td>
<td>bin_packing</td>
<td>594</td>
</tr>
<tr>
<td>5.46</td>
<td>bin_packing_capa</td>
<td>600</td>
</tr>
<tr>
<td>5.47</td>
<td>binary_tree</td>
<td>602</td>
</tr>
<tr>
<td>5.48</td>
<td>bipartite</td>
<td>606</td>
</tr>
<tr>
<td>5.49</td>
<td>calendar</td>
<td>610</td>
</tr>
<tr>
<td>5.50</td>
<td>cardinality_atleast</td>
<td>620</td>
</tr>
<tr>
<td>5.51</td>
<td>cardinality_atmost</td>
<td>624</td>
</tr>
<tr>
<td>5.52</td>
<td>cardinality_atmost_partition</td>
<td>628</td>
</tr>
<tr>
<td>5.53</td>
<td>change</td>
<td>632</td>
</tr>
<tr>
<td>5.54</td>
<td>change_continuity</td>
<td>638</td>
</tr>
<tr>
<td>5.55</td>
<td>change_pair</td>
<td>650</td>
</tr>
<tr>
<td>5.56</td>
<td>change_partition</td>
<td>656</td>
</tr>
<tr>
<td>5.57</td>
<td>change_vectors</td>
<td>660</td>
</tr>
<tr>
<td>5.58</td>
<td>circuit</td>
<td>662</td>
</tr>
<tr>
<td>5.59</td>
<td>circuit_cluster</td>
<td>666</td>
</tr>
<tr>
<td>5.60</td>
<td>circular_change</td>
<td>672</td>
</tr>
<tr>
<td>5.61</td>
<td>clause_and</td>
<td>676</td>
</tr>
<tr>
<td>5.62</td>
<td>clause_or</td>
<td>680</td>
</tr>
<tr>
<td>5.63</td>
<td>clique</td>
<td>684</td>
</tr>
<tr>
<td>5.64</td>
<td>colored_matrix</td>
<td>688</td>
</tr>
<tr>
<td>5.65</td>
<td>coloured_cumulative</td>
<td>692</td>
</tr>
<tr>
<td>5.66</td>
<td>coloured_cumulatives</td>
<td>698</td>
</tr>
<tr>
<td>5.67</td>
<td>common</td>
<td>704</td>
</tr>
<tr>
<td>5.68</td>
<td>common_interval</td>
<td>708</td>
</tr>
<tr>
<td>5.69</td>
<td>common_modulo</td>
<td>712</td>
</tr>
<tr>
<td>5.70</td>
<td>common_partition</td>
<td>716</td>
</tr>
<tr>
<td>Section</td>
<td>Keyword</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.71</td>
<td>compare_and_count</td>
<td>720</td>
</tr>
<tr>
<td>5.72</td>
<td>cond_lex_cost</td>
<td>722</td>
</tr>
<tr>
<td>5.73</td>
<td>cond_lex_greater</td>
<td>726</td>
</tr>
<tr>
<td>5.74</td>
<td>cond_lex_greatereq</td>
<td>730</td>
</tr>
<tr>
<td>5.75</td>
<td>cond_lex_less</td>
<td>734</td>
</tr>
<tr>
<td>5.76</td>
<td>cond_lex_lessseq</td>
<td>738</td>
</tr>
<tr>
<td>5.77</td>
<td>connect_points</td>
<td>742</td>
</tr>
<tr>
<td>5.78</td>
<td>connected</td>
<td>746</td>
</tr>
<tr>
<td>5.79</td>
<td>consecutive_groups_of_ones</td>
<td>748</td>
</tr>
<tr>
<td>5.80</td>
<td>consecutive_values</td>
<td>752</td>
</tr>
<tr>
<td>5.81</td>
<td>contains_sboxes</td>
<td>754</td>
</tr>
<tr>
<td>5.82</td>
<td>correspondence</td>
<td>758</td>
</tr>
<tr>
<td>5.83</td>
<td>count</td>
<td>762</td>
</tr>
<tr>
<td>5.84</td>
<td>counts</td>
<td>766</td>
</tr>
<tr>
<td>5.85</td>
<td>coveredby_sboxes</td>
<td>770</td>
</tr>
<tr>
<td>5.86</td>
<td>covers_sboxes</td>
<td>776</td>
</tr>
<tr>
<td>5.87</td>
<td>crossing</td>
<td>782</td>
</tr>
<tr>
<td>5.88</td>
<td>cumulative</td>
<td>786</td>
</tr>
<tr>
<td>5.89</td>
<td>cumulative_convex</td>
<td>794</td>
</tr>
<tr>
<td>5.90</td>
<td>cumulative_product</td>
<td>802</td>
</tr>
<tr>
<td>5.91</td>
<td>cumulative_two_d</td>
<td>808</td>
</tr>
<tr>
<td>5.92</td>
<td>cumulative_with_level_of_priority</td>
<td>812</td>
</tr>
<tr>
<td>5.93</td>
<td>cumulatives</td>
<td>818</td>
</tr>
<tr>
<td>5.94</td>
<td>cutset</td>
<td>824</td>
</tr>
<tr>
<td>5.95</td>
<td>cycle</td>
<td>828</td>
</tr>
<tr>
<td>5.96</td>
<td>cycle_card_on_path</td>
<td>834</td>
</tr>
<tr>
<td>5.97</td>
<td>cycle_or_accessibility</td>
<td>838</td>
</tr>
<tr>
<td>5.98</td>
<td>cycle_resource</td>
<td>842</td>
</tr>
<tr>
<td>5.99</td>
<td>cyclic_change</td>
<td>848</td>
</tr>
<tr>
<td>5.100</td>
<td>cyclic_change_joker</td>
<td>852</td>
</tr>
<tr>
<td>5.101</td>
<td>dag</td>
<td>856</td>
</tr>
<tr>
<td>5.102</td>
<td>decreasing</td>
<td>858</td>
</tr>
<tr>
<td>5.103</td>
<td>deepest_valley</td>
<td>862</td>
</tr>
<tr>
<td>5.104</td>
<td>derangement</td>
<td>866</td>
</tr>
<tr>
<td>5.105</td>
<td>differ_from_at_least_k_pos</td>
<td>868</td>
</tr>
<tr>
<td>5.106</td>
<td>diffn</td>
<td>872</td>
</tr>
<tr>
<td>5.107</td>
<td>diffn_column</td>
<td>882</td>
</tr>
<tr>
<td>5.108</td>
<td>diffn_include</td>
<td>886</td>
</tr>
<tr>
<td>5.109</td>
<td>discrepancy</td>
<td>890</td>
</tr>
<tr>
<td>5.110</td>
<td>disj</td>
<td>894</td>
</tr>
<tr>
<td>5.111</td>
<td>disjoint</td>
<td>898</td>
</tr>
<tr>
<td>5.112</td>
<td>disjoint_sboxes</td>
<td>902</td>
</tr>
<tr>
<td>5.113</td>
<td>disjoint_tasks</td>
<td>908</td>
</tr>
</tbody>
</table>
5.114 disjunctive..........................912
5.115 disjunctive_or_same_end.........916
5.116 disjunctive_or_same_start......918
5.117 distance..............................920
5.118 distance_between................922
5.119 distance_change...................926
5.120 divisible............................930
5.121 divisible_or.........................932
5.122 dom_reachability...................934
5.123 domain...............................938
5.124 domain_constraint................940
5.125 elem................................946
5.126 elem_from_to.........................954
5.127 element...............................958
5.128 element_greatereq................962
5.129 element_lesseq......................966
5.130 element_matrix......................970
5.131 element_product....................974
5.132 element_sparse......................978
5.133 elementsn...........................982
5.134 elements.............................986
5.135 elements_alldifferent...............990
5.136 elements_sparse.....................996
5.137 eq................................1000
5.138 eq_cst................................1002
5.139 eq_set................................1004
5.140 equal_sboxes.........................1006
5.141 equivalent...........................1010
5.142 exactly...............................1012
5.143 gcd................................1016
5.144 geost................................1018
5.145 geost_time...........................1024
5.146 geq................................1030
5.147 geq_cst................................1032
5.148 global_cardinality..................1034
5.149 global_cardinality_low_up........1040
5.150 global_cardinality_low_up_no_loop1044
5.151 global_cardinality_no_loop........1048
5.152 global_cardinality_with_costs.....1052
5.153 global_contiguity...................1058
5.154 golomb................................1062
5.155 graph_crossing.....................1066
5.156 graph_isomorphism..................1072
5.200 leq .................................................. 1262
5.201 leq_cst ........................................... 1264
5.202 lex2 ................................................. 1266
5.203 lex_alldifferent .................................. 1268
5.204 lex_between ....................................... 1272
5.205 lex_chain_less .................................. 1276
5.206 lex_chain_lesseq ................................. 1280
5.207 lex_different ..................................... 1284
5.208 lex_equal .......................................... 1288
5.209 lex_greater ........................................ 1292
5.210 lex_greatereq .................................... 1298
5.211 lex_less ............................................ 1304
5.212 lex_lesseq ........................................ 1310
5.213 lex_lesseq_allperm .............................. 1316
5.214 link_set_to_booleans ............................. 1318
5.215 longest_change ................................... 1322
5.216 lt ..................................................... 1326
5.217 map .................................................. 1328
5.218 max_index ......................................... 1332
5.219 max_n ............................................... 1334
5.220 max_nvalue ....................................... 1338
5.221 max_size_set_of_consecutive_var ............... 1344
5.222 maximum .......................................... 1348
5.223 maximum_modulo .................................. 1352
5.224 meet_sboxes ...................................... 1354
5.225 min_index ......................................... 1360
5.226 min_n ............................................... 1364
5.227 min_nvalue ....................................... 1368
5.228 min_size_set_of_consecutive_var ............... 1374
5.229 minimum .......................................... 1378
5.230 minimum_except_0 ................................ 1382
5.231 minimum_greater_than ......................... 1386
5.232 minimum_modulo .................................. 1392
5.233 minimum_weight_alldifferent .................. 1394
5.234 multi_global_contiguity ....................... 1398
5.235 multi_inter_distance ............................. 1400
5.236 nand ............................................... 1402
5.237 nclass ............................................. 1406
5.238 neq ................................................ 1410
5.239 neq_cst ........................................... 1412
5.240 nequivalence .................................... 1414
5.241 next_element .................................... 1418
5.242 next_greater_element ........................... 1424
5.243 ninterval ................................................. 1428
5.244 no_peak ............................................... 1432
5.245 no_valley ............................................... 1436
5.246 non_overlap_sboxes ................................. 1440
5.247 nor ....................................................... 1446
5.248 not_all_equal ........................................... 1450
5.249 not_in .................................................... 1454
5.250 npair ..................................................... 1458
5.251 nset_of_consecutive_values ...................... 1462
5.252 nvalue ................................................... 1466
5.253 nvalue_on_intersection ............................ 1472
5.254 nvalues .................................................. 1476
5.255 nvalues_except_0 .................................... 1480
5.256 nvector .................................................. 1484
5.257 nvectors ................................................ 1490
5.258 nvisible_from_end ................................... 1494
5.259 nvisible_from_start ................................ 1496
5.260 open_alldifferent ................................... 1498
5.261 open_among ............................................ 1502
5.262 open_atleast ......................................... 1506
5.263 open_atmost ......................................... 1508
5.264 open_global_cardinality .......................... 1510
5.265 open_global_cardinality_low_up ............... 1514
5.266 open_maximum ....................................... 1518
5.267 open_minimum ....................................... 1520
5.268 opposite_sign ....................................... 1522
5.269 or ......................................................... 1524
5.270 orchard ............................................... 1528
5.271 ordered_atleast_nvector .......................... 1532
5.272 ordered_atmost_nvector ............................ 1536
5.273 ordered_global_cardinality ..................... 1540
5.274 ordered_nvector .................................... 1544
5.275 orth_link_ori_siz_end ............................. 1548
5.276 orth_on_the_ground ................................ 1552
5.277 orth_on_top_of_orth ............................... 1554
5.278 orths_are_connected ................................ 1558
5.279 overlap_sboxes ...................................... 1562
5.280 path ....................................................... 1566
5.281 path_from_to ......................................... 1570
5.282 pattern ................................................ 1574
5.283 peak ...................................................... 1578
5.284 period .................................................. 1582
5.285 period_except_0 ..................................... 1584
<table>
<thead>
<tr>
<th>Section</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.286</td>
<td>period_vectors</td>
</tr>
<tr>
<td>5.287</td>
<td>permutation</td>
</tr>
<tr>
<td>5.288</td>
<td>place_in_pyramid</td>
</tr>
<tr>
<td>5.289</td>
<td>polyomino</td>
</tr>
<tr>
<td>5.290</td>
<td>power</td>
</tr>
<tr>
<td>5.291</td>
<td>precedence</td>
</tr>
<tr>
<td>5.292</td>
<td>product_ctr</td>
</tr>
<tr>
<td>5.293</td>
<td>proper_forest</td>
</tr>
<tr>
<td>5.294</td>
<td>range_ctr</td>
</tr>
<tr>
<td>5.295</td>
<td>relaxed_sliding_sum</td>
</tr>
<tr>
<td>5.296</td>
<td>remainder</td>
</tr>
<tr>
<td>5.297</td>
<td>roots</td>
</tr>
<tr>
<td>5.298</td>
<td>same</td>
</tr>
<tr>
<td>5.299</td>
<td>same_and_global_cardinality</td>
</tr>
<tr>
<td>5.300</td>
<td>same_and_global_cardinality_low_up</td>
</tr>
<tr>
<td>5.301</td>
<td>same_intersection</td>
</tr>
<tr>
<td>5.302</td>
<td>same_interval</td>
</tr>
<tr>
<td>5.303</td>
<td>same_modulo</td>
</tr>
<tr>
<td>5.304</td>
<td>same_partition</td>
</tr>
<tr>
<td>5.305</td>
<td>same_sign</td>
</tr>
<tr>
<td>5.306</td>
<td>scalar_product</td>
</tr>
<tr>
<td>5.307</td>
<td>sequence_folding</td>
</tr>
<tr>
<td>5.308</td>
<td>set_value_precede</td>
</tr>
<tr>
<td>5.309</td>
<td>shift</td>
</tr>
<tr>
<td>5.310</td>
<td>sign_of</td>
</tr>
<tr>
<td>5.311</td>
<td>size_max_seq_alldifferent</td>
</tr>
<tr>
<td>5.312</td>
<td>size_max_starting_seq_alldifferent</td>
</tr>
<tr>
<td>5.313</td>
<td>sliding_card_skip0</td>
</tr>
<tr>
<td>5.314</td>
<td>sliding_distribution</td>
</tr>
<tr>
<td>5.315</td>
<td>sliding_sum</td>
</tr>
<tr>
<td>5.316</td>
<td>sliding_time_window</td>
</tr>
<tr>
<td>5.317</td>
<td>sliding_time_window_from_start</td>
</tr>
<tr>
<td>5.318</td>
<td>sliding_time_window_sum</td>
</tr>
<tr>
<td>5.319</td>
<td>smooth</td>
</tr>
<tr>
<td>5.320</td>
<td>soft_all_equal_max_var</td>
</tr>
<tr>
<td>5.321</td>
<td>soft_all_equal_min_ctr</td>
</tr>
<tr>
<td>5.322</td>
<td>soft_all_equal_min_var</td>
</tr>
<tr>
<td>5.323</td>
<td>soft_alldifferent_ctr</td>
</tr>
<tr>
<td>5.324</td>
<td>soft_alldifferent_var</td>
</tr>
<tr>
<td>5.325</td>
<td>soft_cumulative</td>
</tr>
<tr>
<td>5.326</td>
<td>soft_same_interval_var</td>
</tr>
<tr>
<td>5.327</td>
<td>soft_same_modulo_var</td>
</tr>
<tr>
<td>5.328</td>
<td>soft_same_partition_var</td>
</tr>
<tr>
<td>Constraint</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.372 used_by</td>
<td>1918</td>
</tr>
<tr>
<td>5.373 used_by_interval</td>
<td>1924</td>
</tr>
<tr>
<td>5.374 used_by_modulo</td>
<td>1928</td>
</tr>
<tr>
<td>5.375 used_by_partition</td>
<td>1932</td>
</tr>
<tr>
<td>5.376 uses</td>
<td>1936</td>
</tr>
<tr>
<td>5.377 valley</td>
<td>1940</td>
</tr>
<tr>
<td>5.378 vec_eq_tuple</td>
<td>1944</td>
</tr>
<tr>
<td>5.379 visible</td>
<td>1946</td>
</tr>
<tr>
<td>5.380 weighted_partial_alldiff</td>
<td>1958</td>
</tr>
<tr>
<td>5.381 xor</td>
<td>1962</td>
</tr>
</tbody>
</table>
5.1 abs_value

**DESCRIPTION**

**Origin**
Arithmetic.

**Constraint**
abs_value(Y, X)

**Usual name**
abs

**Synonym**
absolute.value.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>dvar</td>
</tr>
<tr>
<td>X</td>
<td>dvar</td>
</tr>
</tbody>
</table>

**Restriction**

Y \geq 0

**Purpose**
Enforce the fact that the first variable is equal to the absolute value of the second variable.

**Example**

\[ (8, -8) \]

The abs_value constraint holds since 8 is equal to \(| -8 |\).

**Arg. properties**

Functional dependency: Y determined by X.

**Systems**

abs in Choco, abs in Gecode.

**See also**

implied by: eq.
implies: geq.

**Keywords**

constraint arguments: binary constraint, pure functional dependency.

constraint type: predefined constraint, arithmetic constraint.

filtering: arc-consistency.

modelling: functional dependency.
5.2 all_differ_from_at_least_k_pos

**DESCRIPTION**

Inspired by [164].

**LINKS**

all_differ_from_at_least_k_pos(K, VECTORS)

**GRAPH**

VECTOR : collection(var-dvar)

K : int
VECTORS : collection(vec - VECTOR)

**Restrictions**

required(VECTOR, var)
|VECTOR| ≥ 1
|VECTOR| ≥ K
K ≥ 0
required(VECTORS, vec)
same_size(VECTORS, vec)

**Purpose**

Enforce all pairs of distinct vectors of the VECTORS collection to differ from at least K positions.

**Example**

\[
\begin{pmatrix}
2, \langle vec - (2, 5, 2, 0) \rangle, \\
2, \langle vec - (3, 6, 2, 1) \rangle, \\
2, \langle vec - (3, 6, 1, 0) \rangle
\end{pmatrix}
\]

The all_differ_from_at_least_k_pos constraint holds since:

- The first and second vectors differ from 3 positions, which is greater than or equal to K = 2.
- The first and third vectors differ from 3 positions, which is greater than or equal to K = 2.
- The second and third vectors differ from 2 positions, which is greater than or equal to K = 2.

**Typical**

K > 0
|VECTOR| < |VECTORS|
|VECTORS| > 1

**Symmetries**

- Items of VECTORS are permutable.
- Items of VECTORS.vec are permutable (same permutation used).

**Arg. properties**

- Contractible wrt. VECTORS.
- Extensible wrt. VECTORS.vec (add items at same position).
See also

part of system of constraints: differ_from_at_least_k_pos.
used in graph description: differ_from_at_least_k_pos.

Keywords

application area: bioinformatics.
characteristic of a constraint: disequality, vector.
constraint type: system of constraints, decomposition.
final graph structure: no loop, symmetric.
### Arc input(s)
VECTORS

### Arc generator
\[ \text{CLIQUE}(\neq) \rightarrow \text{collection}(\text{vectors1}, \text{vectors2}) \]

### Arc arity
2

### Arc constraint(s)
differ from at least \( k \) pos \((K, \text{vectors1.vec}, \text{vectors2.vec})\)

### Graph property(ies)
\[ \text{NARC} = |\text{VECTORS}| \cdot |\text{VECTORS}| - |\text{VECTORS}| \]

### Graph class
- NO LOOP
- SYMMETRIC

#### Graph model
The Arc constraint(s) slot uses the \text{differ from at least \( k \) pos} constraint defined in this catalogue.

Parts (A) and (B) of Figure 5.1 respectively show the initial and final graph associated with the Example slot. Since we use the \text{NARC} graph property, the arcs of the final graph are stressed in bold. The previous constraint holds since exactly \( 3 \cdot (3 - 1) = 6 \) arc constraints hold.

![Figure 5.1: Initial and final graph of the all_differ_from_at_least_k_pos constraint](image)

**Signature**
Since we use the \text{CLIQUE}(\neq) arc generator on the items of the VECTORS collection, the expression \(|\text{VECTORS}| \cdot |\text{VECTORS}| - |\text{VECTORS}|\) corresponds to the maximum number of arcs of the final graph. Therefore we can rewrite the graph property \text{NARC} = |\text{VECTORS}| \cdot |\text{VECTORS}| - |\text{VECTORS}| to \text{NARC} \geq |\text{VECTORS}| \cdot |\text{VECTORS}| - |\text{VECTORS}|. This leads to simplify \text{NARC} to \text{NARC}. 
5.3 all_equal

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from soft_all_equal_min_ctr</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>all_equal(VARIABLES)</td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td>rel.</td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>VARIABLES : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(VARIABLES, var)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[</td>
</tr>
<tr>
<td>Purpose</td>
<td>Enforce all variables of the collection VARIABLES to take the same value.</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>[(5, 5, 5, 5)]</td>
<td></td>
</tr>
</tbody>
</table>

The all_equal constraint holds since all its variables are fixed to value 5.

| Typical | \[|VARIABLES| > 2\] |

| Symmetries | • Items of VARIABLES are permutable. |
|            | • All occurrences of a value of VARIABLES.var can be renamed to any unused value. |

| Arg. properties | Contractible wrt. VARIABLES. |

| Systems | atMostNValue in Choco, rel in Gecode, all_equal in MiniZinc. |

| See also | generalisation: nvalue (a variable counting the number of distinct values is introduced). |
|          | implies: consecutive_values, decreasing, increasing. |
|          | negation: not_all_equal. |

| soft variant: soft_all_equal_max.var, |
| soft_all_equal_min_ctr (decomposition-based violation measure), |
| soft_all_equal_min_var (variable-based violation measure). |

| specialisation: eq (equality between just two variables). |

| Keywords | constraint type: value constraint. |
Arc input(s)            VARIABLES
Arc generator          \textit{PATH} \mapsto \textit{collection}(\text{variables1}, \text{variables2})
Arc arity              2
Arc constraint(s)      \text{variables1}.\text{var} = \text{variables2}.\text{var}
Graph property(ies)    \text{NARC} = |\text{VARIABLES}| - 1

Graph model

We use the arc generator \textit{PATH} in order to link consecutive variables of the collection \text{VARIABLES} by a binary equality constraint.

Parts (A) and (B) of Figure 5.2 respectively show the initial and final graph of the \textbf{Example} slot. Since we use the \textbf{NARC} graph property, the arcs of the final graph are stressed in bold.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig5.2.png}
\caption{Initial and final graph of the all\_equal constraint}
\end{figure}
### 5.4 all_incomparable

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
</table>

- **Origin**: Inspired by incomparable rectangles.
- **Constraint**: `all_incomparable(VECTORS)`
- **Synonym**: `all_incomparables`
- **Type**: `VECTOR : collection(var−dvar)`
- **Argument**: `VECTORS : collection(vec − VECTOR)`
- **Restrictions**:
  - `required(VECTOR, var)
  - \(|\text{VECTOR}| \geq 1\)
  - `required(VECTORS, vec)
  - \(|\text{VECTORS}| \geq 1\)
  - `same_size(VECTORS, vec)`
- **Purpose**: Enforce for each pair of distinct vectors of the `VECTORS` collection the fact that they are incomparable. Two vectors `VECTOR1` and `VECTOR2` are incomparable if and only, when the components of both vectors are ordered, and respectively denoted by `SVECTOR1` and `SVECTOR2`, we neither have `SVECTOR1[i].var \leq SVECTOR2[i].var` (for all `i ∈ [1, |SVECTOR1|]`) nor have `SVECTOR2[i].var \leq SVECTOR1[i].var` (for all `i ∈ [1, |SVECTOR1|]`).
- **Example**:

  $\left( \left\{ \text{vec} − (16, 2), \text{vec} − (4, 11), \text{vec} − (5, 10) \right\} \right)$

  The `all_incomparable` constraint holds since all distinct pairs of vectors are incomparable.
- **Typical**:
  - `|\text{VECTOR}| > 1`
  - `|\text{VECTORS}| > 1`
  - `|\text{VECTORS}| > |\text{VECTOR}|`
- **Symmetry**: Items of `VECTORS` are permutable.
- **Arg. properties**: Contractible wrt. `VECTORS`.
- **See also**: part of system of constraints: incomparable.
  used in graph description: incomparable.
- **Keywords**: characteristic of a constraint: vector.
  constraint type: system of constraints, decomposition.
  final graph structure: no loop, symmetric.
Arc input(s): VECTORS
Arc generator: $\text{CLIQUE}(\neq) \mapsto \text{collection}(\text{vectors}_1, \text{vectors}_2)$
Arc arity: 2
Arc constraint(s): $\text{incomparable}($vectors$_1.\text{vec}, \text{vectors}_2.\text{vec})$
Graph property(ies): $\text{NARC} = |\text{VECTORS}| + |\text{VECTORS}| - |\text{VECTORS}|$
Graph class:
- NO_LOOP
- SYMMETRIC

Graph model: The Arc constraint(s) slot uses the $\text{incomparable}$ constraint defined in this catalogue.
Parts (A) and (B) of Figure 5.3 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold. The previous constraint holds since exactly $3 \cdot (3 - 1) = 6$ arc constraints hold.

![Graph Diagram](image)

Figure 5.3: Initial and final graph of the all_incomparable constraint
5.5 all_min_dist

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td></td>
<td>[323]</td>
</tr>
<tr>
<td>Constraint</td>
<td>all_min_dist(MINDIST, VARIABLES)</td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>minimum_distance, inter_distance</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>MINDIST : int</td>
<td>VARIABLES : collection(var−dvar)</td>
</tr>
<tr>
<td>Restrictions</td>
<td>MINDIST &gt; 0</td>
<td>VARIABLES</td>
</tr>
<tr>
<td>Purpose</td>
<td>Enforce for each pair (var_i, var_j) of distinct variables of the collection VARIABLES that</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[var_i − var_j] ≥ MINDIST.</td>
</tr>
<tr>
<td>Example</td>
<td>(2, ⟨5, 1, 9, 3⟩)</td>
<td></td>
</tr>
<tr>
<td>Typical</td>
<td>MINDIST &gt; 1</td>
<td>VARIABLES</td>
</tr>
<tr>
<td>Symmetries</td>
<td>• MINDIST can be decreased to any value ≥ 1.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Items of VARIABLES are permutable.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Two distinct values of VARIABLES.var can be swapped.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• One and the same constant can be added to the var attribute of all items of VARIABLES.</td>
<td></td>
</tr>
<tr>
<td>Arg. properties</td>
<td>Contractible wrt. VARIABLES.</td>
<td></td>
</tr>
<tr>
<td>Usage</td>
<td>The all_min_dist constraint was initially created for handling frequency allocation problems. In [10] it is used for scheduling tasks that all have the same fixed duration in the context of air traffic management in the terminal radar control area of airports.</td>
<td></td>
</tr>
<tr>
<td>Remark</td>
<td>The all_min_dist constraint can be modelled as a set of tasks that should not overlap. For each variable var of the VARIABLES collection we create a task t where var and MINDIST respectively correspond to the origin and the duration of t. Some solvers use in a pre-processing phase, while stating constraints of the form</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[X_i − X_j] ≥ D_{ij} (where X_i and X_j are domain variables and D_{ij} is a constant), an algorithm for automatically extracting large cliques [83] from such inequalities in order to state all_min_dist constraints.</td>
<td></td>
</tr>
</tbody>
</table>
Algorithm

K. Artiouchine and P. Baptiste came up with a cubic time complexity algorithm achieving bound-consistency in [10, 11] based on the adaptation of a feasibility test algorithm from M.R. Garey et al. [171]. Later on, C.-G. Quimper et al., proposed a quadratic algorithm achieving the same level of consistency in [312].

See also

generalisation: diffn(line segment, of same length, replaced by orthotope), disjunctive(line segment, of same length, replaced by line segment), multi_inter_distance(LIMIT parameter introduced to specify capacity ≥1).
implies: alldifferent_interval.
related: distance.
specialisation: alldifferent(line segment, of same length, replaced by variable).

Keywords

application area: frequency allocation problem, air traffic management.
characteristic of a constraint: sort based reformulation.
constraint type: value constraint, decomposition, scheduling constraint.
filtering: bound-consistency.
final graph structure: acyclic.
problems: maximum clique.
Arc input(s) | VARIABLES
---|---
Arc generator | $CLIQUE(<) \rightarrow \text{collection}(\text{variables}_1, \text{variables}_2)$
Arc arity | 2
Arc constraint(s) | $\text{abs}(\text{variables}_1.\text{var} - \text{variables}_2.\text{var}) \geq \text{MINDIST}$
Graph property(ies) | $\text{NARC} = |\text{VARIABLES}| \times (|\text{VARIABLES}| - 1)/2$
Graph class | • ACYCLIC • NO LOOP

**Graph model**

We generate a *clique* with a minimum distance constraint between each pair of distinct vertices and state that the number of arcs of the final graph should be equal to the number of arcs of the initial graph.

Parts (A) and (B) of Figure 5.4 respectively show the initial and final graph associated with the **Example** slot. The **all_min_dist** constraint holds since all the arcs of the initial graph belong to the final graph: all the minimum distance constraints are satisfied.

![Figure 5.4: Initial and final graph of the all_min_dist constraint](image-url)
## 5.6 alldifferent

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>[238]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>alldifferent(VARIABLES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Synonyms</strong></td>
<td>alldiff, alldistinct, distinct, bound_alldifferent, bound_alldiff, bound_distinct, rel.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Argument</strong></td>
<td>VARIABLES : collection(var−dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Restriction</strong></td>
<td>required(VARIABLES, var)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>Enforce all variables of the collection VARIABLES to take distinct values.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Example

\[ (5, 1, 9, 3) \]

The alldifferent constraint holds since all the values 5, 1, 9 and 3 are distinct.

### Typical

\[ |\text{VARIABLES}| > 1 \]

### Symmetries

- Items of VARIABLES are permutable.
- Two distinct values of VARIABLES.var can be swapped; a value of VARIABLES.var can be renamed to any unused value.

### Arg. properties

Contractible wrt. VARIABLES.

### Usage

The alldifferent constraint occurs in most practical problems directly or indirectly. A classical example is the n-queen chess puzzle problem: Place \( n \) queens on a \( n \) by \( n \) chessboard in such a way that no queen attacks another. Two queens attack each other if they are located on the same column, on the same row or on the same diagonal. This can be modelled as the conjunction of three alldifferent constraints. We associate to column \( i \) of the chessboard a domain variable \( X_i \) that gives the row number where the corresponding queen is located. The three alldifferent constraints are:

- \( \text{alldifferent}(X_1, X_2 + 1, \ldots, X_n + n - 1) \) for the upper-left to lower-right diagonals,
- \( \text{alldifferent}(X_1, X_2, \ldots, X_n) \) for the rows,
- \( \text{alldifferent}(X_1 + n - 1, X_2 + n - 2, \ldots, X_n) \) for the lower right to upper-left diagonals.
They are respectively depicted by parts (A), (C) and (D) of Figure 5.5.

A second example taken from [13] when the bipartite graph associated with the alldifferent constraint is convex is a ski assignment problem: “a set of skiers have each specified the smallest and largest skis they will accept from a given set of skis”. The task is to find a ski for each skier.

Examples such as Costas arrays or Golomb rulers involve one or several alldifferent constraints on differences of variables.

Quite often, the alldifferent constraint is also used in conjunction with several element constraints, specially in the context of assignment problems [pages 372–374][198], or with several precedence constraints, specially in the context of symmetry breaking or scheduling problems [71].

Other examples involving several alldifferent constraints sharing some variables can be found in the Usage slot of the k_alldifferent constraint.

**Remark**

Even if the alldifferent constraint had not this form, it was specified in ALICE [237, 238] by asking for an injective correspondence between variables and values: \( x \neq y \Rightarrow f(x) \neq f(y) \). From an algorithmic point of view, the algorithm for computing the cardinality of the maximum matching of a bipartite graph was not used for checking the feasibility of the alldifferent constraint, even if the algorithm was already known in 1976. This is because the goal of ALICE was to show that a general system could be as efficient as dedicated algorithms. For this reason the concluding part of [237] explicitly mentions that specialized algorithms should be discarded. On the one hand, many people, specially from the OR community, have complained about such radical statement [343, page 28]. On the other hand, the motivation of such statement stands from the fact that a truly intelligent system should not rely on black box algorithms, but should rather be able to reconstruct them from some kind of first principle. How to achieve this is still an open question.

Some solvers use in a pre-processing phase before stating all constraints, an algorithm for automatically extracting large cliques [83, 140] from a set of binary disequalities in order to replace them by alldifferent constraints.

W.-J. van Hoeve provides a survey about the alldifferent constraint in [396].

For possible relaxation of the alldifferent constraints see the alldifferent_except_0, the k_alldifferent (i.e., some different), the soft_alldifferent_ctr, the soft_alldifferent_var and the weighted_partial_alldiff constraints.

Within the context of linear programming, relaxations of the alldifferent constraint are described in [415] and in [198, pages 362–367].

Within the context of constraint-centered search heuristics, G. Pesant and A. Zararini [421] have proposed several estimators for evaluating the number of solutions of an alldifferent constraint (since counting the total number of maximum matchings of the corresponding variable-value graph is \#P-complete [388]). Faster, but less accurate estimators, based on upper bounds of the number of solutions were proposed three years later by the same authors [422].

Given \( n \) variables taking their values within the interval \([1, n]\), the total number of solutions of the corresponding alldifferent constraint corresponds to the sequence A000142 of the On-Line Encyclopedia of Integer Sequences [370].
Algorithm

The first complete filtering algorithm was independently found by M.-C. Costa [116] and J.-C. Régin [320]. This algorithm is based on a corollary of C. Berge that characterises the edges of a graph that belong to a maximum matching but not to all [53, page 120]. A short time after, assuming that all variables have no holes in their domain, M. Leconte came up with a filtering algorithm [241] based on edge finding. A first bound-consistency algorithm was proposed by Bleuzen-Guernalec et al. [74]. Later on, two different approaches were used to design bound-consistency algorithms. Both approaches model the constraint as a bipartite graph. The first identifies Hall intervals in this graph [304, 247] and the second applies the same algorithm that is used to compute arc-consistency, but achieves a speedup by exploiting the simpler structure [179] of the graph [262]. Ian P. Gent et al. discuss in [174] implementations issues behind the complete filtering algorithm and in particular the computation of the strongly connected components of the residual graph (i.e., a graph constructed from a maximum variable-value matching and from the possible values of the variables of the allDifferent constraint), which appears to be the main bottleneck in practice.

From a worst case complexity point of view, assuming that \( n \) is the number of variables and \( m \) the sum of the domains sizes, we have the following complexity results:

- Complete filtering is achieved in \( O(m\sqrt{n}) \) by Régis’s algorithm [320].
- Range consistency is done in \( O(n^2) \) by Leconte’s algorithm [241].
- Bound-consistency is performed in \( O(n \log n) \) in [304, 262, 247]. If sort can be achieved in linear time, typically when the allDifferent constraint encodes a permutation, the worst case complexity of the algorithms described in [262, 247] goes down to \( O(n) \).

Within the context of explanations [210], the explanation of the filtering algorithm that achieves arc-consistency for the allDifferent constraint is described in [339, pages 60–61]. Given the residual graph (i.e., a graph constructed from a maximum variable-value matching and from the possible values of the variables of the allDifferent constraint), the removal of an arc starting from a vertex belonging to a strongly connected component \( C_1 \) to a distinct strongly connected component \( C_2 \) is explained by all missing arcs starting from a descendant component of \( C_2 \) and ending in an ancestor component of \( C_1 \) (i.e., since the addition of any of these missing arcs would merge the strongly connected components \( C_1 \) and \( C_2 \)). Let us illustrate this on a concrete example. For this purpose assume we have the following variables and the values that can potentially be assigned to each of them, \( A \in \{1, 2\} \), \( B \in \{1, 2\} \), \( C \in \{2, 3, 4, 6\} \), \( D \in \{3, 4\} \), \( E \in \{5, 6\} \), \( F \in \{5, 6\} \), \( G \in \{6, 7, 8\} \), \( H \in \{6, 7, 8\} \). Figure 5.6 represents the residual graph associated with the maximum matching corresponding to the assignment \( A = 1 \), \( B = 2 \), \( C = 3 \), \( D = 4 \), \( E = 5 \), \( F = 6 \), \( G = 7 \), \( H = 8 \). It has four strongly connected components containing respectively vertices \( \{A, B, 1, 2\} \), \( \{C, D, 3, 4\} \), \( \{E, F, 5, 6\} \) and \( \{G, H, 7, 8\} \). Arcs that are between strongly connected components correspond to values that can be removed:

- The removal of value 2 from variable \( C \) is explained by the absence of the arcs corresponding to the assignments \( A = 3 \), \( A = 4 \), \( B = 3 \) and \( B = 4 \) (since adding any of these missing arcs would merge the blue and the pink strongly connected components containing the vertices corresponding to value 2 and variable \( C \)).

1A similar result is in fact given in [289].

2In this context the total number of values that can be assigned to the variables of the allDifferent constraint is equal to the number of variables. Under this assumption sorting the variables on their minimum or maximum values can be achieved in linear time.
• The removal of value 6 from variable C is explained by the absence of the arcs corresponding to the assignments \( E = 3 \), \( E = 4 \), \( F = 3 \) and \( F = 4 \). Again adding the corresponding arcs would merge the two strongly connected components containing the vertices corresponding to value 6 and variable C.

• The removal of value 6 from variable G is explained by the absence of the arcs corresponding to the assignments \( E = 7 \), \( E = 8 \), \( F = 7 \) and \( F = 8 \).

• The removal of value 6 from variable H is explained by the absence of the arcs corresponding to the assignments \( E = 7 \), \( E = 8 \), \( F = 7 \) and \( F = 8 \).

After applying bound-consistency the following property holds for all variables of an alldifferent constraint. Given a Hall interval \([l, u]\), any variable \( V \) whose range \([V, \overline{V}]\) intersects \([l, u]\) without being included in \([l, u]\) has its minimum value \( V \) (respectively maximum value \( \overline{V} \)) that is located before (respectively after) the Hall interval (i.e., \( V < l \leq u < \overline{V} \)).

The alldifferent constraint is entailed if and only if there is no value \( v \) that can be assigned two distinct variables of the VARIABLES collection (i.e., the intersection of the two sets of potential values of any pair of variables is empty).

Reformulation

The alldifferent constraint can be reformulated into a set of disequalities constraints. This model neither preserves bound-consistency nor arc-consistency:

• On the one hand a model, involving linear constraints, preserving bound-consistency was introduced in [67]. For each potential interval \([l, u]\) of consecutive values this model uses \( |\text{VARIABLES}| \) 0-1 variables \( B_{1,l,u}, B_{2,l,u}, \ldots, B_{|\text{VARIABLES}|,l,u} \) for modelling that each variable of the collection VARIABLES is assigned a value within interval \([l, u]\) (i.e., \( \forall i \in [1,|\text{VARIABLES}|] : B_{i,l,u} \Leftrightarrow \text{VARIABLES}[i].\text{var} \in [l, u] \)).

• On the other hand, it was shown in [70] that there is no polynomial sized decomposition that preserves arc-consistency.

Finally the alldifferent(VARIABLES) constraint can also be reformulated as the conjunction sort(VARIABLES, SORTED VARIABLES) \( \wedge \) strictly_increasing(SORTED VARIABLES). Unlike the naive reformulation, i.e., a disequality constraint between each pair of variables, the sort-based reformulation is linear in space.

Systems

allDifferent in Choco, linear in Gecode, alldifferent in JaCoP, alldiff in JaCoP, alldistinct in JaCoP, all_different in MiniZinc, all_different in SICStus, all_distinct in SICStus.

Used in

allDifferent, consecutive_values, circuit_cluster, correspondence, cumulative_convex, size_max_seq_alldifferent, size_max_starting_seq_alldifferent, sort_permutation.

---

3How to encode the reified constraint \( B_{i,l,u} \Leftrightarrow \text{VARIABLES}[i].\text{var} \in [l, u] \) with linear constraints is described in the Reformulation slot of the in_interval_reified constraint.
Figure 5.5: Upper-left to lower-right diagonals (A-B), rows (C) and lower-right to upper-left diagonals (D-E)

Figure 5.6: Strongly connected components of the residual graph illustrating the explanation of the removal of a value for the constraint \texttt{alldifferent}(⟨A, B, C, D, E, F, G, H⟩), \(A \in \{1, 2\}, \ B \in \{1, 2\}, \ C \in \{2, 3, 4, 6\}, \ D \in \{3, 4\}, \ E \in \{5, 6\}, \ F \in \{5, 6\}, \ G \in \{6, 7, 8\}, \ H \in \{6, 7, 8\}\): the explanation why value 2 is removed from variable C corresponds to all missing arcs whose addition would merge the blue and the pink strongly connected components (i.e., the missing arcs corresponding to the assignments \(A = 3, \ A = 4, \ B = 3\) and \(B = 4\) that are depicted by thick pink lines)
See also
circuit, circuit_cluster, cycle, derangement (permutation), golomb (all different), size_max_seq_alldifferent, size_max_starting_seq_alldifferent (all different, disequality), symmetric_alldifferent (permutation).
cost variant: minimum_weight_alldifferent, weighted_partial_alldiff.
generalisation: all_min_dist (variable replaced by line segment, all of the same size), alldifferent_between_sets (variable replaced by set variable), alldifferent_cst (variable replaced by variable + constant), alldifferent_interval (variable replaced by variable/constant), alldifferent_modulo (variable replaced by variable mod constant), alldifferent_partition (variable replaced by variable ∈ partition), diffn (variable replaced by orthotope), disjunctive (variable replaced by task), global_cardinality (control the number of occurrence of each value with a counter variable), global_cardinality_low_up (control the number of occurrence of each value with an interval), lex_alldifferent (variable replaced by vector), nvalue (count number of distinct values).
implied by: alldifferent_consecutive_values, circuit, cycle, strictly_decreasing, strictly_increasing.
implies: alldifferent_except_0, not_all_equal.
negation: some_equal.
part of system of constraints: neq.
shift of concept: alldifferent_on_intersection, alldifferent_same_value.
soft variant: alldifferent_except_0 (value 0 can be used several times), open_alldifferent (open constraint), soft_alldifferent_cst (decomposition-based violation measure), soft_alldifferent_var (variable-based violation measure).

system of constraints: k_alldifferent.
used in reformulation: in_interval_reified (bound-consistency preserving reformulation), sort, strictly_increasing.
uses in its reformulation: cycle, elements_alldifferent, sort_permutation.

Keywords
characteristic of a constraint: core, all different, disequality, sort based reformulation, automaton, automaton with array of counters.
combinatorial object: permutation.
constraint type: system of constraints, value constraint.
filtering: bipartite matching, bipartite matching in convex bipartite graphs, convex bipartite graph, flow, Hall interval, arc-consistency, bound-consistency, SAT, DFS-bottleneck, entailment.
final graph structure: one_succ.
modelling exercises: n-Amazon, zebra puzzle.
problems: maximum clique, graph colouring.
puzzles: n-Amazon, n-queen, Costas arrays, Euler knight, Golomb ruler, magic hexagon, magic square, zebra puzzle, Sudoku.
**Graph model**

We generate a *clique* with an *equality* constraint between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed one.

Parts (A) and (B) of Figure 5.7 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_NSCC** graph property we show one of the largest strongly connected component of the final graph. The **alldifferent** holds since all the strongly connected components have at most one vertex: a value is used at most once.

![Graph](image)

**Figure 5.7:** Initial and final graph of the **alldifferent** constraint
Automaton

Figure 5.8 depicts the automaton associated with the alldifferent constraint. To each item of the collection VARIABLES corresponds a signature variable $S_i$, that is equal to 1. The automaton counts the number of occurrences of each value and finally imposes that each value is taken at most one time.

\begin{align*}
\{C[\_]=0\} & \\
S_i & \\
\text{arith}(C,\!<\!,2) & \\
\{C[\_]=C[\_]+1\} & \\
\{C[\text{VAR}_i]=C[\text{VAR}_i]+1\} &
\end{align*}

Figure 5.8: Automaton of the alldifferent constraint
### 5.7 alldifferent_between_sets

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>ILOG</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>alldifferent_between_sets(VARIABLES)</td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>all_null_intersect, alldiff_between_sets, alldistinct_between_sets, alldiff_on_sets, alldistinct_on_sets, alldifferent_on_sets.</td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>VARIABLES : collection(var−svar)</td>
<td></td>
</tr>
<tr>
<td>Restriction</td>
<td>required(VARIABLES, var)</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>Enforce all sets of the collection VARIABLES to be distinct.</td>
<td></td>
</tr>
</tbody>
</table>
| Example     | \[
\begin{align*}
\text{var} & = \{3, 5\}, \\
\text{var} & = \emptyset, \\
\text{var} & = \{3\}, \\
\text{var} & = \{3, 5, 7\}
\end{align*}
\] |       |

The alldifferent_between_sets constraint holds since all the sets \{3, 5\}, \emptyset, \{3\} and \{3, 5, 7\} are distinct.

| Typical     | \(|\text{VARIABLES}| > 2\) |       |
| Symmetry    | Items of VARIABLES are permutable. |       |
| Arg. properties | Contractible wrt. VARIABLES. |       |
| Usage       | This constraint was available in some configuration library offered by Ilog. |       |
| Algorithm   | A filtering algorithm for the alldifferent_between_sets is proposed by C.-G. Quimper and T. Walsh in [315] and a longer version is available in [316] and in [317]. |       |
| See also    | common keyword: link_set_to_booleans (constraint involving set variables). specialisation: alldifferent(set variable replaced by variable). used in graph description: eq_set. |       |
| Keywords    | characteristic of a constraint: all different, disequality. constraint arguments: constraint involving set variables. filtering: bipartite matching. final graph structure: one_succ. |       |
We generate a *clique* with binary *set equalities* constraints between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed 1.

Parts (A) and (B) of Figure 5.9 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_NSNC** graph property we show one of the largest strongly connected component of the final graph. The **alldifferent_between_sets** holds since all the strongly connected components have at most one vertex.

(B)  
**Figure 5.9**: Initial and final graph of the alldifferent_between_sets constraint
## 5.8 alldifferent_consecutive_values

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from alldifferent.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>alldifferent_consecutive_values(VARIABLES)</td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>VARIABLES : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(VARIABLES.var)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>alldifferent(VARIABLES)</td>
<td></td>
</tr>
</tbody>
</table>

### Purpose

Enforce (1) all variables of the collection VARIABLES to take distinct values and (2) constraint the difference between the largest and the smallest values of the VARIABLES collection to be equal to the number of variables minus one (i.e., there is no holes at all within the used values).

### Example

\[\langle 5, 4, 3, 6 \rangle\]

The alldifferent_consecutive_values constraint holds since (1) all the values 5, 4, 3 and 6 are distinct and since (2) all values between value 3 and value 6 are effectively used.

### Typical

\[|\text{VARIABLES}| > 2\]

### Symmetries

- Items of VARIABLES are permutable.
- Two distinct values of VARIABLES.var can be swapped.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

### See also

- implied by: permutation.
- implies: alldifferent_consecutive_values.

### Keywords

- characteristic of a constraint: all different, disequality, sort based reformulation.
- combinatorial object: permutation.
- constraint type: value constraint.
<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>SELF→collection(variables)</td>
</tr>
<tr>
<td>Arc arity</td>
<td>1</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>TRUE</td>
</tr>
<tr>
<td>Graph property(ies)</td>
<td>[\textit{RANGE}(\text{VARIABLES}, \text{var}) =</td>
</tr>
</tbody>
</table>
5.9 \textbf{alldifferent} \textbf{cst}

**DESCRIPTION**

CHIP

**Constraint**

\texttt{alldifferent\_cst(VARIABLES)}

**Synonyms**

\texttt{alldiff\_cst, all\_distinct\_cst}.

**Argument**

\texttt{VARIABLES : collection(var\,-\,dvar, cst\,-\,int)}

**Restriction**

\texttt{required(VARIABLES, [var, cst])}

**Purpose**

For all pairs of items \((\text{VARIABLES}[i], \text{VARIABLES}[j]) \text{ (} i \neq j \text{)}\) of the collection \text{VARIABLES} enforce \text{VARIABLES}[i].\text{var} + \text{VARIABLES}[i].\text{cst} \neq \text{VARIABLES}[j].\text{var} + \text{VARIABLES}[j].\text{cst}.

**Example**

\[
\begin{pmatrix}
\text{var} - 5 & \text{cst} - 0,
\text{var} - 1 & \text{cst} - 1,
\text{var} - 9 & \text{cst} - 0,
\text{var} - 3 & \text{cst} - 4
\end{pmatrix}
\]

The \texttt{alldifferent\_cst} constraint holds since all the expressions \(5 + 0 = 5\), \(1 + 1 = 2\), \(9 + 0 = 9\) and \(3 + 4 = 7\) correspond to distinct values.

**Typical**

\begin{align*}
|\text{VARIABLES}| & > 2 \\
\text{range(\text{VARIABLES}\.var)} & > 1 \\
\text{range(\text{VARIABLES}\.cst)} & > 1
\end{align*}

**Symmetries**

- Items of \text{VARIABLES} are \texttt{permutable}.
- Attributes of \text{VARIABLES} are \texttt{permutable} \texttt{w.r.t.} permutation \((\text{var, cst}) \text{ (permutable not necessarily applied to all items)}\).
- One and the same constant can be \texttt{added} to the \text{var} attribute of all items of \text{VARIABLES}.
- One and the same constant can be \texttt{added} to the \text{cst} attribute of all items of \text{VARIABLES}.

**Arg. properties**

\texttt{Contractible} \texttt{w.r.t.} \text{VARIABLES}.

**Usage**

The \texttt{alldifferent\_cst} constraint was originally introduced in \texttt{CHIP} in order to express the \(n\)-queen problem with 3 global constraints (see the \texttt{Usage} slot of the \texttt{alldifferent} constraint).

**Algorithm**

See the filtering algorithms of the \texttt{alldifferent} constraint.
Systems

linear in Gecode.

See also

implies (items to collection): lex_alldifferent.
specialisation: alldifferent (variable + constant replaced by variable).

Keywords

characteristic of a constraint: all different, disequality, sort based reformulation.
constraint type: value constraint.
filtering: bipartite matching, bipartite matching in convex bipartite graphs, convex bipartite graph, arc-consistency.
final graph structure: one_suc.
modelling exercises: n-Amazon.
puzzles: n-Amazon, n-queen.
Arc input(s) | VARIABLES
---|---
Arc generator | `CLIQUE`→collection(variables1,variables2)
Arc arity | 2
Arc constraint(s) | variables1.var + variables1.cst = variables2.var + variables2.cst
Graph property(ies) | `MAX_NSCC` ≤ 1
Graph class | `ONE_SUCC`

**Graph model**

We generate a *clique* with an *equality* constraint between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed one.

Parts (A) and (B) of Figure 5.10 respectively show the initial and final graph associated with the *Example* slot. Since we use the `MAX_NSCC` graph property we show one of the largest strongly connected component of the final graph. The *alldifferent_cst* holds since all the strongly connected components have at most one vertex: a value is used at most once.

![Figure 5.10: Initial and final graph of the alldifferent_cst constraint](image)
5.10 alldifferent_except_0

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from alldifferent.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>alldifferent_except_0(VARIABLES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>alldiff_except_0, alldistinct_except_0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>VARIABLES : collection(var−dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restriction</td>
<td>required(VARIABLES, var)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>Enforce all variables of the collection VARIABLES to take distinct values, except those variables that are assigned value 0.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example

\[
\langle \text{var}−5, \\
\text{var}−0, \\
\langle \text{var}−1, \\
\text{var}−9, \\
\text{var}−0, \\
\text{var}−3 \rangle 
\]

The alldifferent_except_0 constraint holds since all the values (that are different from 0) 5, 1, 9 and 3 are distinct.

Typical

\[
|\text{VARIABLES}| > 2 \\
\text{atleast}(2, \text{VARIABLES}, 0) \\
\text{range}(\text{VARIABLES}.\text{var}) > 1
\]

Symmetries

- Items of VARIABLES are permutable.
- Two distinct values of VARIABLES.var that are both different from 0 can be swapped; a value of VARIABLES.var that is different from 0 can be renamed to any unused value that is also different from 0.

Arg. properties

Contractible wrt. VARIABLES.

Usage

Quite often it appears that, for some modelling reason, you create a joker value. You do not want that normal constraints hold for variables that take this joker value. For this purpose we modify the binary arc constraint in order to discard the vertices for which the corresponding variables are assigned value 0. This will be effectively the case since all the corresponding arcs constraints will not hold.

See also

cost variant: weighted_partial_alldiff.
hard version: alldifferent.
implied by: alldifferent.
Keywords

- **characteristic of a constraint:** joker value, all different, sort based reformulation, automaton, automaton with array of counters.
- **constraint type:** value constraint, relaxation.
- **filtering:** arc-consistency.
- **final graph structure:** one_suc.
Arc input(s) Variables
Arc generator $\text{CLIQUE} \rightarrow \text{collection}(\text{variables}_1, \text{variables}_2)$
Arc arity 2
Arc constraint(s)
• $\text{variables}_1$.var $\neq 0$
• $\text{variables}_1$.var = $\text{variables}_2$.var
Graph property(ies) $\text{MAX}_{\text{NSCC}} \leq 1$

Graph model
The graph model is the same as the one used for the $\text{alldifferent}$ constraint, except that we discard all variables that are assigned value 0.

Parts (A) and (B) of Figure 5.11 respectively show the initial and final graph associated with the Example slot. Since we use the $\text{MAX}_{\text{NSCC}}$ graph property we show one of the largest strongly connected component of the final graph. The $\text{alldifferent}$ except 0 holds since all the strongly connected components have at most one vertex: a value different from 0 is used at most once.

![Graph Model](image)

Figure 5.11: Initial and final graph of the $\text{alldifferent}$ except 0 constraint
Figure 5.12 depicts the automaton associated with the \texttt{alldifferent\_except\_0} constraint. To each variable \(\text{VAR}_i\) of the collection \texttt{VARIABLES} corresponds a 0-1 signature variable \(S_i\). The following signature constraint links \(\text{VAR}_i\) and \(S_i\): \(\text{VAR}_i \neq 0 \iff S_i\). The automaton counts the number of occurrences of each value different from 0 and finally imposes that each non-zero value is taken at most one time.

\[
\begin{align*}
\text{VAR}_1 &= 0 \\
S_1 : & \text{ arith}(C, <, 2) \quad \text{VAR}_1 \neq 0, \\
& C[\text{VAR}_1] = C[\text{VAR}_1] + 1 \\
\{C[\_]=0\} & \text{ (initial state)}
\end{align*}
\]

Figure 5.12: Automaton of the \texttt{alldifferent\_except\_0} constraint
5.11 alldifferent_interval

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from alldifferent.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>alldifferent_interval(VARIABLES,SIZE_INTERVAL)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>alldiff_interval, alldistinct_interval.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VARIABLES : collection(var−dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SIZE_INTERVAL : int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(VARIABLES,var)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SIZE_INTERVAL &gt; 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Purpose

Enforce all variables of the collection VARIABLES to belong to distinct intervals. The intervals are defined by $[SIZE_INTERVAL \cdot k, SIZE_INTERVAL \cdot k + SIZE_INTERVAL - 1]$ where $k$ is an integer.

Example

$((2, 4, 10), 3)$

In the example, the second argument $SIZE_INTERVAL = 3$ defines the following family of intervals $[3 \cdot k, 3 \cdot k + 2]$, where $k$ is an integer. Since the three variables of the collection VARIABLES take values that are respectively located within the three following distinct intervals $[0, 2]$, $[3, 5]$ and $[9, 11]$, the alldifferent_interval constraint holds.

Typical

$|VARIABLES| > 2$
$SIZE_INTERVAL > 1$
$SIZE_INTERVAL < range(VARIABLES.var)$

Symmetries

- Items of VARIABLES are permutable.
- A value of VARIABLES.var that belongs to the $k$-th interval, of size $SIZE_INTERVAL$, can be renamed to any unused value of the same interval.
- Two distinct values of VARIABLES.var that belong to two distinct intervals, of size $SIZE_INTERVAL$, can be swapped.

Arg. properties

Contractible wrt. VARIABLES.

See also

implied by: all_min_dist.

specialisation: alldifferent (variable/constant replaced by variable).

Keywords

characteristic of a constraint: all different, sort based reformulation, automaton, automaton with array of counters.

constraint type: value constraint.
filtering: arc-consistency.
final graph structure: one.suce.
modelling: interval.
Graph model

Similar to the alldifferent constraint, but we replace the binary equality constraint of the alldifferent constraint by the fact that two variables are respectively assigned to two values that belong to the same interval. We generate a clique with a belong to the same interval constraint between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed 1.

Parts (A) and (B) of Figure 5.13 respectively show the initial and final graph associated with the Example slot. Since we use the MAX_NSCC graph property we show one of the largest strongly connected component of the final graph.

Figure 5.13: Initial and final graph of the alldifferent interval constraint
Automaton

Figure 5.14 depicts the automaton associated with the `alldifferent_interval` constraint. To each item of the collection `VARIABLES` corresponds a signature variable `S_i` that is equal to 1. For each interval `[SIZE_INTERVAL \cdot k, SIZE_INTERVAL \cdot k + SIZE_INTERVAL - 1]` of values the automaton counts the number of occurrences of its values and finally imposes that the values of an interval are taken at most once.

![Automaton Diagram]

Figure 5.14: Automaton of the `alldifferent_interval` constraint
### 5.12 alldifferent_modulo

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from <code>alldifferent</code>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td><code>alldifferent_modulo(VARIABLES, M)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td><code>alldiff_modulo, alldistinct_modulo</code>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td><code>VARIABLES : collection(var−dvar)</code>&lt;br&gt;<code>M : int</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td><code>required(VARIABLES, var)</code>&lt;br&gt;<code>M &gt; 0</code>&lt;br&gt;`M ≥</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>Enforce all variables of the collection <code>VARIABLES</code> to have a distinct rest when divided by <code>M</code>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td><code>((25, 1, 14, 3), 5)</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The equivalence classes associated with values 25, 1, 14 and 3 are respectively equal to 25 mod 5 = 0, 1 mod 5 = 1, 14 mod 5 = 4 and 3 mod 5 = 3. Since they are distinct the `alldifferent_modulo` constraint holds.

| Typical     | `|VARIABLES| > 2`<br>`M > 1` |               |           |
|-------------|---------------------|----------------|-----------|
| Symmetries  | `• Items of VARIABLES are permutable.`<br>`• A value `u` of `VARIABLES.var` can be renamed to any value `v` such that `v` is congruent to `u` modulo `M`.`<br>`• Two distinct values `u` and `v` of `VARIABLES.var` such that `u mod M ≠ v mod M` can be swapped.` |               |           |
| Arg. properties | `Contractible wrt. VARIABLES.` | | |
| See also    | `specialisation: alldifferent(variable mod constant replaced by variable).` | | |
| Keywords    | `characteristic of a constraint: modulo, all different, sort based reformulation, automaton, automaton with array of counters.`<br>`constraint type: value constraint.`<br>`filtering: arc-consistency.`<br>`final graph structure: one_suc.` | | |
Graph model

Exploit the same model used for the `alldifferent` constraint. We replace the binary equality constraint by another equivalence relation depicted by the arc constraint. We generate a `clique` with a binary `equality modulo` $M$ constraint between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed 1.

Parts (A) and (B) of Figure 5.15 respectively show the initial and final graph associated with the `Example` slot. Since we use the `MAX_NSCC` graph property we show one of the largest strongly connected component of the final graph.

![Figure 5.15: Initial and final graph of the alldifferent modulo constraint](image-url)
Automaton

Figure 5.16 depicts the automaton associated with the \texttt{alldifferent_modulo} constraint. To each item of the collection \texttt{VARIABLES} corresponds a signature variable $S_i$ that is equal to 1. The automaton counts for each equivalence class the number of used values and finally imposes that each equivalence class is used at most one time.

$$\begin{align*}
S_i: & \quad \text{arith}(C, <, 2) \\
& \quad \left\{ \begin{array}{l}
C[\_]=0 \\
C[\text{VAR}_i \mod M]=C[\text{VAR}_i \mod M]+1
\end{array} \right.
\end{align*}$$

Figure 5.16: Automaton of the \texttt{alldifferent_modulo} constraint
5.13  alldifferent_on_intersection

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from common and alldifferent.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>alldifferent_on_intersection(VARIABLES1, VARIABLES2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>alldiff_on_intersection, alldistinct_on_intersection.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VARIABLES1 : collection(var–dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES2 : collection(var–dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(VARIABLES1,var)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES2,var)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>The values that both occur in the VARIABLES1 and VARIABLES2 collections have only one occurrence.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example

\[
\begin{pmatrix}
5, 9, 1, 5, \\
\text{var }-2, \\
\text{var }-1, \\
\text{var }-6, \\
\text{var }-9, \\
\text{var }-6, \\
\text{var }-2
\end{pmatrix}
\]

The alldifferent_on_intersection constraint holds since the values 9 and 1 that both occur in \(\langle 5, 9, 1, 5 \rangle\) as well as in \(\langle 2, 1, 6, 9, 6, 2 \rangle\) have exactly one occurrence in each collection.

Typical

\[|\text{VARIABLES1}| > 1\]
\[|\text{VARIABLES2}| > 1\]

Symmetries

- Arguments are permutable w.r.t. permutation (VARIABLES1, VARIABLES2).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- All occurrences of two distinct values in VARIABLES1.var or VARIABLES2.var can be swapped; all occurrences of a value in VARIABLES1.var or VARIABLES2.var can be renamed to any unused value.

Arg. properties

- Contractible wrt. VARIABLES1.
- Contractible wrt. VARIABLES2.
See also  
common keyword: common, nvalue_on_intersection (constraint on the intersection).
implied by: disjoint.
implies: same_intersection.
root concept: alldifferent.

Keywords  
characteristic of a constraint: all different, automaton, automaton with array of counters.
constraint arguments: constraint between two collections of variables.
constraint type: constraint on the intersection, value constraint.
final graph structure: connected component, acyclic, bipartite, no loop.
Arc input(s) | VARIABLES1 VARIABLES2
---|---
Arc generator | \( \text{PRODUCT} \rightarrow \text{collection} (\text{variables1}, \text{variables2}) \)
Arc arity | 2
Arc constraint(s) | variables1.var = variables2.var
Graph property(ies) | \( \text{MAX\_NCC} \leq 2 \)
Graph class | • ACYCLIC
• BIPARTITE
• NO_LOOP

Graph model

Parts (A) and (B) of Figure 5.17 respectively show the initial and final graph associated with the Example slot. Since we use the \( \text{MAX\_NCC} \) graph property we show one of the largest connected component of the final graph. The alldifferent_on_intersection constraint holds since each connected component has at most two vertices. Note that all the vertices corresponding to the variables that take values 5, 2 or 6 were removed from the final graph since there is no arc for which the associated equality constraint holds.

![Figure 5.17: Initial and final graph of the alldifferent_on_intersection constraint](image-url)
Automaton

Figure 5.18 depicts the automaton associated with the alldifferent on intersection constraint. To each variable \( \text{VAR}_1 \) of the collection \( \text{VARIABLES}_1 \) corresponds a signature variable \( S_i \) that is equal to 0. To each variable \( \text{VAR}_2 \) of the collection \( \text{VARIABLES}_2 \) corresponds a signature variable \( S_{i+|\text{VARIABLES}_1|} \) that is equal to 1. The automaton first counts the number of occurrences of each value assigned to the variables of the \( \text{VARIABLES}_1 \) collection. It then counts the number of occurrences of each value assigned to the variables of the \( \text{VARIABLES}_2 \) collection. Finally, the automaton imposes that each value is not taken by two variables of both collections.

\[
\{C[\_] = 0, D[\_] = 0\}
\]

\[
\rightarrow
\]

\[
S
\]

\[
0,\quad \{C[\text{VAR}_1] = C[\text{VAR}_1] + 1\}
\]

\[
\rightarrow
\]

\[
1,\quad \{D[\text{VAR}_1] = D[\text{VAR}_1] + 1\}
\]

\[
\text{t:}\quad \text{arith_or}(C,D,<,2)
\]

\[
\rightarrow
\]

\[
1,\quad \{D[\text{VAR}_1] = D[\text{VAR}_1] + 1\}
\]

Figure 5.18: Automaton of the alldifferent on intersection constraint
### 5.14 alldifferent_partition

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from alldifferent.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>alldifferent_partition(VARIABLES, PARTITIONS)</td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>alldiff_partition, alldistinct_partition.</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>(\text{VALUES} : \text{collection(val-int)})</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VARIABLES : (\text{collection(var-dvar)})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PARTITIONS : (\text{collection(p-VALUES)})</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>(</td>
<td>\text{VALUES}</td>
</tr>
<tr>
<td></td>
<td>(\text{required(VALUES, val)})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{distinct(VALUES, val)})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(</td>
<td>\text{VARIABLES}</td>
</tr>
<tr>
<td></td>
<td>(\text{required(VARIABLES, var)})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(</td>
<td>\text{PARTITIONS}</td>
</tr>
<tr>
<td></td>
<td>(\text{required(PARTITIONS, p)})</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>Enforce all variables of the collection VARIABLES to take values that belong to distinct partitions.</td>
<td></td>
</tr>
</tbody>
</table>
| Example | \[
\begin{pmatrix}
(6, 3, 4), \\
p = (1, 3), \\
p = (4), \\
p = (2, 6)
\end{pmatrix}
\] | |
| | Since all variables take values that are located within distinct partitions the alldifferent_partition constraint holds. | |
| Typical | \(|\text{VARIABLES}| > 2\) | |
| Symmetries | \bullet \text{Items of VARIABLES are permutable.} | |
| | \bullet \text{Items of PARTITIONS are permutable.} | |
| | \bullet \text{Items of PARTITIONS.p are permutable.} | |
| | \bullet A value of VARIABLES.var can be renamed to any value that belongs to the same partition of PARTITIONS. | |
| | \bullet Two distinct values of VARIABLES.var that do not belong to the same partition of PARTITIONS can be swapped. | |
| Arg. properties | Contractible wrt. VARIABLES. | |
See also

- common keyword: \texttt{in\_same\_partition}(\textit{partition}).
- specialisation: \texttt{alldifferent} (\texttt{variable} \in \textit{partition} \texttt{replaced by variable}).
- used in graph description: \texttt{in\_same\_partition}.

Keywords

- characteristic of a constraint: partition, all different, sort based reformulation.
- constraint type: value constraint.
- filtering: arc-consistency.
- final graph structure: one\_succe.
- modelling: incompatible pairs of values.
Graph model

Similar to the alldifferent constraint, but we replace the binary equality constraint of the alldifferent constraint by the fact that two variables are respectively assigned to two values that belong to the same partition. We generate a clique with an \(\text{in\_same\_partition}\) constraint between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed 1.

Parts (A) and (B) of Figure 5.19 respectively show the initial and final graph associated with the Example slot. Since we use the MAX_NSCC graph property we show one of the largest strongly connected component of the final graph.

![Initial and final graph of the alldifferent_partition constraint](image)
5.15 alldifferent_same_value

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from alldifferent.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>alldifferent_same_value[NSAME, VARIABLES1, VARIABLES2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>alldiff.same.value, alldistinct.same.value.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>NSAME : dvar</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES1 : collection(var−dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES2 : collection(var−dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>NSAME ≥ 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NSAME ≤</td>
<td>VARIABLES1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>VARIABLES1</td>
<td>=</td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES1, var)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES2, var)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>All the values assigned to the variables of the collection VARIABLES1 are pairwise distinct. NSAME is equal to number of constraints of the form VARIABLES1[i].var = VARIABLES2[i].var (1 ≤ i ≤</td>
<td>VARIABLES1</td>
<td>) that hold.</td>
</tr>
<tr>
<td>Example</td>
<td>[ \left( \begin{array}{c} 2, \langle 7,3,1,5 \rangle, \ \langle 1,3,1,7 \rangle \end{array} \right) ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The alldifferent_same_value constraint holds since:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• All the values 7, 3, 1 and 5 are distinct,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Among the four expressions 7 = 1, 3 = 3, 1 = 1 and 5 = 7 exactly 2 conditions hold.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Typical</td>
<td>NSAME &lt;</td>
<td>VARIABLES1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>VARIABLES1</td>
<td>&gt; 2</td>
</tr>
<tr>
<td>Symmetries</td>
<td>• Items of VARIABLES1 and VARIABLES2 are permutable (same permutation used).</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• All occurrences of two distinct values in VARIABLES1.var or VARIABLES2.var can be swapped; all occurrences of a value in VARIABLES1.var or VARIABLES2.var can be renamed to any unused value.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arg. properties</td>
<td>Functional dependency: NSAME determined by VARIABLES1 and VARIABLES2.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Usage       | When all variables of the second collection are initially bound to distinct values the alldifferent_same_value constraint can be explained in the following way:
• We interpret the variables of the second collection as the previous solution of a problem where all variables have to be distinct.

• We interpret the variables of the first collection as the current solution to find, where all variables should again be pairwise distinct.

The variable NSAME measures the distance of the current solution from the previous solution. This corresponds to the number of variables of VARIABLES2 that are assigned to the same previous value.

See also root concept: alldifferent.

Keywords characteristic of a constraint: sort based reformulation, automaton, automaton with array of counters.
constraint type: proximity constraint.
modelling: functional dependency.
## Graph model

The arc generator $PRODUCT(CLIQUE, LOOP, =)$ is used in order to generate all the arcs of the initial graph:

- The arc generator $CLIQUE$ creates all links between the items of the first collection $VARIABLES1$.
- The arc generator $LOOP$ creates a loop for each item of the second collection $VARIABLES2$.
- Finally the arc generator $PRODUCT(=)$ creates an arc between items located at the same position in the collections $VARIABLES1$ and $VARIABLES2$.

Part (A) of Figure 5.20 gives the initial graph associated with the Example slot. Variables of collection $VARIABLES1$ are coloured, while variables of collection $VARIABLES2$ are kept in white. Part (B) represents the final graph associated with the Example slot. In this graph each vertex constitutes a strongly connected component and the number of arcs that do not correspond to a loop is equal to 2 (i.e., NSAME).

![Figure 5.20: Initial and final graph of the alldifferent_same_value constraint](image-url)
Figure 5.21 depicts the automaton associated with the `alldifferent_same_value` constraint. Let $VAR_1_i$ and $VAR_2_i$ respectively denote the $i^{th}$ variables of the `VARIABLES1` and `VARIABLES2` collections. To each pair of variables $(VAR_1_i, VAR_2_i)$ corresponds a signature variable $S_i$. The following signature constraint links $VAR_1_i$, $VAR_2_i$ and $S_i$: $VAR_1_i = VAR_2_i \Leftrightarrow S_i$.

$$\{C[\_]=0, D=0\}$$

$$VAR_1_i \neq VAR_2_i,$$

$$\{C[VAR_1_i]=C[VAR_1_i]+1\}$$

$$\text{arithmetic} \{C, <, 2\}$$

$$S:$$

$$\{C[VAR_1_i]=C[VAR_1_i]+1, D=D+1\}$$

Figure 5.21: Automaton of the `alldifferent_same_value` constraint
5.16 allperm

### Description

<table>
<thead>
<tr>
<th>Origin</th>
<th>[155]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint</td>
<td>allperm(MATRIX)</td>
</tr>
<tr>
<td>Synonyms</td>
<td>all_perm, all_permutations.</td>
</tr>
<tr>
<td>Type</td>
<td>VECTOR : \text{collection}(\text{var}−\text{dvar})</td>
</tr>
<tr>
<td>Argument</td>
<td>MATRIX : \text{collection}(\text{vec}−\text{VECTOR})</td>
</tr>
<tr>
<td>Restrictions</td>
<td>\text{</td>
</tr>
<tr>
<td>Purpose</td>
<td>Given a matrix (\mathcal{M}) of domain variables, enforces that the first row is lexicographically less than or equal to all permutations of all other rows. Note that the components of a given vector of the matrix (\mathcal{M}) may be equal.</td>
</tr>
</tbody>
</table>

### Example

\[
\left( \begin{array}{c}
\text{vec} - \langle 1, 2, 3 \rangle , \\
\text{vec} - \langle 3, 1, 2 \rangle \\
\end{array} \right)
\]

The allperm constraint holds since vector \(\langle 1, 2, 3 \rangle\) is lexicographically less than or equal to all the permutations of vector \(\langle 3, 1, 2 \rangle\) (i.e., \(\langle 1, 2, 3 \rangle\), \(\langle 1, 3, 2 \rangle\), \(\langle 2, 1, 3 \rangle\), \(\langle 2, 3, 1 \rangle\), \(\langle 3, 1, 2 \rangle\), \(\langle 3, 2, 1 \rangle\)).

### Typical

\[
\begin{array}{c}
\text{|VECTOR|} > 1 \\
\text{|MATRIX|} > 1 \\
\end{array}
\]

### Symmetry

One and the same constant can be added to the \text{var} attribute of all items of \text{MATRIX vec}.

### Arg. properties

Suffix-contractible wrt. MATRIX.vec (remove items from same position).

### Usage

A symmetry-breaking constraint.

### See also

- \text{common keyword: lex2, lex\_chain\_lesseq} (matrix symmetry, lexicographic order), \text{lex\_lesseq} (lexicographic order), \text{lex\_lesseq\_allperm} (matrix symmetry, lexicographic order), \text{strict\_lex2} (lexicographic order).
- \text{part of system of constraints: lex\_lesseq\_allperm}.
- \text{used in graph description: lex\_lesseq\_allperm}.
Keywords

characteristic of a constraint: sort based reformulation, vector.
constraint type: order constraint, system of constraints.
final graph structure: acyclic, bipartite.
modelling: matrix, matrix model.
symmetry: matrix symmetry, symmetry, lexicographic order.
Arc input(s) | MATRIX  
---|---
Arc generator | $\text{CLIQUE}(\langle \rangle) \mapsto \text{collection}(\text{matrix1}, \text{matrix2})$
Arc arity | 2
Arc constraint(s) |  
- $\text{matrix1.key} = 1$
- $\text{matrix2.key} > 1$
- $\text{lex_lesseq_allperm} (\text{matrix1.vec}, \text{matrix2.vec})$
Graph property(ies) | NARC = $|\text{MATRIX}| - 1$
Graph class |  
- ACYCLIC
- BIPARTITE
- NO_LOOP

Graph model
We generate a graph with an arc constraint $\text{lex_lesseq_allperm}$ between the vertex corresponding to the first item of the MATRIX collection and the vertices associated with all other items of the MATRIX collection. This is achieved by specifying that (1) an arc should start from the first item (i.e., $\text{matrix1.key} = 1$) and (2) an arc should not end on the first item (i.e., $\text{matrix2.key} > 1$). We finally state that all these arcs should belong to the final graph. Parts (A) and (B) of Figure 5.22 respectively show the initial and final graph associated with the Example slot.

![Figure 5.22: Initial and final graph of the allperm constraint](image)
### 5.17 among

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[39]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>among(NVAR, VARIABLES, VALUES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>between, count.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>NVAR : dvar</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES : collection(var−dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VALUES : collection(val−int)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>NVAR ≥ 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NVAR ≤</td>
<td>VARIABLES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES, var)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES, val)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>distinct(VARIABLES, val)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>NVAR is the number of variables of the collection VARIABLES that take their value in VALUES.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>(3, ⟨4, 5, 5, 4, 1⟩, ⟨1, 5, 8⟩)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The among constraint holds since exactly 3 values of the collection of variables ⟨4, 5, 5, 4, 1⟩ belong to the set of values {1, 5, 8}.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Typical</td>
<td>NVAR &gt; 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NVAR &lt;</td>
<td>VARIABLES</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>VALUES</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>VALUES</td>
<td>&gt; 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VALUES</td>
<td>&gt; 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VARIABLES</td>
<td>&gt;</td>
</tr>
<tr>
<td>Symmetries</td>
<td>• Items of VARIABLES are permutable.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Items of VALUES are permutable.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• An occurrence of a value of VARIABLES.var that belongs to VALUES.val (resp. does not belong to VALUES.val) can be replaced by any other value in VALUES.val (resp. not in VALUES.val).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arg. properties</td>
<td>• Functional dependency: NVAR determined by VARIABLES and VALUES.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Contractible wrt. VARIABLES when NVAR = 0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Contractible wrt. VARIABLES when NVAR =</td>
<td>VARIABLES</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>• Aggregate: NVAR(+), VARIABLES(union), VALUES(sunion).</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Remark
A similar constraint called \textit{between} was introduced in \textbf{CHIP} in 1990. The \textit{common} constraint can be seen as a generalisation of the \textit{among} constraint where we allow the \texttt{val} attributes of the VALUES collection to be domain variables.

A generalisation of this constraint when the values of VALUES are not initially fixed is called \textit{among\_var}.

When the variable \texttt{NVAR} (i.e., the first argument of the \textit{among} constraint) does not occur in any other constraints of the problem, it may be operationally more efficient to replace the \textit{among} constraint by a \textit{among\_low\_up} constraint where \texttt{NVAR} is replaced by the corresponding interval \([\texttt{NVAR}, \texttt{NVAR}]\). This stands for two reasons:

- First, by using an \textit{among\_low\_up} constraint rather than an \textit{among} constraint, we avoid the filtering algorithm related to \texttt{NVAR}.
- Second, unlike the \textit{among} constraint where we need to fix all its variables to get entailment, the \textit{among\_low\_up} constraint can be entailed before all its variables get fixed. As a result, this potentially avoid unnecessary calls to its filtering algorithm.

Algorithm
A filtering algorithm achieving arc-consistency was given by Bessière et al. in \cite{57, 60}.

Systems
\texttt{among} in \textbf{Choco}, \texttt{count} in \textbf{Gecode}, \texttt{among} in \textbf{JaCoP}, \texttt{among} in \textbf{MiniZinc}.

See also
\texttt{common} keyword: \texttt{arith, atleast, atmost (value constraint), count (counting constraint), counts (value constraint, counting constraint), discrepancy, max\_nvalue, min\_nvalue, nvalue (counting constraint)}.

\texttt{generalisation}: \texttt{among\_var} (constant replaced by variable).

\texttt{implies}: \texttt{among\_var, cardinality\_atmost}.

\texttt{related}: \texttt{roots (can be used for expressing among), sliding\_card\_skip0 (counting constraint on maximal sequences)}.

\texttt{shift of concept}: \texttt{among\_seq} (variable replaced by interval and constraint applied in a sliding way), \texttt{common}.

\texttt{soft variant}: \texttt{open\_among} (open constraint).

\texttt{specialisation}: \texttt{among\_diff\_0} (variable \in values replaced by variable different from 0), \texttt{among\_interval} (variable \in values replaced by variable \in interval), \texttt{among\_low\_up} (variable \in values replaced by interval), \texttt{among\_modulo} (list of values replaced by list of values v such that \(v \mod \text{QUOTIENT} = \text{REMAINDER}\), \texttt{exactly} (variable replaced by constant and values replaced by one single value).

\texttt{system of constraints}: \texttt{global\_cardinality} (count the number of occurrences of different values).

\texttt{used in graph description}: \texttt{in}.

\texttt{uses in its reformulation}: \texttt{count}.

Keywords
\texttt{characteristic of a constraint}: \texttt{automaton, automaton with counters, non-deterministic automaton}.

\texttt{constraint arguments}: pure functional dependency.

\texttt{constraint network structure}: \texttt{alpha-acyclic constraint network(2), Berge-acyclic constraint network}. 
constraint type: value constraint, counting constraint.
filtering: arc-consistency, SAT.
modelling: functional dependency.
Arc input(s) | VARIABLES
---|---
Arc generator | \( SELF \rightarrow \text{collection(Variables)} \)
Arc arity | 1
Arc constraint(s) | in(Variables.var, VALUES)
Graph property(ies) | NARC = NVAR

**Graph model**

The arc constraint corresponds to the unary constraint \( \text{in}(\text{variables.var}, \text{VALUES}) \) defined in this catalogue. Since this is a unary constraint we employ the \( SELF \) arc generator in order to produce an initial graph with a single loop on each vertex.

Parts (A) and (B) of Figure 5.23 respectively show the initial and final graph associated with the Example slot. Since we use the \( \text{NARC} \) graph property, the loops of the final graph are stressed in bold.

![Graph Model](image)

Figure 5.23: Initial and final graph of the among constraint
Automaton

Figure 5.24 depicts a first automaton that only accepts all the solutions of the among constraint. This automaton uses a counter in order to record the number of satisfied constraints of the form \( \text{VAR}_i \in \text{VALUES} \). To each variable \( \text{VAR}_i \) of the collection \( \text{VARIABLES} \) corresponds a 0-1 signature variable \( S_i \). The following signature constraint links \( \text{VAR}_i \) and \( S_i \): \( \text{VAR}_i \in \text{VALUES} \iff S_i \). The automaton counts the number of variables of the \( \text{VARIABLES} \) collection that take their value in \( \text{VALUES} \) and finally assigns this number to \( \text{NVAR} \).

![Figure 5.24: Automaton (with a counter) of the among constraint](image)

We now describe a second counter free automaton that also only accepts all the solutions of the among constraint. Without loss of generality, assume that the collection of variables \( \text{VARIABLES} \) contains at least one variable (i.e., \( |\text{VARIABLES}| \geq 1 \)). Let \( n \) and \( D \) respectively denote the number of variables of the collection \( \text{VARIABLES} \), and the union of the domains of the variables of \( \text{VARIABLES} \). Clearly, the maximum number of variables of \( \text{VARIABLES} \) that are assigned a value in \( \text{VALUES} \) cannot exceed the quantity \( m = \min(n, \text{NVAR}) \). The \( m + 2 \) states of the automaton that only accepts all the solutions of the among constraint can be defined in the following way:

- We have an initial state labelled by \( s_0 \).
- We have \( m \) intermediate states labelled by \( s_i \) (\( 1 \leq i \leq m \)). The intermediate states are indexed by the number of already encountered satisfied constraints of the form \( \text{VAR}_i \in \text{VALUES} \) from the initial state \( s_0 \) to the state \( s_i \).
- We have a final state labelled by \( s_F \).

Three classes of transitions are respectively defined in the following way:
1. There is a transition, labeled by \( j \) (\( j \in D \setminus \text{VALUES} \)), from every state \( s_i \), (\( i \in [0, m] \)), to itself.

2. There is a transition, labeled by \( j \) (\( j \in \text{VALUES} \)), from every state \( s_i \), (\( i \in [0, m - 1] \)), to the state \( s_{i+1} \).

3. There is a transition, labelled by \( i \), from every state \( s_i \), (\( i \in [0, m] \)), to the final state \( s_F \).

This leads to an automaton that has \( m \cdot |D| + |D \setminus \text{VALUES}| + m + 1 \) transitions. Since the maximum value of \( m \) is equal to \( n \), in the worst case we have \( n \cdot |D| + |D \setminus \text{VALUES}| + n + 1 \) transitions.

Figure 5.26 depicts a counter free non deterministic automaton associated with the \( \text{among} \) constraint under the hypothesis that (1) all variables of \( \text{VARIABLES} \) are assigned a value in \( \{0, 1, 2, 3\} \), (2) \( |\text{VARIABLES}| \) is equal to 3, (3) \( \text{VALUES} \) corresponds to odd values.

The sequence \( \text{VAR}_1, \text{VAR}_2, \ldots, \text{VAR}_{|\text{VARIABLES}|}, \text{NVAR} \) is passed to this automaton. A state \( s_i \) (\( 1 \leq i \leq 3 \)) represents the fact that \( i \) odd values were already encountered, while \( s_F \) represents the final state. A transition from \( s_i \) (\( 1 \leq i \leq 3 \)) to \( s_F \) is labelled by \( i \) and represents the fact that we can only go in the final state from a state that is compatible with the total number of odd values enforced by \( \text{NVAR} \). Note that non determinism only occurs if there is a non-empty intersection between the set of potential values that can be assigned to the variables of \( \text{VARIABLES} \) and the potential value of the \( \text{NVAR} \). While the counter free non deterministic automaton depicted by Figure 5.26 has 5 states and 18 transitions, its minimum-state deterministic counterpart shown in Figure 5.27 has 7 states and 23 transitions.

We make the following final observation. Since the \textbf{Symmetries} slot of the \( \text{among} \) constraint indicates that the variables of \( \text{VARIABLES} \) are permutatable, and since all incoming transitions to any state of the automaton depicted by Figure 5.26 are labelled with distinct values, we can mechanically construct from this automaton a counter free deterministic automaton that takes as input the sequence \( \text{NVAR}, \text{VAR}_3, \text{VAR}_2, \text{VAR}_1 \) rather than the sequence \( \text{VAR}_1, \text{VAR}_2, \text{VAR}_3, \text{NVAR} \). This is achieved by respectively making \( s_F \) and \( s_0 \) the initial and the final state, and by reversing each transition.
The sequence of variables \( \text{VAR}_1, \text{VAR}_2, \text{VAR}_3, \text{NVAR} \) is passed to the automaton.

Figure 5.26: Counter free non deterministic automaton of the among(\( \text{NVAR}, (\text{VAR}_1, \text{VAR}_2, \text{VAR}_3), (1, 3) \)) constraint assuming \( \text{VAR}_i \in [0, 3] \) \((1 \leq i \leq 3)\), with initial state \( s_0 \) and final state \( s_F \).
The sequence of variables $\text{VAR}_1, \text{VAR}_2, \text{VAR}_3, \text{NVAR}$ is passed to the automaton.

Figure 5.27: Counter free minimum-state deterministic automaton of the among($\text{NVAR}, \langle \text{VAR}_1, \text{VAR}_2, \text{VAR}_3 \rangle, \langle 1, 3 \rangle$) constraint assuming $\text{VAR}_i \in [0, 3]$ ($1 \leq i \leq 3$), with initial state $s_0$ and final state $s_F$. 
### 5.18 among_diff_0

**Origin**

Used in the automaton of `nvalue`.

**Constraint**

`among_diff_0(NVAR, VARIABLES)`

**Arguments**

- `NVAR` : `dvar`
- `VARIABLES` : `collection(var−dvar)`

**Restrictions**

- `NVAR ≥ 0`
- `NVAR ≤ |VARIABLES|`
  
  `required(VARIABLES, var)`

**Purpose**

`NVAR` is the number of variables of the collection `VARIABLES` that take a value different from 0.

**Example**

```
(3, (0, 5, 5, 0, 1))
```

The `among_diff_0` constraint holds since exactly 3 values of the collection of values `(0, 5, 5, 0, 1)` are different from 0.

**Typical**

- `NVAR > 0`
- `NVAR < |VARIABLES|`
- `|VARIABLES| > 1`
- `atleast(1, VARIABLES, 0)`

**Symmetries**

- Items of `VARIABLES` are permutable.
- An occurrence of a value of `VARIABLES.var` that is different from 0 can be replaced by any other value that is also different from 0.

**Arg. properties**

- **Functional dependency**: `NVAR` determined by `VARIABLES`.
- **Contractible** wrt. `VARIABLES` when `NVAR = 0`.
- **Contractible** wrt. `VARIABLES` when `NVAR = |VARIABLES|`.
- **Aggregate**: `NVAR(+), VARIABLES(union)`.

**See also**

**common keyword**: `nvalue` *(counting constraint)*.

**generalisation**: `among` *(variable ≠ 0 replaced by variable ∈ values)*.

**Keywords**

- **characteristic of a constraint**: joker value, automaton, automaton with counters.
- **constraint arguments**: pure functional dependency.
- **constraint network structure**: alpha-acyclic constraint network(2).
- **constraint type**: value constraint, counting constraint.
- **filtering**: arc-consistency.
- **modelling**: functional dependency.
Arc input(s) VARIABLES
Arc generator $SELF \rightarrow \text{collection}(\text{variables})$
Arc arity 1
Arc constraint(s) variables.var $\neq$ 0
Graph property(ies) NARC = NVAR

Graph model
Since this is a unary constraint we employ the $SELF$ arc generator in order to produce an initial graph with a single loop on each vertex. Parts (A) and (B) of Figure 5.28 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the loops of the final graph are stressed in bold.

![Figure 5.28: Initial and final graph of the among_diff_0 constraint](image_url)
Automaton

Figure 5.29 depicts the automaton associated with the among\textunderscore diff\textunderscore 0 constraint. To each variable \( \text{VAR}_i \) of the collection \( \text{VARIABLES} \) corresponds a 0-1 signature variable \( S_i \). The following signature constraint links \( \text{VAR}_i \) and \( S_i \): \( \text{VAR}_i \neq 0 \iff S_i \). The automaton counts the number of variables of the \( \text{VARIABLES} \) collection that take a value different from 0 and finally assigns this number to \( \text{NVAR} \).

\begin{align*}
\text{VAR}_1 &<> 0, \\
\{C=C+1\}
\end{align*}

\begin{align*}
\text{VAR}_1 &= 0 \\
\{\text{NVAR}=C\}
\end{align*}

Figure 5.29: Automaton of the among\textunderscore diff\textunderscore 0 constraint

\begin{align*}
\text{VAR}_1 &
\end{align*}

\begin{align*}
\text{VAR}_2 &
\end{align*}

\begin{align*}
\text{VAR}_n &
\end{align*}

\begin{align*}
\text{S}_1 
\end{align*}

\begin{align*}
\text{S}_2 
\end{align*}

\begin{align*}
\text{S}_n 
\end{align*}

\begin{align*}
Q_0 &= s \\
C_0 &= 0 \\
Q_1 
\end{align*}

\begin{align*}
Q_1 
\end{align*}

\begin{align*}
Q_n &= s \\
C_n &= \text{NVAR}
\end{align*}

Figure 5.30: Hypergraph of the reformulation corresponding to the automaton of the among\textunderscore diff\textunderscore 0 constraint
### 5.19 among_interval

**Origin**
Derived from among.

**Constraint**

`among_interval(NVAR, VARIABLES, LOW, UP)`

**Arguments**

- `NVAR` : `dvar`
- `VARIABLES` : `collection(var−dvar)`
- `LOW` : `int`
- `UP` : `int`

**Restrictions**

- `NVAR ≥ 0`
- `NVAR ≤ |VARIABLES|`
- `required(VARIABLES, var)`
- `LOW ≤ UP`

**Purpose**

`NVAR` is the number of variables of the collection `VARIABLES` taking a value that is located within interval `[LOW, UP]`.

**Example**

`(3, ⟨4, 5, 8, 4, 1⟩, 3, 5)`

The `among_interval` constraint holds since we have 3 values, namely 4, 5 and 4 that are situated within interval `[3, 5]`.

**Typical**

- `NVAR > 0`
- `NVAR < |VARIABLES|`
- `|VARIABLES| > 1`
- `LOW < UP`
- `LOW ≤ maxval(VARIABLES.var)`
- `UP ≥ minval(VARIABLES.var)`

**Symmetries**

- Items of `VARIABLES` are **permutable**.
- An occurrence of a value of `VARIABLES.var` that belongs to `[LOW, UP]` (resp. does not belong to `[LOW, UP]`) can be **replaced** by any other value in `[LOW, UP]` (resp. not in `[LOW, UP]`).

**Arg. properties**

- **Functional dependency**: `NVAR` determined by `VARIABLES`, `LOW` and `UP`.
- **Contractible** wrt. `VARIABLES` when `NVAR = 0`.
- **Contractible** wrt. `VARIABLES` when `NVAR = |VARIABLES|`.
- **Aggregate**: `NVAR(+)`, `VARIABLES(union)`, `LOW(id)`, `UP(id)`.

**Remark**

By giving explicitly all values of the interval `[LOW, UP]` the `among_interval` constraint can be modelled with the `among` constraint. However when `LOW = UP + 1` is a large quantity the `among_interval` constraint provides a more compact form.
See also

generalisation: among (variable in interval replaced by variable $\in$ values).

Keywords

characteristic of a constraint: automaton, automaton with counters.

constraint arguments: pure functional dependency.

constraint network structure: alpha-acyclic constraint network(2).

constraint type: value constraint, counting constraint.

filtering: arc-consistency.

modelling: interval, functional dependency.
Arc input(s) | VARIABLES
--- | ---
Arc generator | SELF→collection(variables)
Arc arity | 1
Arc constraint(s) | • LOW ≤ variables.var
• variables.var ≤ UP
Graph property(ies) | NARC= NVAR

Graph model
The arc constraint corresponds to a unary constraint. For this reason we employ the SELF arc generator in order to produce a graph with a single loop on each vertex.

Parts (A) and (B) of Figure 5.31 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the loops of the final graph are stressed in bold.

Figure 5.31: Initial and final graph of the among_interval constraint
Automaton

Figure 5.32 depicts the automaton associated with the \texttt{among\_interval} constraint. To each variable \texttt{VAR}_i of the collection \texttt{VARIABLES} corresponds a 0-1 signature variable \texttt{S}_i. The following signature constraint links \texttt{VAR}_i and \texttt{S}_i: \( \text{LOW} \leq \text{VAR}_i \land \text{VAR}_i \leq \text{UP} \iff \text{S}_i \). The automaton counts the number of variables of the \texttt{VARIABLES} collection that take their value in \([\text{LOW}, \text{UP}]\) and finally assigns this number to \texttt{NVAR}.

![Automaton Diagram](image)

Figure 5.33: Hypergraph of the reformulation corresponding to the automaton of the \texttt{among\_interval} constraint
5.20 **among_low_up**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[39]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>among_low_up(LOW, UP, VARIABLES, VALUES)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Arguments   | LOW : int  
UP : int  
VARIABLES : collection(var–dvar)  
VALUES : collection(val–int) |       |           |
| Restrictions| LOW ≥ 0  
LOW ≤ |VARIABLES|  
UP ≥ 0  
UP ≤ |VARIABLES|  
UP ≥ LOW  
required(VARIABLES, var)  
required(VARIABLES, val)  
distinct(VARIABLES, val) |       |           |
| Purpose     | Between LOW and UP variables of the VARIABLES collection are assigned a value of the VALUES collection. |       |           |
| Example     | \(\begin{pmatrix} 1, 2, (9, 2, 4, 5), \\
(0, 2, 4, 6, 8) \end{pmatrix}\) |       |           |
| Typical     | LOW < |VARIABLES|  
UP > 0  
LOW < UP  
|VARIABLES| > 1  
|VALUES| > 1  
|VALUES| > |VALUES|  
LOW > 0 ∨ UP < |VARIABLES| |       |           |
| Symmetries  | • Items of VARIABLES are permutable.  
• Items of VALUES are permutable.  
• LOW can be decreased to any value ≥ 0.  
• UP can be increased to any value ≤ |VARIABLES|.  
• An occurrence of a value of VARIABLES.var that belongs to VALUES.val (resp. does not belong to VALUES.val) can be replaced by any other value in VALUES.val (resp. not in VALUES.val). |       |           |
Arg. properties

- Contractible wrt. VARIABLES when UP = 0.
- Contractible wrt. VARIABLES when UP = |VARIABLES|.
- Aggregate: LOW(+), UP(+), VARIABLES(union), VALUES(sunion).

Algorithm

The among_low_up constraint is entailed if and only if the following two conditions hold:

1. The number of variables of the VARIABLES collection assigned a value of the VALUES collection is greater than or equal to LOW.
2. The number of variables of the VARIABLES collection that can potentially be assigned a value of the VALUES collection is less than or equal to UP.

Used in

among_seq, cycle_card_on_path, interval_and_count, sliding_card_skip0.

See also

assignment dimension added: interval_and_count (assignment dimension corresponding to intervals added).

generalisation: among (interval replaced by variable), sliding_card_skip0 (full sequence replaced by maximal sequences of non-zeros).

system of constraints: among_seq.

Keywords

characteristic of a constraint: automaton, automaton with counters.

constraint network structure: alpha-acyclic constraint network(2).

constraint type: value constraint, counting constraint.

filtering: arc-consistency, entailment.

final graph structure: acyclic, bipartite, no loop.
Arc input(s)  VARIABLES VALUES
Arc generator  \( \text{PRODUCT} \rightarrow \text{collection}(\text{variables, values}) \)
Arc arity  2
Arc constraint(s)  \( \text{variables.var} = \text{values.val} \)
Graph property(ies)  • \( \text{NARC} \geq \text{LOW} \)
                        • \( \text{NARC} \leq \text{UP} \)
Graph class  • \text{ACYCLIC}
                      • \text{BIPARTITE}
                      • \text{NO_LOOP}

Graph model  Each arc constraint of the final graph corresponds to the fact that a variable is assigned to a value that belong to the VALUES collection. The two graph properties restrict the total number of arcs to the interval \([\text{LOW}, \text{UP}]\).

Parts (A) and (B) of Figure 5.34 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

![Graph diagrams](A) (B)

Figure 5.34: Initial and final graph of the among_low_up constraint
Automaton

Figure 5.35 depicts the automaton associated with the \texttt{among\_low\_up} constraint. To each variable \texttt{VAR}_i of the collection \texttt{VARIABLES} corresponds a 0-1 signature variable \texttt{S}_i. The following signature constraint links \texttt{VAR}_i and \texttt{S}_i: \texttt{VAR}_i \in \texttt{VALUES} \Leftrightarrow \texttt{S}_i. The automaton counts the number of variables of the \texttt{VARIABLES} collection that take their value in \texttt{VALUES} and finally checks that this number is within the interval \([\texttt{LOW}, \texttt{UP}]\).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{automaton.png}
\caption{Automaton of the \texttt{among\_low\_up} constraint}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{hypergraph.png}
\caption{Hypergraph of the reformulation corresponding to the automaton of the \texttt{among\_low\_up} constraint}
\end{figure}
### 5.21 among_modulo

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from among.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>among_modulo(NVAR, VARIABLES, REMAINDER, QUOTIENT)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Arguments

- **NVAR**: dvar
- **VARIABLES**: collection(var–dvar)
- **REMAINDER**: int
- **QUOTIENT**: int

#### Restrictions

- NVAR ≥ 0
- NVAR ≤ |VARIABLES|
- required(VARIABLES, var)
- REMAINDER ≥ 0
- REMAINDER < QUOTIENT
- QUOTIENT > 0

#### Purpose

NVAR is the number of variables of the collection VARIABLES taking a value that is congruent to REMAINDER modulo QUOTIENT.

#### Example

- (3, ⟨4, 5, 8, 4, 1⟩, 0, 2)

In this example REMAINDER = 0 and QUOTIENT = 2 specifies that we count the number of even values taken by the different variables. As a consequence the among_modulo constraint holds since exactly 3 values of the collection ⟨4, 5, 8, 4, 1⟩ are even.

#### Typical

- NVAR > 0
- NVAR < |VARIABLES|
- |VARIABLES| > 1
- QUOTIENT > 1
- QUOTIENT < maxval(VARIABLES.var)

#### Symmetries

- Items of VARIABLES are permutable.
- An occurrence of a value u of VARIABLES.var such that u \(\mod\) QUOTIENT = REMAINDER (resp. u \(\mod\) QUOTIENT ≠ REMAINDER) can be replaced by any other value v such that v \(\mod\) QUOTIENT = REMAINDER (resp. v \(\mod\) QUOTIENT ≠ REMAINDER).

#### Arg. properties

- **Functional dependency**: NVAR determined by VARIABLES, REMAINDER and QUOTIENT.
- **Contractible** wrt. VARIABLES when NVAR = 0.
- **Contractible** wrt. VARIABLES when NVAR = |VARIABLES|.
- **Aggregate**: NVAR(+), VARIABLES(union), REMAINDER(id), QUOTIENT(id).
Remark

By giving explicitly all values \( v \) that satisfy the equality \( v \mod QUOTIENT = REMAINDER \), the among_modulo constraint can be modelled with the among constraint. However the among_modulo constraint provides a more compact form.

See also

generalisation: among (list of values \( v \) such that \( v \mod QUOTIENT = REMAINDER \) replaced by list of values).

Keywords

characteristic of a constraint: modulo, automaton, automaton with counters.
constraint arguments: pure functional dependency.
constraint network structure: alpha-acyclic constraint network(2).
constraint type: value constraint, counting constraint.
filtering: arc-consistency.
modelling: functional dependency.
Arc input(s)  VARIABLES
Arc generator  \( \text{SELF} \rightarrow \text{collection} (\text{variables}) \)
Arc arity  1
Arc constraint(s)  \( \text{variables.var mod QUOTIENT = REMAINDER} \)
Graph property(ies)  \( \text{NARC} = \text{NVAR} \)

Graph model

The arc constraint corresponds to a unary constraint. For this reason we employ the \textit{SELF} arc generator in order to produce a graph with a single loop on each vertex.

Parts (A) and (B) of Figure 5.37 respectively show the initial and final graph associated with the \textbf{Example} slot. Since we use the \textit{NARC} graph property, the loops of the final graph are stressed in bold.

![Graph Diagram]

\( \text{NARC} = 3 \)

Figure 5.37: Initial and final graph of the \texttt{among\_modulo} constraint
Automaton

Figure 5.38 depicts the automaton associated with the among modulo constraint. To each variable $VAR_i$ of the collection $VARIABLES$ corresponds a 0-1 signature variable $S_i$. The following signature constraint links $VAR_i$ and $S_i$: $VAR_i \mod QUOTIENT = REMAINDER \iff S_i$.

$$\begin{align*}
\text{VAR}_1 \mod QUOTIENT &= REMAINDER, \\
\{C=0\} &\implies S: NVAR=C \implies VAR_1 \mod QUOTIENT \not= REMAINDER.
\end{align*}$$

Figure 5.38: Automaton of the among modulo constraint

Figure 5.39: Hypergraph of the reformulation corresponding to the automaton of the among modulo constraint
5.22  among_seq

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[39]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>among_seq(Low, Up, Seq, VARIABLES, VALUES)</td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td>sequence.</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>LOW : int</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UP : int</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEQ : int</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VALUES : collection(val−int)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>LOW ≥ 0</td>
<td></td>
</tr>
</tbody>
</table>
Symmetries

- Items of VARIABLES can be reversed.
- Items of VALUES are permutable.
- LOW can be decreased to any value $\geq 0$.
- UP can be increased to any value $\leq SEQ$.
- An occurrence of a value of VARIABLES var that belongs to VALUES val (resp. does not belong to VALUES val) can be replaced by any other value in VALUES val (resp. not in VALUES val).

Arg. properties

- Contractible wrt. VARIABLES when UP = 0.
- Contractible wrt. VARIABLES when SEQ = 1.
- Prefix-contractible wrt. VARIABLES.
- Suffix-contractible wrt. VARIABLES.

Usage

The among_seq constraint occurs in many timetabling problems. As a typical example taken from [401], consider for instance a nurse-rostering problem where each nurse can work at most 2 night shifts during every period of 7 consecutive days.

Algorithm

Beldiceanu and Carlsson [29] have proposed a first incomplete filtering algorithm for the among_seq constraint. Later on, W.-J. van Hoeve et al. proposed two filtering algorithms [401] establishing arc-consistency as well as an incomplete filtering algorithm based on dynamic programming concepts. In 2007 Brand et al. came up with a reformulation [82] that provides a complete filtering algorithm. One year later, Maher et al. use a reformulation in term of a linear program [254] where (1) each coefficient is an integer in $\{-1, 0, 1\}$, (2) each column has a block of consecutive 1’s or −1’s. From this reformulation they derive a flow model that leads to an algorithm that achieves a complete filtering in $O(n^2)$ along a branch of the search tree.

Systems

sequence in Gecode, sequence in JaCoP.

See also

generalisation: sliding_distribution (single set of values replaced by individual values).

part of system of constraints: among_low_up.
root concept: among.
used in graph description: among_low_up.

Keywords

characteristic of a constraint: hypergraph.
combinatorial object: sequence.
constraint type: system of constraints, decomposition, sliding sequence constraint.
filtering: arc-consistency, linear programming, flow.
<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>$PATH \mapsto \text{collection}$</td>
</tr>
<tr>
<td>Arc arity</td>
<td>$\text{SEQ}$</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>$\text{among}<em>\text{low}</em>\text{up}(\text{LOW}, \text{UP}, \text{collection}, \text{VALUES})$</td>
</tr>
<tr>
<td>Graph property(ies)</td>
<td>$\text{NARC} =</td>
</tr>
</tbody>
</table>

**Graph model**

A constraint on sliding sequences of consecutive variables. Each vertex of the graph corresponds to a variable. Since they link $\text{SEQ}$ variables, the arcs of the graph correspond to hyperarcs. In order to link $\text{SEQ}$ consecutive variables we use the arc generator $PATH$. The constraint associated with an arc corresponds to the $\text{among}_\text{low}_\text{up}$ constraint defined at another entry of this catalogue.

**Signature**

Since we use the $PATH$ arc generator with an arity of $\text{SEQ}$ on the items of the VARIABLES collection, the expression $|\text{VARIABLES}| - \text{SEQ} + 1$ corresponds to the maximum number of arcs of the final graph. Therefore we can rewrite the graph property $\text{NARC} = |\text{VARIABLES}| - \text{SEQ} + 1$ to $\text{NARC} \geq |\text{VARIABLES}| - \text{SEQ} + 1$ and simplify $\text{NARC}$ to $\text{NARC}$. 
5.23 among_var

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Generalisation of among</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>among_var(NVAR, VARIABLES, VALUES)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>NVAR : dvar</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VALUES : collection(val−dvar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>NVAR ≥ 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NVAR ≤</td>
<td>VARIABLES</td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES, var)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>required(VALEUES, val)</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>NVAR is the number of variables of the collection VARIABLES that are equal to one of the variables of the collection VALUES.</td>
<td></td>
</tr>
</tbody>
</table>
| Example     | \[
|             | (3, \langle 4, 5, 5, 4, 1 \rangle, \langle 1, 5, 8, 1 \rangle) \]
|             | The among_var constraint holds since exactly 3 values of the collection of variables \langle 4, 5, 5, 4, 1 \rangle occurs within the collection \langle 1, 5, 8, 1 \rangle. |
| Typical     | |VARIABLES| > 1 |
|             | |VALUES| > 1 |
|             | |VARIABLES| > |VALUES| |
| Symmetries  | • Items of VARIABLES are permutable. |
|             | • Items of VALUES are permutable. |
|             | • All occurrences of two distinct values in VARIABLES.var or VALUES.val can be swapped; all occurrences of a value in VARIABLES.var or VALUES.val can be renamed to any unused value. |
|             | • An occurrence of a value of VARIABLES.var that belongs to VALUES.val (resp. does not belong to VALUES.val) can be replaced by any other value in VALUES.val (resp. not in VALUES.val). |
| Arg. properties | • Functional dependency: NVAR determined by VARIABLES and VALUES. |
|             | • Contractible wrt. VARIABLES when NVAR = 0. |
|             | • Contractible wrt. VARIABLES when NVAR = |VARIABLES|. |
|             | • Aggregate: NVAR(+), VARIABLES(union), VALUES(union). |
| Systems     | among in Choco, count in Gecode, amongvar in JaCoP. |
See also

implied by: among.
related: common.
specialisation: among (variable replaced by constant within list of values VALUES).
uses in its reformulation: min_n.

Keywords

constraint arguments: pure functional dependency.
constraint type: counting constraint.
final graph structure: acyclic, bipartite, no loop.
modelling: functional dependency.
Arc input(s) | VARIABLES VALUES
--- | ---
Arc generator | PRODUCT↦→collection(variables,values)
Arc arity | 2
Arc constraint(s) | variables.var = values.val
Graph property(ies) | NSOURCE = NVAR
Graph class | • ACYCLIC
• BIPARTITE
• NO_LOOP

Graph model

Parts (A) and (B) of Figure 5.40 respectively show the initial and final graph associated with the Example slot. Since we use the NSOURCE graph property, the source vertices of the final graph are stressed with a double circle. Since the final graph has only 3 sources the variables NVAR is fixed to 3.

Figure 5.40: Initial and final graph of the among_var constraint
5.24 and

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Logic</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>and(VAR, VARIABLES)</td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td>rel.</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VAR : dvar, VARIABLES : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>VAR ≥ 0, VAR ≤ 1,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>VARIABLES ≥ 2, required(VARIABLES, var)</td>
</tr>
<tr>
<td>Purpose</td>
<td>Let VARIABLES be a collection of 0-1 variables VAR₁, VAR₂, ..., VARₙ (n ≥ 2). Enforce VAR = VAR₁ ∧ VAR₂ ∧ ... ∧ VARₙ.</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>(0, (0, 0)), (0, (0, 1)), (0, (1, 0)), (1, (1, 1)), (0, (1, 0, 1))</td>
<td></td>
</tr>
<tr>
<td>Symmetry</td>
<td>Items of VARIABLES are permutable.</td>
<td></td>
</tr>
<tr>
<td>Arg. properties</td>
<td>• Functional dependency: VAR determined by VARIABLES.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Extensible wrt. VARIABLES when VAR = 0.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Aggregate: VAR(∧), VARIABLES(union).</td>
<td></td>
</tr>
<tr>
<td>Systems</td>
<td>reifiedAnd in Choco, rel in Gecode, andbool in JaCoP, #/\ in SICStus.</td>
<td></td>
</tr>
<tr>
<td>See also</td>
<td>common keyword: clause_and, equivalent, imply, nand, nor, or, xor(Boolean constraint).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>implies: atleast_nvalue, minimum.</td>
<td></td>
</tr>
<tr>
<td>Keywords</td>
<td>characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>constraint arguments: pure functional dependency.</td>
<td></td>
</tr>
</tbody>
</table>
constraint network structure: Berge-acyclic constraint network.
constraint type: Boolean constraint.
filtering: arc-consistency.
modelling: functional dependency.
Automaton

Figure 5.41 depicts the automaton associated with the and constraint. To the first argument \( \text{VAR} \) of the and constraint corresponds the first signature variable. To each variable \( \text{VAR}_i \) of the second argument \( \text{VARIABLES} \) of the and constraint corresponds the next signature variable. There is no signature constraint.

![Automaton Diagram]

Figure 5.41: Automaton of the and constraint

![Hypergraph Diagram]

Figure 5.42: Hypergraph of the reformulation corresponding to the automaton of the and constraint
### 5.25 arith

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Used in the definition of several automata</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>arith(VARIABLES, RELOP, VALUE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td>rel.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Arguments   | VARIABLES : collection(var–dvar)  
 | RELOP : atom  
 | VALUE : int  |
| Restrictions| required(VARIABLES, var)  
 | RELOP ∈ [\(=, \neq, <, \geq, >, \leq\)] |
| Purpose     | Enforce for all variables var of the VARIABLES collection to have var RELOP VALUE. |
| Example     | \((4, 5, 7, 4, 5), <, 9)\) |
| Typical     | \(|\text{VARIABLES}| > 1  
 | RELOP ∈ [=]  |
| Symmetries  | ● Items of VARIABLES are **permutable**.  
 | ● An occurrence of a value of VARIABLES.var can be **replaced** by any value of VARIABLES.var. |
| Arg. properties | **Contractible** wrt. VARIABLES. |
| Systems     | eq in Choco, neq in Choco, geq in Choco, gt in Choco, leq in Choco, lt in Choco,  
 | rel in Gecode, #< in SICStus, #\= in SICStus, #> in SICStus, #\= in SICStus, #\= in SICStus. |
| Used in     | arith_sliding. |
| See also    | **common keyword**: among, count (value constraint).  
 | **generalisation**: arith_or (variable RELOP VALUE replaced by variable RELOP VALUE  
 | \lor variable RELOP VALUE).  
 | **system of constraints**: arith_sliding. |
Keywords

**characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint,

**constraint network structure:** Berge-acyclic constraint network.

**constraint type:** decomposition, value constraint.

**filtering:** arc-consistency.

**modelling:** domain definition.
### Arc input(s)
VARIABLES

### Arc generator
\[ \text{SELF} \rightarrow \text{collection}(\text{variables}) \]

### Arc arity
1

### Arc constraint(s)
variables.var \text{RELOP} \text{VALUE}

### Graph property(ies)
\[ \text{NARC} = |\text{VARIABLES}| \]

### Graph model
Parts (A) and (B) of Figure 5.43 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the loops of the final graph are stressed in bold.

![Graph Figure](image)

Figure 5.43: Initial and final graph of the arith constraint
Automaton

Figure 5.44 depicts the automaton associated with the arith constraint. To each variable VAR\(_i\) of the collection VARIABLES corresponds a 0-1 signature variable S\(_i\). The following signature constraint links VAR\(_i\) and S\(_i\): VAR\(_i\), RELOP VALUE ⇔ S\(_i\). The automaton enforces for each variable VAR\(_i\) the condition VAR\(_i\), RELOP VALUE.

Figure 5.44: Automaton of the arith constraint

Figure 5.45: Hypergraph of the reformulation corresponding to the automaton of the arith constraint
5.26  \texttt{arith\_or}

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Used in the definition of several automata</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>\texttt{arith_or(VARIABLES1, VARIABLES2, RELOP, VALUE)}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Arguments   | \begin{itemize} 
  \item \texttt{VARIABLES1} : \texttt{collection(var−dvar)}
  \item \texttt{VARIABLES2} : \texttt{collection(var−dvar)}
  \item \texttt{RELOP} : \texttt{atom}
  \item \texttt{VALUE} : \texttt{int}
\end{itemize} |       |           |
| Restrictions| \begin{itemize} 
  \item \texttt{required(VARIABLES1, var)}
  \item \texttt{required(VARIABLES2, var)}
  \item \texttt{|VARIABLES1| = |VARIABLES2|}
  \item \texttt{RELOP \in \{=, \neq, <, \geq, >, \leq\}}
\end{itemize} |       |           |
| Purpose     | \begin{itemize} 
  \item \textbf{Enforce for all pairs of variables} \texttt{var1\_i, var2\_i} \textbf{of the VARIABLES1 and VARIABLES2 collections to have} \texttt{var1\_i RELOP VALUE} \texttt{\lor var2\_i RELOP VALUE}.
\end{itemize} |       |           |
| Example     | \begin{itemize} 
  \item \texttt{(\{0, 1, 0, 0, 1\}, \{0, 0, 0, 1, 0\}, =, 0)}
\end{itemize} |       |           |
| Typical     | \begin{itemize} 
  \item \texttt{|VARIABLES1| > 0}
  \item \texttt{RELOP \in \{=\}}
\end{itemize} |       |           |
| Symmetry    | \begin{itemize} 
  \item \textbf{Items of VARIABLES1 and VARIABLES2 are permutable} (same permutation used).
\end{itemize} |       |           |
| Arg. properties | \begin{itemize} 
  \item Contractible \texttt{wrt. VARIABLES1 and VARIABLES2} (remove items from same position).
\end{itemize} |       |           |
| See also    | \begin{itemize} 
  \item \textbf{specialisation: arith} (variable RELOP VALUE \lor variable RELOP VALUE replaced by variable RELOP VALUE).
\end{itemize} |       |           |
| Keywords    | \begin{itemize} 
  \item \textbf{characteristic of a constraint:} automaton, automaton without counters, reified automaton constraint.
  \item \textbf{constraint network structure:} Berge-acyclic constraint network.
  \item \textbf{constraint type:} decomposition, value constraint.
  \item \textbf{filtering:} arc-consistency.
  \item \textbf{final graph structure:} acyclic, bipartite, no loop.
  \item \textbf{modelling:} disjunction.
\end{itemize} |       |           |
Arc input(s) VARIABLES1 VARIABLES2
Arc generator \( PRODUCT(=) \rightarrow \text{collection}(\text{variables1}, \text{variables2}) \)
Arc arity 2
Arc constraint(s) \text{variables1}.\text{var RELOP VALUE} \lor \text{variables2}.\text{var RELOP VALUE}
Graph property(ies) \text{NARC} = |\text{VARIABLES1}|
Graph class
- ACYCLIC
- BIPARTITE
- NO_LOOP

Graph model
Parts (A) and (B) of Figure 5.46 respectively show the initial and final graphs associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

Figure 5.46: Initial and final graph of the arith_or constraint
Automaton

Figure 5.47 depicts the automaton associated with the \texttt{arith\_or} constraint. Let \texttt{VAR1}_i and \texttt{VAR2}_i be the \(i\)th variables of the \texttt{VARIABLES1} and \texttt{VARIABLES2} collections. To each pair of variables \((\texttt{VAR1}_i, \texttt{VAR2}_i)\) corresponds a signature variable \(S_i\). The following signature constraint links \(\texttt{VAR1}_i, \texttt{VAR2}_i\), and \(S_i\): \(\texttt{VAR1}_i \texttt{RELOP VALUE} \lor \texttt{VAR2}_i \texttt{RELOP VALUE} \Leftrightarrow S_i\). The automaton enforces for each pair of variables \(\texttt{VAR1}_i, \texttt{VAR2}_i\) the condition \(\texttt{VAR1}_i \texttt{RELOP VALUE} \lor \texttt{VAR2}_i \texttt{RELOP VALUE}\).

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{automaton.png}
\caption{Automaton of the \texttt{arith\_or} constraint}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{hypergraph.png}
\caption{Hypergraph of the reformulation corresponding to the automaton of the \texttt{arith\_or} constraint}
\end{figure}
5.27 arith_sliding

### Description

<table>
<thead>
<tr>
<th>Variable Collection</th>
<th>Link Type</th>
<th>Graph Type</th>
<th>Automaton Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARIABLES</td>
<td>collection</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Constraint

arith_sliding(VARIABLES, RELOP, VALUE)

### Arguments

- **VARIABLES**: collection(var−dvar)
- **RELOP**: atom
- **VALUE**: int

### Restrictions

- **required**(VARIABLES, var)
- **RELOP** ∈ [=, ≠, <, ≥, >, ≤]

### Purpose

Enforce for all sequences of variables var_1, var_2, ..., var_i (1 ≤ i ≤ |VARIABLES|) of the VARIABLES collection to have (var_1 + var_2 + ... + var_i) RELOP VALUE.

### Example

\[
\begin{pmatrix}
\text{var} - 0, \\
\text{var} - 0, \\
\text{var} - 1, \\
\langle \text{var} - 2, \rangle, <, 4 \\
\text{var} - 0, \\
\text{var} - 0, \\
\text{var} - 3
\end{pmatrix}
\]

The arith_sliding constraint holds since all the following seven inequalities hold:

- 0 < 4,
- 0 + 0 < 4,
- 0 + 0 + 1 < 4,
- 0 + 0 + 1 + 2 < 4,
- 0 + 0 + 1 + 2 + 0 < 4,
- 0 + 0 + 1 + 2 + 0 + 0 < 4,
- 0 + 0 + 1 + 2 + 0 + 0 − 3 < 4.

### Typical

- |VARIABLES| > 1
- **RELOP** ∈ [<, ≥, >, ≤]

### Arg. properties

- **Contractible** wrt. VARIABLES when **RELOP** ∈ [<, ≤] and minval(VARIABLES, var) ≥ 0.
- **Suffix-contractible** wrt. VARIABLES.
See also

common keyword: `sum_cstr (arithmetic constraint)`.
part of system of constraints: `arith`.
used in graph description: `arith`.

Keywords

characteristic of a constraint: hypergraph, automaton, automaton with counters.
combinatorial object: sequence.
constraint type: arithmetic constraint, decomposition, sliding sequence constraint.
<table>
<thead>
<tr>
<th><strong>Arc input(s)</strong></th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arc generator</strong></td>
<td>PATH, I→collection</td>
</tr>
<tr>
<td><strong>Arc arity</strong></td>
<td>*</td>
</tr>
<tr>
<td><strong>Arc constraint(s)</strong></td>
<td>arith(collection, RELOP, VALUE)</td>
</tr>
<tr>
<td><strong>Graph property(ies)</strong></td>
<td>NARC =</td>
</tr>
</tbody>
</table>
Automaton

Figure 5.49 depicts the automaton associated with the arith_sliding constraint. To each item of the collection VARIABLES corresponds a signature variable $S_i$ that is equal to 0.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure5_49}
\caption{Automaton of the arith_sliding constraint}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure5_50}
\caption{Hypergraph of the reformulation corresponding to the automaton of the arith_sliding constraint}
\end{figure}
5.28 assign_and_counts

**DESCRIPTION**

Origin

N. Beldiceanu

Constraint

assign_and_counts(COLOURS, ITEMS, RELOP, LIMIT)

Arguments

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOURS</td>
<td>collection(val−int)</td>
</tr>
<tr>
<td>ITEMS</td>
<td>collection(bin−dvar,colour−dvar)</td>
</tr>
<tr>
<td>RELOP</td>
<td>atom</td>
</tr>
<tr>
<td>LIMIT</td>
<td>dvar</td>
</tr>
</tbody>
</table>

Restrictions

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>required(COLOURS.val)</td>
</tr>
<tr>
<td>distinct(COLOURS.val)</td>
</tr>
<tr>
<td>required(ITEMS,[bin.colour])</td>
</tr>
<tr>
<td>RELOP ∈ [=, ≠, &lt;, ≥, &gt;, ≤]</td>
</tr>
</tbody>
</table>

**Purpose**

Given several items (each of them having a specific colour that may not be initially fixed), and different bins, assign each item to a bin, so that the total number \( n \) of items of colour \( \text{COLOURS} \) in each bin satisfies the condition \( n \ \text{RELOP} \ \text{LIMIT} \).

**Example**

\[
\begin{pmatrix}
\langle 4 \rangle, \\
\langle \text{bin−1 colour−4}, \\
\text{bin−3 colour−4}, \\
\text{bin−1 colour−4}, \\
\text{bin−1 colour−5} \rangle, \leq, 2
\end{pmatrix}
\]

Figure 5.51 shows the solution associated with the example. The items and the bins are respectively represented by little squares and by the different columns. Each little square contains the value of the key attribute of the item to which it corresponds. The items for which the colour attribute is equal to 4 are located under the thick line. The assign_and_counts constraint holds since for each used bin (i.e., namely bins 1 and 3) the number of assigned items for which the colour attribute is equal to 4 is less than or equal to the limit 2.

**Figure 5.51:** Assignment of the items to the bins
Typical

| COLOURS | > 0
| ITEMS | > 1
| \texttt{range}(ITEMS.bin) | > 1
| RELOP \in [\texttt{<}, \texttt{\leq}] |
| LIMIT | > 0
| LIMIT | < |ITEMS|

Symmetries

- Items of COLOURS are \textit{permutable}.
- Items of ITEMS are \textit{permutable}.
- All occurrences of two distinct values of ITEMS.bin can be \textit{swapped}; all occurrences of a value of ITEMS.bin can be \textit{renamed} to any unused value.

Arg. properties

- \textit{Contractible} wrt. ITEMS when \texttt{RELOP} \in [\texttt{<}, \texttt{\leq}].
- \textit{Extensible} wrt. ITEMS when \texttt{RELOP} \in [\texttt{\geq}, \texttt{>}].

Usage

Some persons have pointed out that it is impossible to use constraints such as \texttt{among}, \texttt{atleast}, \texttt{atmost}, \texttt{count}, or \texttt{global_cardinality} if the set of variables is not initially known. However, this is for instance required in practice for some timetabling problems.

See also

- \texttt{assignment dimension removed}: \texttt{count}, \texttt{counts}.
- \texttt{used in graph description}: \texttt{counts}.

Keywords

- \texttt{application area}: assignment.
- \texttt{characteristic of a constraint}: coloured, automaton, automaton with array of counters, derived collection.
- \texttt{final graph structure}: acyclic, bipartite, no loop.
- \texttt{modelling}: assignment dimension.
Derived Collection

\[
\text{col(values-collection(val-int),item(val-\text{COLOURS.val})])}
\]

Arc input(s)

ITEMS ITEMS

Arc generator

PRODUCT \rightarrow \text{collection}(\text{items1,items2})

Arc arity

2

Arc constraint(s)

\text{items1.bin} = \text{items2.bin}

Graph class

\begin{itemize}
  \item ACYCLIC
  \item BIPARTITE
  \item NO_LOOP
\end{itemize}

Sets

SUCC \mapsto \begin{bmatrix}
\text{source}, \\
\text{variables} - \text{col} \left(\begin{array}{c}
\text{VALUES-collection(var-dvar)}, \\
\text{item(var-\text{ITEMS.colour})}
\end{array}\right)
\end{bmatrix}

Constraint(s) on sets

\text{counts(values,variables,RELOP,LIMIT)}

Graph model

We enforce the \text{counts} constraint on the colour of the items that are assigned to the same bin.

Parts (A) and (B) of Figure 5.52 respectively show the initial and final graph associated with the \text{Example} slot. The final graph consists of the following two connected components:

- The connected component containing six vertices corresponds to the items that are assigned to bin 1.
- The connected component containing two vertices corresponds to the items that are assigned to bin 3.

![Figure 5.52: Initial and final graph of the assign_and_counts constraint](image)

The \text{assign_and_counts} constraint holds since for each set of successors of the vertices of the final graph no more than two items take colour 4.
Figure 5.53 depicts the automaton associated with the `assign_and_counts` constraint. To each colour attribute $\text{COLOUR}_i$ of the collection $\text{ITEMS}$ corresponds a 0-1 signature variable $S_i$. The following signature constraint links $\text{COLOUR}_i$ and $S_i$: $\text{COLOUR}_i \in \text{COLOURS} \Leftrightarrow S_i$. For all items of the collection $\text{ITEMS}$ for which the colour attribute takes its value in $\text{COLOURS}$, counts for each value assigned to the bin attribute its number of occurrences $n$, and finally imposes the condition $n \text{ RELOP LIMIT}$.

\[
\text{arith}(C, \text{RELOP}, \text{LIMIT}) \rightarrow \begin{cases} 
\text{not} \text{ in}(\text{COLOUR}_i, \text{COLOURS}) \\
S_i \\
in(\text{COLOUR}_i, \text{COLOURS}), \{C[\text{BIN}_i]=C[\text{BIN}_i]+1\} 
\end{cases}
\]

Figure 5.53: Automaton of the `assign_and_counts` constraint
5.29 **assign_and_nvalues**

**DESCRIPTION**

Derived from `assign_and_counts` and `nvalues`.

**Constraint**

`assign_and_nvalues(ITEMS, RELOP, LIMIT)`

**Arguments**

- `ITEMS`: `collection(bin−dvar, value−dvar)`
- `RELOP`: `atom`
- `LIMIT`: `dvar`

**Restrictions**

- `required(ITEMS, [bin, value])`
- `RELOP ∈ [=, ≠, <, ≥, >, ≤]`

**Purpose**

Given several items (each of them having a specific value that may not be initially fixed), and different bins, assign each item to a bin, so that the number \( n \) of distinct values in each bin satisfies the condition \( n \ RELOP LIMIT \).

**Example**

\[
\begin{pmatrix}
\text{bin} − 2 & \text{value} = 3,
\text{bin} − 1 & \text{value} = 5,
\text{bin} − 2 & \text{value} = 3,
\text{bin} − 2 & \text{value} = 3,
\text{bin} − 2 & \text{value} = 4
\end{pmatrix} \leq 2
\]

Figure 5.54 depicts the solution corresponding to the example. The `assign_and_nvalues` constraint holds since for each used bin (i.e., namely bins 1 and 2) the number of distinct colours of the corresponding assigned items is less than or equal to the limit 2.

![Figure 5.54: An assignment with at most two distinct values in parallel](image-url)

Figure 5.54: An assignment with at most two distinct values in parallel
**Typical**

- $|\text{ITEMS}| > 1$
- $\text{range}(\text{ITEMS.bin}) > 1$
- $\text{range}(\text{ITEMS.value}) > 1$
- $\text{RELOP} \in [<, \leq]$
- $\text{LIMIT} > 1$
- $\text{LIMIT} < |\text{ITEMS}|$

**Symmetries**

- Items of $\text{ITEMS}$ are *permutable*.
- All occurrences of two distinct values of $\text{ITEMS.bin}$ can be *swapped*; all occurrences of a value of $\text{ITEMS.bin}$ can be *renamed* to any unused value.

**Arg. properties**

- *Contractible* wrt. $\text{ITEMS}$ when $\text{RELOP} \in [<, \leq]$.
- *Extensible* wrt. $\text{ITEMS}$ when $\text{RELOP} \in [\geq, >]$.

**Usage**

Let us give two examples where the `assign_and_nvalues` constraint is useful:

- Quite often, in bin-packing problems, each item has a specific type, and one wants to assign items of similar type to each bin.
- In a vehicle routing problem, one wants to restrict the number of towns visited by each vehicle. Note that several customers may be located at the same town. In this example, each bin would correspond to a vehicle, each item would correspond to a visit to a customer, and the colour of an item would be the location of the corresponding customer.

**See also**

- *assignment dimension removed*: `nvalue`, `nvalues`.
- *common keyword*: `nvalues_except_0` (*number of distinct values*).
- *related*: `roots`.
- *used in graph description*: `nvalues`.

**Keywords**

- *application area*: `assignment`.
- *final graph structure*: `acyclic`, `bipartite`, `no loop`.
- *modelling*: `assignment dimension`, `number of distinct values`. 
Arc input(s) | ITEMS, ITEMS
--- | ---
Arc generator | \( \text{PRODUCT} \rightarrow \text{collection}(\text{items1, items2}) \)
Arc arity | 2
Arc constraint(s) | \( \text{items1.bin} = \text{items2.bin} \)
Graph class | • ACYCLIC
 • BIPARTITE
 • NO_LOOP
Sets | \( \text{SUCC} \mapsto [\text{source, variables - col(VARIABLES - collection(var - dvar),)}] \)
Constraint(s) on sets | \( \text{nvalues(variables, RELOP, LIMIT)} \)

Graph model
We enforce the \text{nvalues} constraint on the items that are assigned to the same bin.

Parts (A) and (B) of Figure 5.55 respectively show the initial and final graph associated with the Example slot. The final graph consists of the following two connected components:

- The connected component containing 8 vertices corresponds to the items that are assigned to bin 2.
- The connected component containing 2 vertices corresponds to the items that are assigned to bin 1.

![Figure 5.55: Initial and final graph of the assign_and_nvalues constraint](image)

The \text{assign_and_nvalues} constraint holds since for each set of successors of the vertices of the final graph no more than two distinct values are used:

- The unique item assigned to bin 1 uses value 5.
- Items assigned to bin 2 use values 3 and 4.
### 5.30 at least

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>CHIP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>atleast(N, \text{VARIABLES}, \text{VALUE})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td>count.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Arguments   | \(N\ : \text{int}\)  
\(\text{VARIABLES} : \text{collection(var\_dvar)}\)  
\(\text{VALUE} : \text{int}\) |
| Restrictions| \(N \geq 0\)  
\(N \leq |\text{VARIABLES}|\)  
\(\text{required}(\text{VARIABLES}, \text{var})\) |
| Purpose     | At least \(N\) variables of the \(\text{VARIABLES}\) collection are assigned value \(\text{VALUE}\). |
| Example     | \((2,\langle 4,2,4,5\rangle,4)\)  
The \texttt{atleast} constraint holds since at least 2 values of the collection \(\langle 4,2,4,5\rangle\) are equal to value 4. |
| Typical     | \(N > 0\)  
\(N < |\text{VARIABLES}|\)  
\(|\text{VARIABLES}| > 1\) |
| Symmetries  | • Items of \(\text{VARIABLES}\) are \texttt{permutable}.  
• \(N\) can be \texttt{decreased} to any value \(\geq 0\).  
• An occurrence of a value of \(\text{VARIABLES}\).\texttt{var} that is different from \(\text{VALUE}\) can be \texttt{replaced} by any other value. |
| Arg. properties | \texttt{Extensible} wrt. \texttt{VARIABLES}. |
| Systems     | \texttt{occurrenceMin} in \texttt{Choco}, \texttt{count} in \texttt{Gecode}, \texttt{atleast} in \texttt{Gecode}, \texttt{count} in \texttt{JaCoP}, \texttt{at}\texttt{least} in \texttt{MiniZinc}, \texttt{count} in \texttt{SICStus}. |
| Used in     | \texttt{alldifferent\_except\_0}, \texttt{among\_diff\_0}, \texttt{atmost}, \texttt{int\_value\_precede}, \texttt{ith\_pos\_different\_from\_0}, \texttt{minimum\_except\_0}, \texttt{nvalues\_except\_0}, \texttt{period\_except\_0}, \texttt{sliding\_card\_skip0}, \texttt{weighted\_partial\_alldiff}. |
See also

- **common keyword**: among (value constraint).
- **comparison swapped**: atmost.
- **implied by**: exactly ($\geq N$ replaced by $= N$).
- **related**: roots.
- **soft variant**: open_atleast (open constraint).

**Keywords**

- **characteristic of a constraint**: automaton, automaton with counters.
- **constraint network structure**: alpha-acyclic constraint network(2).
- **constraint type**: value constraint.
- **filtering**: arc-consistency.
- **modelling**: at least.
Arc input(s) | VARIABLES
--- | ---
Arc generator | SELF→collection(variables)
Arc arity | 1
Arc constraint(s) | variables.var = VALUE
Graph property(ies) | NARC ≥ N

Graph model

Since each arc constraint involves only one vertex (VALUE is fixed), we employ the SELF arc generator in order to produce a graph with a single loop on each vertex.

Parts (A) and (B) of Figure 5.56 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the loops of the final graph are stressed in bold.

![Graph diagram](image)

Figure 5.56: Initial and final graph of the atleast constraint
Automaton

Figure 5.57 depicts the automaton associated with the \texttt{atleast} constraint. To each variable $\text{VAR}_i$ of the collection $\text{VARIABLES}$ corresponds a 0-1 signature variable $S_i$. The following signature constraint links $\text{VAR}_i$ and $S_i$: $\text{VAR}_i = \text{VALUE} \Leftrightarrow S_i$. The automaton counts the number of variables of the $\text{VARIABLES}$ collection that are assigned value $\text{VALUE}$ and finally checks that this number is greater than or equal to $N$.

Figure 5.57: Automaton of the \texttt{atleast} constraint

Figure 5.58: Hypergraph of the reformulation corresponding to the automaton of the \texttt{atleast} constraint
### 5.31 atleast_nvalue

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
</table>

**Origin**

[321]

**Constraint**

atleast_nvalue(NVAL, VARIABLES)

**Synonym**

$k_{diff}$.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVAL</td>
<td>dvar</td>
</tr>
<tr>
<td>VARIABLES</td>
<td>collection(var−dvar)</td>
</tr>
</tbody>
</table>

**Restrictions**

required(VARIABLES, var)

NVAL $\geq$ 0  
NVAL $\leq$ |VARIABLES|  
NVAL $\leq$ range(VARIABLES, var)

**Purpose**

The number of distinct values taken by the variables of the collection VARIABLES is greater than or equal to NVAL.

**Example**

(2, ⟨3, 1, 7, 1, 6⟩)

The atleast_nvalue constraint holds since the collection ⟨3, 1, 7, 1, 6⟩ involves at least 2 distinct values (i.e., in fact 4 distinct values).

**Typical**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVAL $&gt;$ 0</td>
<td></td>
</tr>
<tr>
<td>NVAL $&lt;$</td>
<td></td>
</tr>
<tr>
<td>NVAL $&lt;$ range(VARIABLES, var)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Symmetries**

- NVAL can be **decreased** to any value $\geq$ 0.
- Items of VARIABLES are **permutable**.
- All occurrences of two distinct values of VARIABLES.var can be **swapped**; all occurrences of a value of VARIABLES.var can be **renamed** to any unused value.

**Arg. properties**

Extensible wrt. VARIABLES.

**Remark**

The atleast_nvalue constraint was first introduced by J.-C. Régin under the name $k_{diff}$ in [321]. Later on the atleast_nvalue constraint was introduced together with the **atmost_nvalue** constraint by C. Bessière et al. in a article [58] providing filtering algorithms for the nvalue constraint.

**Algorithm**

[58] provides a sketch of a filtering algorithm enforcing **arc-consistency** for the atleast_nvalue constraint. This algorithm is based on the maximal matching in a bipartite graph.
See also

*comparison swapped*: atmost\_nvalue.

*implied by*: and, nand, nor, nvalue ($\geq NVAL$ replaced by $= NVAL$), or, size\_max\_seq\_alldifferent, size\_max\_starting\_seq\_alldifferent.

*uses in its reformulation*: not\_all\_equal.

**Keywords**

*constraint type*: counting constraint, value partitioning constraint.

*filtering*: bipartite matching, arc-consistency.

*final graph structure*: strongly connected component, equivalence.

*modelling*: number of distinct equivalence classes, number of distinct values.
Arc input(s): VARIABLES
Arc generator: $\text{CLIQUE} \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity: 2
Arc constraint(s): $\text{variables1}.\text{var} = \text{variables2}.\text{var}$
Graph property(ies): \(\text{NSCC} \geq \text{NVAL}\)
Graph class: EQUIVALENCE

Graph model

Parts (A) and (B) of Figure 5.59 respectively show the initial and final graph associated with the Example slot. Since we use the NSCC graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a specific value that is assigned to some variables of the VARIABLES collection. The following values 1, 3, 6 and 7 are used by the variables of the VARIABLES collection.

![Graph Diagram](image)

**(A)**

![SCC Diagram](image)

**(B)**

Figure 5.59: Initial and final graph of the at least n value constraint
## 5.32 atleast_nvector

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from nvector</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>atleast_nvector(NVEC, VECTORS)</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>VECTOR : collection(var-dvar)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>NVEC : dvar</td>
<td>VECTORS : collection(vec - VECTOR)</td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>The number of distinct tuples of values taken by the vectors of the collection VECTORS is greater than or equal to NVEC. Two tuples of values (\langle A_1, A_2, \ldots, A_m \rangle) and (\langle B_1, B_2, \ldots, B_m \rangle) are distinct if and only if there exist an integer (i \in [1, m]) such that (A_i \neq B_i).</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>The (\text{atleast}_n\text{vector}) constraint holds since the collection VECTORS involves at least 2 distinct tuples of values (i.e., in fact the 3 distinct tuples (\langle 5, 6 \rangle, \langle 9, 3 \rangle) and (\langle 9, 4 \rangle)).</td>
<td></td>
</tr>
<tr>
<td>Typical</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symmetries</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arg. properties</td>
<td>Extensible wrt. VECTORS.</td>
<td></td>
</tr>
</tbody>
</table>
Reformulation

By introducing an extra variable \( NV \in [0, |VECTORS|] \), the \( \text{atleast} \_\text{nvector}(NV, VECTORS) \) constraint can be expressed in term of an \( \text{nvector}(NV, VECTORS) \) constraint and of an inequality constraint \( NV \geq NVEC \).

See also

- \textit{comparison swapped}: \textit{atmost} \_\text{nvector}.
- \textit{implied by}: \textit{nvector (} \geq NVEC \text{ replaced by} = NVEC\}), \textit{ordered} \_\text{atleast} \_\text{nvector}.
- \textit{used in graph description}: \textit{lex} \_\text{equal}.

Keywords

- \textit{characteristic of a constraint}: vector.
- \textit{constraint type}: counting constraint, value partitioning constraint.
- \textit{final graph structure}: strongly connected component, equivalence.
- \textit{modelling}: number of distinct equivalence classes.
- \textit{problems}: domination.
Arc input(s): VECTORS
Arc generator: \( CLIQUE \rightarrow \text{collection}(\text{vectors1}, \text{vectors2}) \)
Arc arity: 2
Arc constraint(s): \( \text{lex} \text{.equal}(\text{vectors1.vec}, \text{vectors2.vec}) \)
Graph property(ies): \( \text{NSCC} \geq \text{NVEC} \)
Graph class: \( \text{EQUIVALENCE} \)

Graph model

Parts (A) and (B) of Figure 5.60 respectively show the initial and final graph associated with the Example slot. Since we use the NSCC graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a tuple of values that is assigned to some vectors of the VECTORS collection. The 3 following tuple of values \((5, 6), (9, 3)\) and \((9, 4)\) are used by the vectors of the VECTORS collection.

Figure 5.60: Initial and final graph of the at least \(n\) vector constraint
5.33 atmost

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>CHIP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>atmost(N, VARIABLES, VALUE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td>count</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>N : int</td>
<td>VARIABLES : collection(var−dvar)</td>
<td>VALUE : int</td>
</tr>
<tr>
<td>Restrictions</td>
<td>N \geq 0</td>
<td>required(VARIABLES, var)</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>At most N variables of the VARIABLES collection are assigned value VALUE.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>(1, ⟨4, 2, 4, 5⟩, 2)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The atmost constraint holds since at most 1 value of the collection ⟨4, 2, 4, 5⟩ is equal to value 2.

Typical

N > 0
N < |VARIABLES|
|VARIABLES| > 1
atleast(1, VARIABLES, VALUE)

Symmetries

• Items of VARIABLES are permutable.
• N can be increased.
• An occurrence of a value of VARIABLES.var can be replaced by any other value that is different from VALUE.

Arg. properties

Contractible wrt. VARIABLES.

Systems

occurrenceMax in Choco, count in Gecode, atmost in Gecode, count in JaCoP, at_most in MiniZinc, count in SICStus.

See also

common keyword: among (value constraint).
comparison swapped: atleast.
generalisation: cumulative (variable replaced by task).
implied by: exactly (\leq N replaced by =N).
related: roots.
soft variant: open_atmost (open constraint).
Keywords

characteristic of a constraint: automaton, automaton with counters.
constraint network structure: alpha-acyclic constraint network(2).
constraint type: value constraint.
filtering: arc-consistency.
modelling: at most.
Arc input(s)  VARIABLES
Arc generator  \( \text{SELF} \rightarrow \text{collection}(\text{variables}) \)
Arc arity  1
Arc constraint(s)  \( \text{variables}.\text{var} = \text{VALUE} \)
Graph property(ies)  \( \text{NARC} \leq N \)

Graph model

Since each arc constraint involves only one vertex (VALUE is fixed), we employ the SELF arc generator in order to produce a graph with a single loop on each vertex.

Parts (A) and (B) of Figure 5.61 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the loops of the final graph are stressed in bold.

![Figure 5.61: Initial and final graph of the atmost constraint](image-url)
Automaton

Figure 5.62 depicts the automaton associated with the \textit{atmost} constraint. To each variable \( \text{VAR}_i \) of the collection \( \text{VARIABLES} \) corresponds a 0-1 signature variable \( S_i \). The following signature constraint links \( \text{VAR}_i \) and \( S_i \): \( \text{VAR}_i = \text{VALUE} \leftrightarrow S_i \). The automaton counts the number of variables of the \( \text{VARIABLES} \) collection that are assigned value \( \text{VALUE} \) and finally checks that this number is less than or equal to \( N \).

<table>
<thead>
<tr>
<th>State</th>
<th>Transition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S )</td>
<td>{C=0}</td>
<td>( \text{VAR}_i = \text{VALUE} ), {C=C+1}</td>
</tr>
<tr>
<td>( N \geq C )</td>
<td>{C=C+1}</td>
<td>( \text{VAR}_i \neq \text{VALUE} ), ( i )</td>
</tr>
</tbody>
</table>

Figure 5.62: Automaton of the \textit{atmost} constraint

<table>
<thead>
<tr>
<th>State</th>
<th>Transition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Q_0 )</td>
<td>{C=0}</td>
<td>( S_1 ), ( C_0 = 0 )</td>
</tr>
<tr>
<td>( Q_1 )</td>
<td></td>
<td>( S_2 ), ( C_1 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( Q_n )</td>
</tr>
<tr>
<td>( Q_n )</td>
<td></td>
<td>( S_n ), ( C_n \leq N )</td>
</tr>
</tbody>
</table>

Figure 5.63: Hypergraph of the reformulation corresponding to the automaton of the \textit{atmost} constraint
5.34 atmost1

**DESCRIPTION**

**Origin**

[344]

**Constraint**

atmost1(SETS)

**Synonym**

pair_atmost1.

**Argument**

SETS : collection(s–svar, c–int)

**Restrictions**

required(SETS, [s, c])

SETS.c ≥ 1

**Purpose**

Given a collection of set variables $s_1, s_2, \ldots, s_n$ and their respective cardinality $c_1, c_2, \ldots, c_n$, the **atmost1** constraint enforces the following two conditions:

- $\forall i \in [1, n] : |s_i| = c_i$,
- $\forall i, j \in [1, n] \ (i < j) : |s_i \cap s_j| \leq 1$.

**Example**

$$\begin{pmatrix}
  s = \{5, 8\} & c = 2, \\
  s = \{5\} & c = 1, \\
  s = \{5, 6, 7\} & c = 3, \\
  s = \{1, 4\} & c = 2
\end{pmatrix}$$

The **atmost1** constraint holds since:

- $|\{5, 8\}| = 2, |\{5\}| = 1, |\{5, 6, 7\}| = 3, |\{1, 4\}| = 2$.
- $|\{5, 8\} \cap \{5\}| \leq 1, |\{5, 8\} \cap \{5, 6, 7\}| \leq 1, |\{5, 8\} \cap \{1, 4\}| \leq 1, |\{5\} \cap \{5, 6, 7\}| \leq 1, |\{5\} \cap \{1, 4\}| \leq 1, |\{5, 6, 7\} \cap \{1, 4\}| \leq 1$.

**Typical**

$|SETS| > 1$

**Symmetries**

- All occurrences of SETS are **permutable**.
- All occurrences of two distinct values of SETS.a can be **swapped**; all occurrences of a value of SETS.a can be **renamed** to any unused value.

**Arg. properties**

**Contractible** wrt. SETS.

**Remark**

When we have only two set variables the **atmost1** constraint was called **pair_atmost1** in [403].
Algorithm

C. Bessière et al. have shown in [64] that it is NP-hard to enforce bound consistency for the atmost1 constraint. Consequently, following the first filtering algorithm from A. Sadler and C. Gervet [344], W.-J. van Hoeve and A. Sabharwal have proposed an algorithm that enforces bound-consistency when the atmost1 constraint involves only two sets variables [403].

Systems

at_most1 in MiniZinc.

Keywords

costRAINT arguments: constraint involving set variables.
costRAINT type: predefined constraint.
costRAINT filtering: bound-consistency.
### 5.35 **atmost_nvalue**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>[58]</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>$\text{atmost_nvalue}(NVAL, \text{VARIABLES})$</td>
<td></td>
</tr>
<tr>
<td><strong>Synonyms</strong></td>
<td>$\text{soft_alldiff_max_var}$, $\text{soft_alldifferent_max_var}$, $\text{soft_alldistinct_max_var}$</td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>$NVAL : \text{dvar}$, $\text{VARIABLES} : \text{collection(\text{var} - \text{dvar})}$</td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>$NVAL \geq \min(1,</td>
<td>\text{VARIABLES}</td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>The number of distinct values taken by the variables of the collection $\text{VARIABLES}$ is less than or equal to $NVAL$.</td>
<td></td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>$(4, (3, 1, 3, 1, 6))$</td>
<td></td>
</tr>
</tbody>
</table>

The $\text{atmost\_nvalue}$ constraint holds since the collection $(3, 1, 3, 1, 6)$ involves at most 4 distinct values (i.e., in fact 3 distinct values).

| Typical | $NVAL > 1$
| $NVAL < |\text{VARIABLES}|$
| $|\text{VARIABLES}| > 1$ |

<table>
<thead>
<tr>
<th>Symmetries</th>
</tr>
</thead>
<tbody>
<tr>
<td>$NVAL$ can be increased.</td>
</tr>
<tr>
<td>Items of $\text{VARIABLES}$ are permutable.</td>
</tr>
<tr>
<td>All occurrences of two distinct values of $\text{VARIABLES}.\text{var}$ can be swapped; all occurrences of a value of $\text{VARIABLES}.\text{var}$ can be renamed to any unused value.</td>
</tr>
<tr>
<td>An occurrence of a value of $\text{VARIABLES}.\text{var}$ can be replaced by any value of $\text{VARIABLES}.\text{var}$.</td>
</tr>
</tbody>
</table>

| Arg. properties | Contractible wrt. $\text{VARIABLES}$. |

<table>
<thead>
<tr>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>This constraint was introduced together with the $\text{atleast_nvalue}$ constraint by C. Bessière et al. in a article [58] providing filtering algorithms for the $\text{nvalue}$ constraint.</td>
</tr>
<tr>
<td>It was shown in [65] that, finding out whether a $\text{atmost_nvalue}$ constraint has a solution or not is NP-hard. This was achieved by reduction from $\text{3-SAT}$.</td>
</tr>
</tbody>
</table>

| Algorithm | [26] provides an algorithm that achieves bound consistency. [38] provides two filtering algorithms, while [58] provides a greedy algorithm and a graph invariant for evaluating the minimum number of distinct values. [58] also gives a linear relaxation for approximating the minimum number of distinct values. |
atMostNValue in Choco.

See also

comparison swapped: atleast_nvalue.

implied by: nvalue \( \leq NVAL \) replaced by \( NVAL \).

related: soft_all_equal_max_var, soft_all_equal_min ctr,
soft_all_equal_min_var, soft_alldifferent_ctr, soft_alldifferent_var.

Keywords

complexity: 3-SAT.

collection type: counting constraint, value partitioning constraint.

filtering: bound-consistency.

final graph structure: strongly connected component, equivalence.

modelling: number of distinct equivalence classes, number of distinct values.
Parts (A) and (B) of Figure 5.64 respectively show the initial and final graph associated with the Example slot. Since we use the NSCC graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a specific value that is assigned to some variables of the VARIABLES collection. The following values 1, 3 and 6 are used by the variables of the VARIABLES collection.

Figure 5.64: Initial and final graph of the atmost_nvalue constraint
## 5.36 atmost_nvector

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Derived from <code>nvector</code></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td><code>atmost_nvector(NVEC, VECTORS)</code></td>
<td></td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td><code>VECTOR : collection(var−dvar)</code></td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td><code>NVEC : dvar</code></td>
<td><code>VECTORS : collection(vec − VECTOR)</code></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>`</td>
<td>VECTOR</td>
</tr>
<tr>
<td></td>
<td><code>required(VECTORS, vec)</code></td>
<td><code>same_size(VECTORS, vec)</code></td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>The number of distinct tuples of values taken by the vectors of the collection <code>VECTORS</code> is less than or equal to <code>NVEC</code>. Two tuples of values (\langle A_1, A_2, \ldots, A_m \rangle) and (\langle B_1, B_2, \ldots, B_m \rangle) are distinct if and only if there exist an integer (i \in [1, m]) such that (A_i \neq B_i).</td>
<td></td>
</tr>
</tbody>
</table>
| **Example** | \[
\begin{pmatrix}
  \text{vec} \to 5,6, \\
  \text{vec} \to 5,6, \\
  3, \text{vec} \to 9,3, \\
  \text{vec} \to 5,6, \\
  \text{vec} \to 9,3
\end{pmatrix}
\] |       |

The `atmost_nvector` constraint holds since the collection `VECTORS` involves at most 3 distinct tuples of values (i.e., in fact the 2 distinct tuples \(\langle 5, 6 \rangle\) and \(\langle 9, 3 \rangle\)).

| **Typical** | `|VECTOR| > 1` | `NVEC > 1` | `NVEC < |VECTORS|` | `|VECTORS| > 1` |
|-------------|-------|-----------|----------------|----------|

| **Symmetries** | • `NVEC` can be increased. |       |
|               | • Items of `VECTORS` are permutable. |       |
|               | • Items of `VECTORS.vec` are permutable (same permutation used). |       |
|               | • All occurrences of two distinct tuples of values of `VECTORS.vec` can be swapped; all occurrences of a tuple of values of `VECTORS.vec` can be renamed to any unused tuple of values. |       |

| **Arg. properties** | `Contractible wrt. VECTORS.` |       |
Reformulation

By introducing an extra variable $NV \in [0, |VECTORS|]$, the atmost_nvvector($NV, VECTORS$) constraint can be expressed in term of an nvector($NV, VECTORS$) constraint and of an inequality constraint $NV \leq NVEC$.

See also

comparison swapped: atleast_nvvector.

implied by: nvector ($\leq NVEC$ replaced by $= NVEC$), ordered_atmost_nvvector.

used in graph description: lex_equal.

Keywords

characteristic of a constraint: vector.

constraint type: counting constraint, value partitioning constraint.

final graph structure: strongly connected component, equivalence.

modelling: number of distinct equivalence classes.

problems: domination.
Part (A) and (B) of Figure 5.65 respectively show the initial and final graph associated with the Example slot. Since we use the NSCC graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a tuple of values that is assigned to some vectors of the VECTORS collection. The 2 following tuple of values ⟨5, 6⟩ and ⟨9, 3⟩ are used by the vectors of the VECTORS collection.

Figure 5.65: Initial and final graph of the atmost_nvector constraint
### 5.37 balance

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>N. Beldiceanu</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>balance(BALANCE, VARIABLES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>BALANCE : dvar</td>
<td>VARIABLES : collection(var–dvar)</td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>BALANCE ≥ 0</td>
<td>BALANCE ≤ max(0,</td>
<td>VARIABLES</td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>BALANCE is equal to the difference between the number of occurrence of the value that occurs the most and the value that occurs the least within the collection of variables VARIABLES.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>(2, ⟨3, 1, 7, 1, 1⟩)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In this example, values 1, 3 and 7 are respectively used 3, 1 and 1 times. The balance constraint holds since its first argument BALANCE is assigned to the difference between the maximum and minimum number of the previous occurrences (i.e., 3 − 1). Figure 5.66 shows the solution associated with the example.

<table>
<thead>
<tr>
<th>Typical</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symmetries</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>● Items of VARIABLES are permutable.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>● All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arg. properties</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional dependency: BALANCE determined by VARIABLES.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usage</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>An application of the balance constraint is to enforce a balanced assignment of values, no matter how many distinct values will be used. In this case one will push down the maximum value of the first argument of the balance constraint.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Remark</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>If we do not want to use an automaton with an array of counters a possible reformulation of the balance constraint can be achieved in the following way. We use a sort constraint in order to reorder the variables of the collection VARIABLES and compute the difference between the longest and the smallest sequences of consecutive values.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>See also</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>generalisation: balance_interval (variable replaced by variable/constant), balance_modulo (variable replaced by variable mod constant), balance_partition (variable replaced by variable ∈ partition).</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
related: balance_cycle (balanced assignment versus graph partitioning with balanced cycles), balance_path (balanced assignment versus graph partitioning with balanced paths), balance_tree (balanced assignment versus graph partitioning with balanced trees), nvalue (no restriction on how balanced an assignment is), tree_range (balanced assignment versus balanced tree).

Keywords

application area: assignment.
characteristic of a constraint: automaton, automaton with array of counters.
constraint arguments: pure functional dependency.
constraint type: value constraint.
final graph structure: equivalence.
modelling: balanced assignment, functional dependency.
Figure 5.66: Illustration of the example: five variables respectively fixed to values 3, 1, 7, 1 and 1, and the corresponding value of BALANCE = 2
Arc input(s) | VARIABLES
---|---
Arc generator | $CLIQUE \rightarrow \text{collection}(\text{variables1}, \text{variables2})$
Arc arity | 2
Arc constraint(s) | variables1.var = variables2.var
Graph property(ies) | RANGE\_NSCC = BALANCE
Graph class | EQUIVALENCE

Graph model

The graph property RANGE\_NSCC constraints the difference between the sizes of the largest and smallest strongly connected components.

Parts (A) and (B) of Figure 5.67 respectively show the initial and final graph associated with the Example slot. Since we use the RANGE\_NSCC graph property, we show the largest and smallest strongly connected components of the final graph.

![Graph](image)

Figure 5.67: Initial and final graph of the balance constraint
Automaton Figure 5.68 depicts the automaton associated with the balance constraint. To each item of the collection VARIABLES corresponds a signature variable $S_i$ that is equal to 1.

$$\{C[\_]=0\}$$

$$S:\ egin{array}{l}
\text{minimum\_except\_0}(N1,C) \\
\text{maximum}(N2,C) \\
\text{BALANCE}=N2-N1
\end{array}$$

$$\{C[VAR_i]=C[VAR_i]+1\}$$

Figure 5.68: Automaton of the balance constraint
### 5.38 balance_cycle

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>balance_cycle(BALANCE, NODES)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>BALANCE : dvar</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES : collection(index-int, succ-dvar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>BALANCE \geq 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BALANCE \leq \max(0,</td>
<td>NODES</td>
</tr>
<tr>
<td></td>
<td>required(NODES, [index, succ])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.index \geq 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.index \leq</td>
<td>NODES</td>
</tr>
<tr>
<td></td>
<td>distinct(NODES, index)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.succ \geq 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.succ \leq</td>
<td>NODES</td>
</tr>
</tbody>
</table>

**Purpose**

Consider a digraph $G$ described by the `NODES` collection. Partition $G$ into a set of vertex disjoint circuits in such a way that each vertex of $G$ belongs to one single circuit. BALANCE is equal to the difference between the number of vertices of the largest circuit and the number of vertices of the smallest circuit.

**Example**

\[
\begin{pmatrix}
\text{index} - 1 & \text{succ} - 2, \\
\text{index} - 2 & \text{succ} - 1, \\
1, ( & \text{index} - 3 \text{ succ} - 5, \\
\text{index} - 4 & \text{succ} - 3, \\
\text{index} - 5 & \text{succ} - 4
\end{pmatrix}
\]

In this example we have the following two circuits: $1 \rightarrow 2 \rightarrow 1$ and $3 \rightarrow 5 \rightarrow 4 \rightarrow 3$. Since $\text{BALANCE} = 1$ is the difference between the number of vertices of the largest circuit (i.e., 3) and the number of vertices of the smallest circuit (i.e., 2) the balance_cycle constraint holds.

**Typical**

$|\text{NODES}| > 2$

**Symmetry**

Items of `NODES` are permutable.

**Arg. properties**

Functional dependency: BALANCE determined by `NODES`.

**See also**

related: balance (equivalence classes correspond to vertices in same cycle rather than variables assigned to the same value), cycle (do not care how many cycles but how balanced the cycles are).
Keywords

- **combinatorial object:** permutation.
- **constraint type:** graph constraint, graph partitioning constraint.
- **filtering:** DFS-bottleneck.
- **final graph structure:** circuit, connected component, strongly connected component, one_suc.
- **modelling:** cycle, functional dependency.
Arc input(s)  NODES
Arc generator  \textit{CLIQUE} \rightarrow \text{collection}(\text{nodes}_1, \text{nodes}_2)
Arc arity 2
Arc constraint(s)  \text{nodes}_1.\text{succ} = \text{nodes}_2.\text{index}
Graph property(ies)  
  \bullet \text{NTREE} = 0 
  \bullet \text{RANGE}_\text{NCC} = \text{BALANCE}
Graph class \text{ONE}_\text{SUCC}

Graph model

From the restrictions and from the arc constraint, we deduce that we have a bijection from the successor variables to the values of interval $[1, \text{|NODES|}]$. With no explicit restrictions it would have been impossible to derive this property.

In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices. This is why the \textit{balance}_\text{cycle} constraint considers objects that have two attributes:

  \bullet One fixed attribute \text{index} that is the identifier of the vertex,
  \bullet One variable attribute \text{succ} that is the successor of the vertex.

The graph property $\text{NTREE} = 0$ is used in order to avoid having vertices that both do not belong to a \textit{circuit} and have at least one successor located on a \textit{circuit}. This concretely means that all vertices of the final graph should belong to a \textit{circuit}.

Parts (A) and (B) of Figure 5.69 respectively show the initial and final graph associated with the \textbf{Example} slot. Since we use the \textit{RANGE}_\text{NCC} graph property, we show the connected components of the final graph. The constraint holds since all the vertices belong to a \textit{circuit} (i.e., $\text{NTREE} = 0$) and since $\text{BALANCE} = \text{RANGE}_\text{NCC} = 1$.

![Graphs](image)

Figure 5.69: Initial and final graph of the \textit{balance}_\text{cycle} constraint
5.39 balance_interval

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from balance.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>balance_interval(BALANCE, VARIABLES, SIZE_INTERVAL)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>BALANCE : dvar</td>
<td>VARIABLES : collection(var−dvar)</td>
<td>SIZE_INTERVAL : int</td>
</tr>
<tr>
<td>Restrictions</td>
<td>BALANCE ≥ 0</td>
<td>BALANCE ≤ max(0,</td>
<td>VARIABLES</td>
</tr>
<tr>
<td>Purpose</td>
<td>Consider the largest set $S_1$ (respectively the smallest set $S_2$) of variables of the collection VARIABLES that take their value in a same interval $[\text{SIZE_INTERVAL}·k, \text{SIZE_INTERVAL}·k + \text{SIZE_INTERVAL} − 1]$, where $k$ is an integer. BALANCE is equal to the difference between the cardinality of $S_2$ and the cardinality of $S_1$.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>(3, (6, 4, 3, 3, 4), 3)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the example, the third argument $\text{SIZE_INTERVAL} = 3$ defines the following family of intervals $[3 · k, 3 · k + 2]$, where $k$ is an integer. Values 6, 4, 3, 3 and 4 are respectively located within intervals [6, 8], [3, 5], [3, 5], [3, 5] and [3, 5]. Therefore intervals [6, 8] and [3, 5] are respectively used 1 and 4 times. The balance_interval constraint holds since its first argument BALANCE is assigned to the difference between the maximum and minimum number of the previous occurrences (i.e., $4 − 1$).

Typical

- $|\text{VARIABLES}| > 2$
- $\text{SIZE_INTERVAL} > 1$
- $\text{SIZE_INTERVAL} < \text{range}(\text{VARIABLES}.\text{var})$

Symmetries
- Items of VARIABLES are permutable.
- An occurrence of a value of VARIABLES.var that belongs to the $k$-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.

Arg. properties
- Functional dependency: BALANCE determined by VARIABLES and SIZE_INTERVAL.

Usage
An application of the balance_interval constraint is to enforce a balanced assignment of interval of values, no matter how many distinct interval of values will be used. In this case one will push down the maximum value of the first argument of the balance_interval constraint.

See also
specialisation: balance(variable/constant replaced by variable).
Keywords

- **application area**: assignment.
- **characteristic of a constraint**: automaton, automaton with array of counters.
- **constraint arguments**: pure functional dependency.
- **constraint type**: value constraint.
- **final graph structure**: equivalence.
- **modelling**: interval, balanced assignment, functional dependency.
Graph model

The graph property \texttt{RANGE\_NSCC} constrains the difference between the sizes of the largest and smallest strongly connected components.

Parts (A) and (B) of Figure 5.70 respectively show the initial and final graph associated with the \texttt{Example} slot. Since we use the \texttt{RANGE\_NSCC} graph property, we show the largest and smallest strongly connected components of the final graph.

![Figure 5.70: Initial and final graph of the balance interval constraint](image-url)
Automaton

Figure 5.71 depicts the automaton associated with the balance_interval constraint. To each item of the collection VARIABLES corresponds a signature variable $S_i$ that is equal to 1.

$$\{C[\_]=0\}$$

$$S:$$
- $\text{minimum\_except\_0}(N1,C)$
- $\text{maximum}(N2,C)$
- $\text{BALANCE}=N2-N1$

$$\{C[\text{VAR}_1/\text{SIZE\_INTERVAL}]=C[\text{VAR}_1/\text{SIZE\_INTERVAL}]+1\}$$

Figure 5.71: Automaton of the balance_interval constraint
5.40 balance_modulo

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from balance.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>balance_modulo(BALANCE, VARIABLES, M)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Arguments   | BALANCE : dvar  
VARiABLES : collection(var−dvar)  
M : int |       |           |
| Restrictions| BALANCE ≥ 0  
BALANCE ≤ max(0, |VARiABLES| − 2)  
required(VARiABLES, var)  
M > 0 |       |           |
| Purpose     | Consider the largest set $S_1$ (respectively the smallest set $S_2$) of variables of the collection VARIABLES that have the same remainder when divided by M. BALANCE is equal to the difference between the cardinality of $S_2$ and the cardinality of $S_1$. |       |           |
| Example     | (2, (6, 1, 7, 1, 5), 3) |       |           |
| Typical     | | | |
| Symmetries  | • Items of VARIABLES are permutable.  
• An occurrence of a value $u$ of VARIABLES.var can be replaced by any other value $v$ such that $v$ is congruent to $u$ modulo M. |       |           |
| Arg. properties | Functional dependency: BALANCE determined by VARIABLES and M. |       |           |
| Usage       | An application of the balance_modulo constraint is to enforce a balanced assignment of values, no matter how many distinct equivalence classes will be used. In this case one will push down the maximum value of the first argument of the balance_modulo constraint. |       |           |
| See also    | specialisation: balance(variable mod constant replaced by variable). |       |           |
Keywords

- **application area**: assignment.
- **characteristic of a constraint**: modulo, automaton, automaton with array of counters.
- **constraint arguments**: pure functional dependency.
- **constraint type**: value constraint.
- **final graph structure**: equivalence.
- **modelling**: balanced assignment, functional dependency.
The graph property RANGE_NS CC constraints the difference between the sizes of the largest and smallest strongly connected components.

Parts (A) and (B) of Figure 5.72 respectively show the initial and final graph associated with the Example slot. Since we use the RANGE_NS CC graph property, we show the largest and smallest strongly connected components of the final graph.

Figure 5.72: Initial and final graph of the balance_modulo constraint
Automaton

Figure 5.73 depicts the automaton associated with the balance\_modulo constraint. To each item of the collection VARIABLES corresponds a signature variable $S_i$ that is equal to 1.

\begin{itemize}
  \item $C[\_]=0$
  \item $S: \text{minimum\_except\_0}(N1,C)$
  \item $\text{maximum}(N2,C)$
  \item $\text{BALANCE}=N2-N1$
  \item $1, \quad \{C[\text{VAR\_mod\_M}]=C[\text{VAR\_mod\_M}]+1\}$
\end{itemize}

Figure 5.73: Automaton of the balance\_modulo constraint
5.41 balance_partition

### Description

**Origin**
Derived from balance.

**Constraint**
balance_partition(BALANCE, VARIABLES, PARTITIONS)

**Type**
VALUES : collection(val → int)

**Arguments**
BALANCE : dvar
VARIABLES : collection(var → dvar)
PARTITIONS : collection(p → VALUES)

**Restrictions**
- |VALUES| ≥ 1
- required(VARIABLES, val)
- distinct(VARIABLES, val)
- BALANCE ≥ 0
- BALANCE ≤ max(0, |VARIABLES| − 2)
- required(VARIABLES, var)
- required(PARTITIONS, p)
- |PARTITIONS| ≥ 2

**Purpose**
Consider the largest set $S_1$ (respectively the smallest set $S_2$) of variables of the collection VARIABLES that take their value in the same partition of the collection PARTITIONS. BALANCE is equal to the difference between the cardinality of $S_2$ and the cardinality of $S_1$.

**Example**

\[
\begin{align*}
&1, \langle 6, 2, 6, 4, 4 \rangle, \\
P &\langle 1, 3 \rangle, \\
P &\langle 4 \rangle, \\
P &\langle 2, 6 \rangle
\end{align*}
\]

In this example values 6, 2, 6, 4, 4 are respectively associated with the partitions $p \langle 2, 6 \rangle$ and $p \langle 4 \rangle$. Partitions $p \langle 4 \rangle$ and $p \langle 2, 6 \rangle$ are respectively used 2 and 3 times. The balance_partition constraint holds since its first argument BALANCE is assigned to the difference between the maximum and minimum number of the previous occurrences (i.e., $3 - 2$). Note that we do not consider those partitions that are not used at all.

**Typical**

- |VARIABLES| > 2
- |VARIABLES| > |PARTITIONS|

**Symmetries**

- Items of VARIABLES are permutable.
- Items of PARTITIONS are permutable.
- Items of PARTITIONS.p are permutable.
- An occurrence of a value of VARIABLES.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.
Arg. properties

Functional dependency: BALANCE determined by VARIABLES and PARTITIONS.

Usage

An application of the balance.partition is to enforce a balanced assignment of values, no matter how many distinct partitions will be used. In this case one will push down the maximum value of the first argument of the balance.partition constraint.

See also

specialisation: balance(variable ∈ partition replaced by variable).

used in graph description: in_same.partition.

Keywords

application area: assignment.

characteristic of a constraint: partition.

constraint arguments: pure functional dependency.

constraint type: value constraint.

final graph structure: equivalence.

modelling: balanced assignment, functional dependency.
Arc input(s) | VARIABLES
---|---
Arc generator | \( \text{CLIQUE} \rightarrow \text{collection}(\text{variables1,variables2}) \)
Arc arity | 2
Arc constraint(s) | \( \text{in\_same\_partition}(\text{variables1.var,variables2.var,PARTITIONS}) \)
Graph property(ies) | RANGE_NSCC = BALANCE
Graph class | EQUIVALENCE

Graph model

The graph property \text{RANGE_NSCC} constrains the difference between the sizes of the largest and smallest strongly connected components.

Parts (A) and (B) of Figure 5.74 respectively show the initial and final graph associated with the Example slot. Since we use the \text{RANGE_NSCC} graph property, we show the largest and smallest strongly connected components of the final graph.

![Graph Model](image)

**Figure 5.74:** Initial and final graph of the balance\_partition constraint
5.42 balance_path

**DESCRIPTION**  
 Origin: derived from balance and path

**LINKS**

**GRAPH**

**Constraint**  
`balance_path(BALANCE, NODES)`

**Arguments**

`BALANCE` : dvar

`NODES` : collection(index=int, succ=dvar)

**Restrictions**

`BALANCE ≥ 0`

`BALANCE ≤ max(0, |NODES| − 2)`

`required(NODES, [index, succ])`

`NODES.index ≥ 1`

`NODES.index ≤ |NODES|`

`distinct(NODES, index)`

`NODES.succ ≥ 1`

`NODES.succ ≤ |NODES|`

**Purpose**

Consider a digraph `G` described by the `NODES` collection. Partition `G` into a set of vertex disjoint paths in such a way that each vertex of `G` belongs to one single path. `BALANCE` is equal to the difference between the number of vertices of the largest path and the number of vertices of the smallest path.

**Example**

\[
\begin{pmatrix}
\text{index} - 1 & \text{succ} - 1, \\
\text{index} - 2 & \text{succ} - 3, \\
\text{index} - 3 & \text{succ} - 5, \\
\text{index} - 4 & \text{succ} - 4, \\
\text{index} - 5 & \text{succ} - 1, \\
\text{index} - 6 & \text{succ} - 6, \\
\text{index} - 7 & \text{succ} - 7, \\
\text{index} - 8 & \text{succ} - 6
\end{pmatrix}
\]

In this example we have the following four paths: `2 → 3 → 5 → 1`, `8 → 6`, `4`, and `7`. Since `BALANCE = 3` is the difference between the number of vertices of the largest path (i.e., 4) and the number of vertices of the smallest path (i.e., 1) the `balance_path` constraint holds.

**Typical**

`|NODES| > 2`

**Symmetry**

Items of `NODES` are permutable.

**Arg. properties**

Functional dependency: `BALANCE` determined by `NODES`.

**See also**

related: balance (equivalence classes correspond to vertices in same path rather than variables assigned to the same value), path (do not care how many paths but how balanced the paths are).
Keywords

- **combinatorial object:** path.
- **constraint type:** graph constraint, graph partitioning constraint.
- **filtering:** DFS-bottleneck.
- **final graph structure:** connected component, tree, one_suce.
- **modelling:** functional dependency.
Arc input(s)  NODES
Arc generator  \( CLIQUE \rightarrow \text{collection}(\text{nodes1}, \text{nodes2}) \)
Arc arity  2
Arc constraint(s)  \( \text{nodes1}.\text{succ} = \text{nodes2}.\text{index} \)
Graph property(ies)  
- \( \text{MAX}_\text{NSCC} \leq 1 \)
- \( \text{MAX}_\text{ID} \leq 1 \)
- \( \text{RANGE}_\text{NCC} = \text{BALANCE} \)

Graph class  \( \text{ONE}_\text{SUCC} \)

Graph model  
In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices. This is why the \text{balance_path} constraint considers objects that have two attributes:

- One fixed attribute \text{index} that is the identifier of the vertex,
- One variable attribute \text{succ} that is the successor of the vertex.

We use the graph property \( \text{MAX}_\text{NSCC} \leq 1 \) in order to specify the fact that the size of the largest strongly connected component should not exceed one. In fact each root of a tree is a strongly connected component with one single vertex. The graph property \( \text{MAX}_\text{ID} \leq 1 \) constrains the maximum in-degree of the final graph to not exceed 1. \( \text{MAX}_\text{ID} \) does not consider loops: This is why we do not have any problem with the final node of each path.

Parts (A) and (B) of Figure 5.75 respectively show the initial and final graphs associated with the \text{Example} slot. Since we use the \text{RANGE}_\text{NCC} graph property, we show the connected components of the final graph. The constraint holds since all the vertices belong to a path and since \( \text{BALANCE} = \text{RANGE}_\text{NCC} = 3 \).

![Figure 5.75: Initial and final graph of the balance_path constraint](image-url)
5.43 balance_tree

**DESCRIPTION**

origin derived from `balance` and `tree`.

**Constraint**

`balance_tree(BALANCE, NODES)`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BALANCE</td>
<td><code>dvar</code></td>
</tr>
<tr>
<td>NODES</td>
<td><code>collection(index-int, succ-dvar)</code></td>
</tr>
</tbody>
</table>

**Restrictions**

- `BALANCE ≥ 0`
- `BALANCE ≤ max(0, |NODES| − 2)`
- `required(NODES, [index, succ])`
- `NODES.index ≥ 1`
- `NODES.index ≤ |NODES|`
- `distinct(NODES, index)`
- `NODES.succ ≥ 1`
- `NODES.succ ≤ |NODES|`

**Purpose**

Consider a digraph `G` described by the `NODES` collection. Partition `G` into a set of vertex disjoint trees in such a way that each vertex of `G` belongs to one single `tree`. `BALANCE` is equal to the difference between the number of vertices of the largest tree and the number of vertices of the smallest tree.

**Example**

In this example we have two trees involving respectively the set of vertices `{1, 2, 3, 5, 6, 8}` and the set `{4, 7}`. They are depicted by Figure 5.76. Since `BALANCE = 6 − 2 = 4` is the difference between the number of vertices of the largest tree (i.e., 6) and the number of vertices of the smallest tree (i.e., 2) the `balance_tree` constraint holds.

![Figure 5.76](image)

**Figure 5.76:** The two trees associated with the example respectively containing 6 and 2 vertices, therefore `BALANCE = 6 − 2 = 4`
Typical

$$|\text{NODES}| > 2$$

Symmetry

Items of NODES are permutable.

Arg. properties

Functional dependency: BALANCE determined by NODES.

See also

Related: balance (equivalence classes correspond to vertices in same tree rather than variables assigned to the same value), tree (do not care how many trees but how balanced the trees are).

Keywords

Constraint type: graph constraint, graph partitioning constraint.
Final graph structure: connected component, tree, one_suc.
Modelling: functional dependency.
Arc input(s)  
NODES

Arc generator  
\( CLIQUE \mapsto \text{collection}(\text{nodes}_1, \text{nodes}_2) \)

Arc arity  
2

Arc constraint(s)  
\( \text{nodes}_1.\text{succ} = \text{nodes}_2.\text{index} \)

Graph property(ies)  
- \( \text{MAX\_NSCC} \leq 1 \)
- \( \text{RANGE\_NCC} = \text{BALANCE} \)

Graph model  
In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices. This is why the \( \text{balance\_tree} \) constraint considers objects that have two attributes:

- One fixed attribute \( \text{index} \) that is the identifier of the vertex,
- One variable attribute \( \text{succ} \) that is the successor of the vertex.

We use the graph property \( \text{MAX\_NSCC} \leq 1 \) in order to specify the fact that the size of the largest strongly connected component should not exceed one. In fact each root of a tree is a strongly connected component with one single vertex.

Parts (A) and (B) of Figure 5.77 respectively show the initial and final graphs associated with the Example slot. Since we use the \( \text{RANGE\_NCC} \) graph property, we show the connected components of the final graph. The constraint holds since all the vertices belong to a tree and since \( \text{BALANCE} = \text{RANGE\_NCC} 6 - 2 = 4 \).

![Graph (A)](image_a.png)

![Graph (B)](image_b.png)

Figure 5.77: Initial and final graph of the balance\_tree constraint
5.44  **between_min_max**

**DESCRIPTION**

Used for defining `cumulative.convex`.

**LINKS**

**GRAPH**

**AUTOMATON**

**Origin**

```
between_min_max(VAR, VARIABLES)
```

**Arguments**

```
VAR : dvar
VARIABLES : collection(var−dvar)
```

**Restrictions**

```
required(VARIABLES.var)
|VARIABLES| > 0
```

**Purpose**

VAR is greater than or equal to at least one variable of the collection VARIABLES and less than or equal to at least one variable of the collection VARIABLES.

**Example**

```
(3, ⟨1, 1, 4, 8⟩)
```

The `between_min_max` constraint holds since its first argument 3 is greater than or equal to the minimum value of the values of the collection ⟨1, 1, 4, 8⟩ and less than or equal to the maximum value of ⟨1, 1, 4, 8⟩.

**Typical**

```
|VARIABLES| > 1
range(VARIABLES.var) > 1
```

**Symmetries**

- Items of VARIABLES are **permutable**.
- VAR can be **set** to any value of VARIABLES.var.

**Arg. properties**

**Extensible** wrt. VARIABLES.

**Reformulation**

By introducing two extra variables MIN and MAX, the `between_min_max(VAR, VARIABLES)` constraint can be expressed in term of the following conjunction of constraints:

```
minimum(MIN, VARIABLES),
maximum(MAX, VARIABLES),
VAR ≥ MIN,
VAR ≤ MAX.
```

**Used in**

`cumulative.convex`.

**See also**

**implied by:** in.

**Keywords**

**characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

**constraint network structure:** centered cyclic(1) constraint network(1).
### Derived Collection

\[
\text{col}(\text{ITEM}_{-\text{collection}}(\text{var}_{-\text{dvar}}), [\text{item}_{-\text{VAR}}])
\]

<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>ITEM VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>(PRODUCT \rightarrow \text{collection}([\text{item}, \text{variables}]))</td>
</tr>
<tr>
<td>Arc arity</td>
<td>2</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>(\text{item-var} \geq \text{variables-var})</td>
</tr>
<tr>
<td>Graph property(ies)</td>
<td>(\text{NARC} \geq 1)</td>
</tr>
<tr>
<td>Graph class</td>
<td>• ACYCLIC</td>
</tr>
<tr>
<td></td>
<td>• BIPARTITE</td>
</tr>
<tr>
<td></td>
<td>• NO LOOP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>ITEM VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>(PRODUCT \rightarrow \text{collection}([\text{item}, \text{variables}]))</td>
</tr>
<tr>
<td>Arc arity</td>
<td>2</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>(\text{item-var} \leq \text{variables-var})</td>
</tr>
<tr>
<td>Graph property(ies)</td>
<td>(\text{NARC} \geq 1)</td>
</tr>
<tr>
<td>Graph class</td>
<td>• ACYCLIC</td>
</tr>
<tr>
<td></td>
<td>• BIPARTITE</td>
</tr>
<tr>
<td></td>
<td>• NO LOOP</td>
</tr>
</tbody>
</table>

### Graph model

Parts (A) and (B) of Figure 5.78 respectively show the initial and final graph associated with the second graph constraint of the Example slot. Since we use the \(\text{NARC}\) graph property, the two arcs of the final graph are stressed in bold. The constraint holds since 3 is greater than 1 and since 3 is less than 8.

![Graph Model](image)

Figure 5.78: Initial and final graph of the between \text{min} \_\text{max} constraint
Automaton Figure 5.79 depicts the automaton associated with the \texttt{between\_min\_max} constraint. To each pair \((\text{VAR}, \text{VAR}_i)\), where \text{VAR}_i is a variable of the collection \texttt{VARIABLES} corresponds a signature variable \(S_i\). The following signature constraint links \text{VAR}, \text{VAR}_i, and \(S_i\): 

\[(\text{VAR} < \text{VAR}_i \Leftrightarrow S_i = 0) \land (\text{VAR} = \text{VAR}_i \Leftrightarrow S_i = 1) \land (\text{VAR} > \text{VAR}_i \Leftrightarrow S_i = 2).\]

Figure 5.79: Automaton of the \texttt{between\_min\_max} constraint
Figure 5.80: Hypergraph of the reformulation corresponding to the automaton of the \texttt{between\_min\_max} constraint
5.45 bin Packing

<table>
<thead>
<tr>
<th>Description</th>
<th>Links</th>
<th>Graph</th>
<th>Automaton</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Derived from cumulative.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bin_packing(CAPACITY, ITEMS)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPACITY : int</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ITEMS : collection(bin−dvar, weight−int)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAPACITY ≥ 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>required(ITEMS, [bin, weight])</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ITEMS.weight ≥ 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ITEMS.weight ≤ CAPACITY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Given several items of the collection ITEMS (each of them having a specific weight), and different bins of a fixed capacity, assign each item to a bin so that the total weight of the items in each bin does not exceed CAPACITY.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(5, [bin−3 weight−4, bin−1 weight−3, bin−3 weight−1])</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The bin_packing constraint holds since the sum of the height of items that are assigned to bins 1 and 3 is respectively equal to 3 and 5. The previous quantities are both less than or equal to the maximum CAPACITY 5. Figure 5.81 shows the solution associated with the example.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Figure 5.81: Bin-packing solution</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Typical

\[ \text{CAPACITY} > \text{maxval}(\text{ITEMS.weight}) \]
\[ \text{CAPACITY} \leq \text{sum}(\text{ITEMS.weight}) \]
\[ |\text{ITEMS}| > 1 \]
\[ \text{range}(\text{ITEMS.bin}) > 1 \]
\[ \text{range}(\text{ITEMS.weight}) > 1 \]
\[ \text{ITEMS.bin} \geq 0 \]
\[ \text{ITEMS.weight} > 0 \]

Symmetries

- CAPACITY can be increased.
- Items of ITEMS are permutable.
- ITEMS.weight can be decreased to any value \( \geq 0 \).
- All occurrences of two distinct values of ITEMS.bin can be swapped; all occurrences of a value of ITEMS.bin can be renamed to any unused value.

Arg. properties

Contractible wrt. ITEMS.

Remark

Note the difference with the classical bin-packing problem [256, page 221] where one wants to find solutions that minimise the number of bins. In our case each item may be assigned only to specific bins (i.e., the different values of the bin variable) and the goal is to find a feasible solution. This constraint can be seen as a special case of the cumulative constraint [1], where all task durations are equal to 1.

In [358] the CAPACITY parameter of the bin_packing constraint is replaced by a collection of domain variables representing the load of each bin (i.e., the sum of the weights of the items assigned to a bin). This allows representing problems where a minimum level has to be reached in each bin.

Coffman and al. give in [112] the worst case bounds of different list algorithms for the bin packing problem (i.e., given a positive integer CAPACITY and a list \( L \) of integer sizes \( \text{weight}_1, \text{weight}_2, \ldots, \text{weight}_n \) \( (0 \leq \text{weight}_i \leq \text{CAPACITY}) \), what is the smallest integer \( m \) such that there is a partition \( L = L_1 \cup L_2 \cup \ldots \cup L_m \) satisfying \( \sum_{\text{weight}_j \in L_j} \text{weight}_j \leq \text{CAPACITY} \) for all \( j \in [1, m] \)?

Algorithm

Initial filtering algorithms are described in [271, 268, 269, 270, 358]. More recently, linear continuous relaxations based on the graph associated with the dynamic programming approach for knapsack by Trick [383], and on the more compact model introduced by Carvalho [96, 97] are presented in [84].

Systems

pack in Choco, binpacking in Gecode, bin_packing in MiniZinc.

See also

generalisation: bin_packing_capa (fixed overall capacity replaced by non-fixed capacity), cumulative (task of duration 1 replaced by task of given duration), cumulative_two_d (task of duration 1 replaced by square of size 1 with a height), indexed_sum (negative contribution also allowed, fixed capacity replaced by a set of variables).

used in graph description: sum_ctr.

Keywords

application area: assignment.

characteristic of a constraint: automaton, automaton with array of counters.
constraint type: resource constraint.
final graph structure: acyclic, bipartite, no loop.
modelling: assignment dimension, assignment to the same set of values.
modelling exercises: assignment to the same set of values.
Arc input(s) ITEMS ITEMS
Arc generator PRODUCT \rightarrow \text{collection}(\text{items1,items2})
Arc arity 2
Arc constraint(s) \text{items1.bin} = \text{items2.bin}
Graph class • ACYCLIC
• BIPARTITE
• NO LOOP
Sets \text{SUCC} \mapsto 
\begin{bmatrix}
\text{source, variables} - \text{col (VARIABLES.collection(var-dvar), ] [item(var - ITEMS.weight])}}
\end{bmatrix}
Constraint(s) on sets \text{sumCtr}(\text{variables,} \leq \text{CAPACITY})

Graph model We enforce the \text{sumCtr} constraint on the weight of the items that are assigned to the same bin.

Parts (A) and (B) of Figure 5.82 respectively show the initial and final graph associated with the Example slot. Each connected component of the final graph corresponds to the items that are all assigned to the same bin.

![Graph Diagram](image)

Figure 5.82: Initial and final graph of the bin packing constraint
Automaton

Figure 5.83 depicts the automaton associated with the bin_packing constraint. To each item of the collection ITEMS corresponds a signature variable $S_i$ that is equal to 1.

\[
\begin{align*}
&C[_0]=0 \\
&\text{arithmetic}(C, <=, \text{CAPACITY}) \\
&1, \{C[BIN_i]=C[BIN_i]+\text{WEIGHT}_i\}
\end{align*}
\]

Figure 5.83: Automaton of the bin_packing constraint
5.46 bin_packing_capa

**DESCRIPTION**

Derived from bin_packing.

**Constraint**

bin_packing_capa(BINS, ITEMS)

**Arguments**

| BINS | collection(id−int, capa−int) |
| ITEMS | collection(bin−dvar, weight−int) |

**Restrictions**

| \(|BINS| > 0 \|
| \(required(BINS, [id, capa])\|
| \(distinct(BINS, id)\|
| \(BINS.id >= 1\|
| \(BINS.id <= \|BINS\|\|
| \(BINS.capa >= 0\|
| \(required(ITEMS, [bin, weight])\|
| \(in_attr(ITEMS, bin, BINS.id)\|
| \(ITEMS.weight >= 0\|

**Purpose**

Given several items of the collection ITEMS (each of them having a specific weight), and different bins described the the items of collection BINS (each of them having a specific capacity capa), assign each item to a bin so that the total weight of the items in each bin does not exceed the capacity of the bin.

**Example**

\(\begin{pmatrix}
  \text{id} - 1 & \text{capa} - 4, \\
  \text{id} - 2 & \text{capa} - 3, \\
  \text{id} - 3 & \text{capa} - 5, \\
  \text{id} - 4 & \text{capa} - 3, \\
  \text{id} - 5 & \text{capa} - 3 \\
\end{pmatrix}\)

The bin_packing_capa constraint holds since the sum of the height of items that are assigned to bins 1 and 3 is respectively equal to 3 and 5. The previous quantities are respectively less than or equal to the maximum capacities 4 and 5 of bins 1 and 3. Figure 5.84 shows the solution associated with the example.

**Typical**

| \(|BINS| > 1 \|
| \(range(BINS.capa) > 1\|
| \(BINS.capa > maxval(ITEMS.weight)\|
| \(BINS.capa <= sum(ITEMS.weight)\|
| \(|ITEMS| > 1 \|
| \(range(ITEMS.bin) > 1\|
| \(range(ITEMS.weight) > 1\|
| \(ITEMS.weight > 0\|

The bin_packing_capa constraint holds since the sum of the height of items that are assigned to bins 1 and 3 is respectively equal to 3 and 5. The previous quantities are respectively less than or equal to the maximum capacities 4 and 5 of bins 1 and 3. Figure 5.84 shows the solution associated with the example.
Symmetries

- Items of BINS are permutable.
- Items of ITEMS are permutable.
- BINS.cap can be increased.
- ITEMS.weight can be decreased to any value $\geq 0$.
- All occurrences of two distinct values in BINS.id or ITEMS.bin can be swapped; all occurrences of a value in BINS.id or ITEMS.bin can be renamed to any unused value.

Arg. properties

Contractible wrt. ITEMS.

Remark

In MiniZinc (http://www.g12.cs.mu.oz.au/minizinc/) there is also a constraint called bin_packing_load which, for each bin has a domain variable that is equal to the sum of the weights assigned to the corresponding bin.

Systems

pack in Choco, binpacking in Gecode, bin_packing_capa in MiniZinc.

See also

generalisation: indexed_sum (negative contribution also allowed).
specialisation: bin_packing (non-fixed capacity replaced by fixed overall capacity).

Keywords

application area: assignment.
constraint type: predefined constraint, resource constraint.
modelling: assignment dimension, assignment to the same set of values.
modelling exercises: assignment to the same set of values.

Figure 5.84: Bin-packing solution
### 5.47 binary_tree

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from <code>tree</code>.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td><code>binary_tree(NTREES, NODES)</code></td>
<td></td>
</tr>
</tbody>
</table>
| Arguments   | `NTREES : dvar`  
`NODES : collection(index-int, succ-dvar)` |       |
| Restrictions| `NTREES ≥ 0`  
`NTREES ≤ |NODES|`  
`required(NODES, [index, succ])`  
`NODES.index ≥ 1`  
`NODES.index ≤ |NODES|`  
`distinct(NODES, index)`  
`NODES.succ ≥ 1`  
`NODES.succ ≤ |NODES|` |       |
| Purpose     | Cover the digraph $G$ described by the `NODES` collection with `NTREES` binary trees in such a way that each vertex of $G$ belongs to exactly one binary tree (i.e., each vertex of $G$ has at most two children). The edges of the binary trees are directed from their leaves to their respective root. |       |

Example

```
2,  
| index - 1 succ - 1,  
| index - 2 succ - 3,  
| index - 3 succ - 5,  
| index - 4 succ - 7,  
| index - 5 succ - 1,  
| index - 6 succ - 1,  
| index - 7 succ - 7,  
| index - 8 succ - 5 |
```

The `binary_tree` constraint holds since its second argument corresponds to the 2 (i.e., the first argument of the `binary_tree` constraint) binary trees depicted by Figure 5.85.

![Figure 5.85: The two binary trees corresponding to the Example slot](image-url)
Typical

| NTREES > 0  
| NTREES < |NOD|ES|  
| |NOD|ES| > 2 |

Symmetry

Items of NOD|ES| are permutable.

Arg. properties

Functional dependency: NTREES determined by NOD|ES|.

Reformulation

The binary.tree constraint can be expressed in term of (1) a set of |NOD|ES|² reified constraints for avoiding circuit between more than one node and of (2) |NOD|ES| reified constraints and of one sum constraint for counting the trees and of (3) a set of |NOD|ES|² reified constraints and of |NOD|ES| inequalities constraints for enforcing the fact that each vertex has at most two children.

1. For each vertex NOD|ES|[i] (i ∈ [1, |NOD|ES|]) of the NOD|ES| collection we create a variable Ri that takes its value within interval [1, |NOD|ES|]. This variable represents the rank of vertex NOD|ES|[i] within a solution. It is used to prevent the creation of circuit involving more than one vertex as explained now. For each pair of vertices NOD|ES|[i], NOD|ES|[j] (i, j ∈ [1, |NOD|ES|]) of the NOD|ES| collection we create a reified constraint of the form NOD|ES|[i].succ = NOD|ES|[j].index ∧ i ≠ j ⇒ R[i] < R[j]. The purpose of this constraint is to express the fact that, if there is an arc from vertex NOD|ES|[i] to another vertex NOD|ES|[j], then Ri should be strictly less than R[j].

2. For each vertex NOD|ES|[i] (i ∈ [1, |NOD|ES|]) of the NOD|ES| collection we create a 0-1 variable Bi and state the following reified constraint NOD|ES|[i].succ = NOD|ES|[i].index ⇔ Bi in order to force variable Bi to be set to value 1 if and only if there is a loop on vertex NOD|ES|[i]. Finally we create a constraint NTREES = B1 + B2 + ... + B|NOD|ES| for stating the fact that the number of trees is equal to the number of loops of the graph.

3. For each pair of vertices NOD|ES|[i], NOD|ES|[j] (i, j ∈ [1, |NOD|ES|]) of the NOD|ES| collection we create a 0-1 variable Bij and state the following reified constraint NOD|ES|[i].succ = NOD|ES|[j].index ∧ i ≠ j ⇔ Bij. Variable Bij is set to value 1 if and only if there is an arc from NOD|ES|[i] to NOD|ES|[j]. Then for each vertex NOD|ES|[j] (j ∈ [1, |NOD|ES|]) we create a constraint of the form B1j + B2j + ... + B|NOD|ES|[j] ≤ 2.

See also

generalisation: tree (at most two childrens replaced by no restriction on maximum number of children).

implied by: path.

implies: tree.

specialisation: path (at most two childrens replaced by at most one child).

Keywords

constraint type: graph constraint, graph partitioning constraint.

final graph structure: connected component, tree, one.succ.

modelling: functional dependency.
Arc input(s)  NODES
Arc generator  CLIQUE→collection(nodes1,nodes2)
Arc arity  2
Arc constraint(s)  nodes1.succ = nodes2.index
Graph property(ies)  • MAX_NSCC ≤ 1
                      • NCC = NTREES
                      • MAX_ID ≤ 2
Graph class  ONE_SUCC

Graph model  We use the same graph constraint as for the tree constraint, except that we add the graph property MAX_ID ≤ 2, which constraints the maximum in-degree of the final graph to not exceed 2. MAX_ID does not consider loops: This is why we do not have any problem with the root of each tree.

Parts (A) and (B) of Figure 5.86 respectively show the initial and final graph associated with the Example slot. Since we use the NCC graph property, we display the two connected components of the final graph. Each of them corresponds to a binary tree. Since we use the MAX_IN_DEGREE graph property, we also show with a double circle a vertex that has a maximum number of predecessors.

The binary_tree constraint holds since all strongly connected components of the final graph have no more than one vertex, since NTREES = NCC = 2 and since MAX_ID = 2.
Figure 5.86: Initial and final graph of the binary_tree constraint
5.48 bipartite

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[131]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>bipartite(NODES)</td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>NODES : collection(index=int, succ=svar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(NODES,[index, succ])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.index ≥ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.index ≤</td>
<td>NODES</td>
</tr>
<tr>
<td></td>
<td>distinct(NODES, index)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.succ ≥ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.succ ≤</td>
<td>NODES</td>
</tr>
</tbody>
</table>

Purpose
Consider a digraph $G$ described by the NODES collection. Select a subset of arcs of $G$ so that the corresponding graph is symmetric (i.e., if there is an arc from $i$ to $j$, there is also an arc from $j$ to $i$) and bipartite (i.e., there is no cycle involving an odd number of vertices).

Example
```
(index - 1 succ = {2, 3},
  index - 2 succ = {1, 4},
  index - 3 succ = {1, 4, 5},
  index - 4 succ = {2, 3, 6},
  index - 5 succ = {3, 6},
  index - 6 succ = {4, 5})
```

The bipartite constraint holds since the NODES collection depicts a symmetric graph with no cycle involving an odd number of vertices. The corresponding graph is depicted by Figure 5.87.

![Figure 5.87: The bipartite graph associated with the example](image)

Typical
$|\text{NODES}| > 2$

Symmetry
Items of NODES are permutable.

Algorithm
The sketch of a filtering algorithm for the bipartite constraint is given in [131, page 91]. Beside enforcing the fact that the graph is symmetric, it checks that the subset of mandatory vertices and arcs is bipartite and removes all potential arcs that would make the previous graph non-bipartite.
See also

used in graph description: in_set.

Keywords

constraint arguments: constraint involving set variables.
constraint type: graph constraint.
filtering: DFS-bottleneck.
final graph structure: bipartite, symmetric.
Graph model

Part (A) of Figure 5.88 shows the initial graph from which we start. It is derived from the set associated with each vertex. Each set describes the potential values of the succ attribute of a given vertex. Part (B) of Figure 5.88 gives the final graph associated with the Example slot.

Figure 5.88: Initial and final graph of the bipartite set constraint
5.49 calendar

**DESCRIPTION**

**Constraint**
calendar(INSTANTS, MACHINES)

**Type**
UNAVAILABILITIES : collection(low−int, up−int)

**Arguments**
INSTANTS : collection(machine−dvar, virtual−dvar, ireal−dvar, flagend−int)
MACHINES : collection(id−int, cal − UNAVAILABILITIES)

**Restrictions**
required(UNAVAILABILITIES, [low, up])
UNAVAILABILITIES.low ≤ UNAVAILABILITIES.up
required(INSTANTS, [machine, virtual, ireal, flagend])
in_attr(INSTANTS, machine, MACHINES, id)
INSTANTS.flagend ≥ 0
INSTANTS.flagend ≤ 1
|MACHINES| > 0
required(MACHINES, [id, cal])
distinct(MACHINES, id)

**Purpose**
Makes the link between an universal calendar and resource dependent calendars. Given a collection of machines MACHINES where each machine is defined by its identifier and its unavailability periods the calendar constraint maps items of real and virtual dates depending on the machine assignment as well as of the fact that we consider start (flagend = 0) or end (flagend = 1) times. Virtual dates on a given machine m do not consider the unavailability periods on m, while real dates consider all time points.

**Example**

```
(machine − 1 virtual − 2 ireal − 3 flagend − 0,
machine − 1 virtual − 5 ireal − 6 flagend − 1,
machine − 2 virtual − 4 ireal − 5 flagend − 0,
machine − 2 virtual − 6 ireal − 9 flagend − 1,
machine − 3 virtual − 2 ireal − 2 flagend − 0,
machine − 3 virtual − 5 ireal − 5 flagend − 1,
machine − 4 virtual − 2 ireal − 2 flagend − 0,
machine − 4 virtual − 7 ireal − 9 flagend − 1
id − 1 cal − ⟨low − 2 up − 2, low − 6 up − 7⟩,
id − 2 cal − ⟨low − 2 up − 2, low − 6 up − 7⟩,
id − 3 cal − [],
id − 4 cal − ⟨low − 3 up − 4⟩)
```

Figure 5.89 illustrates the example. It present four machines with their respective
unavailability periods (in grey) as well as four tasks (in blue and pink). Each item of the INSTANTS collection corresponds to the start or to the end of one of the previous four tasks. The calendar constraint holds since:

- The real date 3 (INSTANTS[1].ireal = 3) associated with the start (INSTANTS[1].flagend = 0) of task (a) in the universal time corresponds to the virtual date 2 (INSTANTS[1].virtual = 2) on machine 1 (INSTANTS[1].machine = 1).
- The real date 6 (INSTANTS[2].ireal = 6) associated with the end (INSTANTS[2].flagend = 1) of task (a) in the universal time corresponds to the virtual date 5 (INSTANTS[2].virtual = 5) on machine 1 (INSTANTS[2].machine = 1).
- The real date 5 (INSTANTS[3].ireal = 5) associated with the start (INSTANTS[3].flagend = 0) of task (b) in the universal time corresponds to the virtual date 4 (INSTANTS[3].virtual = 4) on machine 2 (INSTANTS[3].machine = 2).
- The real date 9 (INSTANTS[4].ireal = 9) associated with the end (INSTANTS[4].flagend = 1) of task (b) in the universal time corresponds to the virtual date 6 (INSTANTS[4].virtual = 6) on machine 2 (INSTANTS[4].machine = 2).
- The real date 2 (INSTANTS[5].ireal = 2) associated with the start (INSTANTS[5].flagend = 0) of task (c) in the universal time corresponds to the virtual date 2 (INSTANTS[5].virtual = 2) on machine 3 (INSTANTS[5].machine = 3).
- The real date 5 (INSTANTS[6].ireal = 5) associated with the end (INSTANTS[6].flagend = 1) of task (c) in the universal time corresponds to the virtual date 5 (INSTANTS[6].virtual = 5) on machine 3 (INSTANTS[6].machine = 3).
- The real date 2 (INSTANTS[7].ireal = 2) associated with the start (INSTANTS[7].flagend = 0) of task (d) in the universal time corresponds to the virtual date 2 (INSTANTS[7].virtual = 2) on machine 4 (INSTANTS[7].machine = 4).
- The real date 9 (INSTANTS[8].ireal = 9) associated with the end (INSTANTS[8].flagend = 1) of task (d) in the universal time corresponds to the virtual date 7 (INSTANTS[8].virtual = 7) on machine 4 (INSTANTS[8].machine = 4).

Typical

| INSTANTS | > 1 |
| MACHINES | > 1 |

Symmetries

- Items of INSTANTS are permutable.
- Items of MACHINES are permutable.

Arg. properties

Contractible wrt. INSTANTS.

Usage

The calendar constraint is used as a channelling constraint in resource scheduling problems where resources have unavailability periods that can preempt the execution of a task. In this context two time coordinates systems are used:
A first coordinate system, so called the *virtual coordinate system*, ignores all unavailability periods on the different resources. All resource constraints are stated within this virtual coordinate system.

A second coordinate system, so called the *real coordinate system*, corresponds to the real time. All temporal constraints (e.g., precedence constraints) are stated within this real coordinate system.

In this context, each task has a *virtual origin*, a *virtual duration*, a *virtual end*, a *real origin*, a *real duration*, a *real end* and the *calendar constraint* links together the virtual origin and the real origin as well as the virtual end and the real end. The virtual duration (i.e., the real duration plus the sum of the unavailability periods crossed by the task) is linked to the virtual end and the virtual origin through an equality constraint on the difference between the virtual end and the virtual origin. The real duration is linked in a similar way to the real end and the real origin. The keyword *scheduling with machine choice, calendars and preemption* provides a concrete example of resource scheduling problem using the calendar constraint.

**Reformulation**

The calendar constraint can be reformulated into two generalised case constraints (i.e., two case constraints augmented with linear constraints). Part (A) (respectively Part (B)) of Figure 5.90 provides the dag that allows mapping the virtual start and real start (respectively the virtual end and real end) of a task. This dag can be computed directly from the arguments of the calendar constraint:

1. We create an initial root node labelled by $m$ and we partition the set of machines into classes of consecutive machines that all share exactly the same unavailability periods. For each such class we create an arc from the root node to a new node $vs$ labelled by the corresponding interval of consecutive machines identifiers. In Part (A) this corresponds to node $m$ and its three outgoing arcs respectively labelled by intervals $[1, 2]$, $[3, 3]$ and $[4, 4]$.

2. For each class of consecutive machines found previously, we label in increasing order each timepoint that is not part of an unavailability period. We create an arc from the corresponding node $vs$ for each maximum interval of available timepoints to a new node labelled by $rs$. In Part (A) this translate to:

   - For the class corresponding to machines 1 and 2 we create three outgoing arcs labelled by the time intervals $[1, 1]$, $[2, 4]$ and $[5, 6]$.

![Figure 5.89: Four machines with their unavailability periods as well as four tasks assigned to these machines (virtual dates mentioned in the Example slot use a bold font)](image-url)
• For the class corresponding to machine 3 we create the outgoing arc labelled by time interval $[1, 9]$.
• For the class corresponding to machine 4 we create the two outgoing arcs labelled by the time intervals $[1, 2]$ and $[3, 7]$.

3. For each class of consecutive machines and for each maximum interval $[i, j]$ of available timepoints previously computed, we find out the number of unavailable timepoints $b_i$ on the same class of machines that are located before the virtual date $i$. We create an outgoing arc from the corresponding node $rs$ to a new node labelled by true (there is one single true node for the full dag). This arc is labelled by the interval $[i + b_i, j + b_i]$ and by the linear constraint $rs = vs + b_i$. In Part (A) this translate to:

• For the class corresponding to machines 1 and 2 and for each $rs$ node associated with the time intervals $[1, 1]$, $[2, 4]$ and $[5, 6]$ we respectively create an outgoing arc labelled by intervals $[1, 1]$, $[3, 5]$ and $[8, 9]$. To each of these arcs we also respectively associate the linear constraints $rs = vs + 0$ (+0 since on machines 1 and 2 there is no unavailability period before the virtual date 1), $rs = vs + 1$ (+1 since on machines 1 and 2 there is one single unavailable timepoint before the virtual date 2) and $rs = vs + 3$ (+3 since on machines 1 and 2 there is three unavailable timepoints before the virtual date 5).

• For the class corresponding to machine 3 and for the $rs$ node associated with the time interval $[1, 9]$ we create the outgoing arc labelled by time interval $[1, 9]$ and by $rs = vs + 0$ (i.e., since their is no unavailability period at all on machine 3).

• For the class corresponding to machine 4 and for each $rs$ node associated with the time intervals $[1, 2]$ and $[3, 7]$ we respectively create an outgoing arc labelled by $[1, 2]$ and $[5, 9]$. To each of these arcs we also respectively associate the linear constraints $rs = vs + 0$ (+0 since on machine 4 there is no unavailability period before the virtual date 1) and $rs = vs + 2$ (+2 since on machine 4 there is two unavailable timepoints before the virtual date 3).

The calendar constraint can also be reformulated into a conjunction of reified constraints. This is done by generating, for each pair of items $(I, M)$ of the INSTANTS and MACHINES collections, a set of reified constraints expressing:

• The link between the real and the virtual dates under the hypothesis that the machine attribute of item $I$ is assigned to the value of the id attribute of item $M$. More precisely, we generate one reified constraint for each available time interval on machine id.

• The fact that a real date should not be located within an unavailability period of its corresponding machine.

Operationally, this leads to the following cases:

1. When machine id has no unavailability at all we state an equality constraint between the real and virtual dates.

2. When the real date is located before the first unavailability period we also state an equality constraint between the real and virtual dates.

3. When the real date is located between two consecutive unavailability periods we state:
Figure 5.90: The two generalised case constraints for respectively mapping (1) the virtual start and real start of a task corresponding to the Example slot as well as (2) the virtual end and real end; dags were generated under the hypothesis that the virtual and real dates are located in [1, 9].
• An equality constraint between the real date and the virtual date plus the sum of all unavailabilities located before the real date.

• An implication between the fact that the real date belongs to the first unavailability period (among the two consecutive unavailability periods) and the fact that the real date is not assigned to the machine that contains the unavailability period.

4. When the real date is located after the last unavailability period we state:

• An equality constraint between the real date and the virtual date plus the sum of all unavailabilities.

• An implication between the fact that the real date belongs to the last unavailability period and the fact that the real date is not assigned to the machine that contains the unavailability period.

As an example consider again consider the instance given in the Example slot. For the start of task a (i.e., the first item $\langle$ machine $\rightarrow$ virtual $\rightarrow$ ireal $\rightarrow$ flagend $\rightarrow$ 0$\rangle$ of collection INSTANTS), we generate the following reified constraints, where equivalences of the form $\text{true} \leftrightarrow \text{true}$ are shown in bold:

• (if task a is assigned on machine 1)
  * before $[2, 2]$:
    \begin{align*}
    1 &= 1 \land 3 < 2 \iff 1 = 1 \land 3 = 2 \\
    * &\text{between } [2, 2] \text{ and } [6, 7]:
    1 &= 1 \land 3 > 2 \land 3 < 6 \iff 1 = 1 \land 3 = 2 + 1 \\
    * &\text{after } [6, 7]:
    1 &= 1 \land 3 > 7 \iff 1 = 1 \land 3 = 2 + 3 \\
    * &\text{do not cross } [2, 2], [6, 7]:
    3 \in [2, 2] \implies 1 \neq 1, 3 \in [6, 7] \iff 1 \neq 1
  \end{align*}

• (if task a is assigned on machine 2)
  * before $[2, 2]$:
    \begin{align*}
    1 &= 2 \land 3 < 2 \iff 1 = 2 \land 3 = 2 \\
    * &\text{between } [2, 2] \text{ and } [6, 7]:
    1 &= 2 \land 3 > 2 \land 3 < 6 \iff 1 = 2 \land 3 = 2 + 1 \\
    * &\text{after } [6, 7]:
    1 &= 2 \land 3 > 7 \iff 1 = 2 \land 3 = 2 + 3 \\
    * &\text{do not cross } [2, 2], [6, 7]:
    3 \in [2, 2] \implies 1 \neq 2, 3 \in [6, 7] \iff 1 \neq 2
  \end{align*}

• (if task a is assigned on machine 3)
  * no unavailability:
    \begin{align*}
    1 &= 3 \iff 1 = 3 \land 3 = 2
  \end{align*}

• (if task a is assigned on machine 4)
  * before $[3, 4]$:
    \begin{align*}
    1 &= 4 \land 3 < 3 \iff 1 = 4 \land 3 = 2 \\
    * &\text{after } [3, 4]:
    1 &= 4 \land 3 > 4 \iff 1 = 4 \land 3 = 2 + 2 \\
    * &\text{do not cross } [3, 4]:
    3 \in [3, 4] \implies 1 \neq 4
  \end{align*}
For the end of task a (i.e., the second item \( \text{machine} - 1 \ \text{virtual} - 5 \ \text{ireal} - 6 \ \text{flagend} - 1 \)) of collection INSTANTS, we generate the following reified constraints:

- (if task a is assigned on machine 1)
  - before [2, 2]: \( 1 \equiv 1 \land 6 \lessdot 3 \iff 1 \land 6 = 5 \)
  - between [2, 2] and [6, 7]: \( 1 \equiv 1 \land 6 \equiv 3 \land 6 \lessdot 7 \iff 1 \equiv 1 \land 6 = 5 + 1 \)
  - after [6, 7]: \( 1 \equiv 1 \land 6 \equiv 8 \iff 1 \equiv 1 \land 6 = 5 + 3 \)
  - do not cross [2, 2], [6, 7]: \( 6 \in [3, 3] \Rightarrow 1 \neq 1, 6 \in [7, 8] \Rightarrow 1 \neq 1 \)

- (if task a is assigned on machine 2)
  - before [2, 2]: \( 1 = 2 \land 6 \lessdot 3 \iff 1 = 2 \land 6 = 5 \)
  - between [2, 2] and [6, 7]: \( 1 = 2 \land 6 > 3 \land 6 \lessdot 7 \iff 1 = 2 \land 6 = 5 + 1 \)
  - after [6, 7]: \( 1 = 2 \land 6 \equiv 8 \iff 1 = 2 \land 6 = 5 + 3 \)
  - do not cross [2, 2], [6, 7]: \( 6 \in [3, 3] \Rightarrow 1 \neq 1, 6 \in [7, 8] \Rightarrow 1 \neq 2 \)

- (if task a is assigned on machine 3)
  - no unavailability: \( 1 = 3 \iff 1 = 3 \land 6 = 5 \)

- (if task a is assigned on machine 4)
  - before [3, 4]: \( 1 = 4 \land 6 \lessdot 4 \iff 1 = 2 \land 6 = 5 \)
  - after [3, 4]: \( 1 = 4 \land 6 \equiv 5 \iff 1 = 4 \land 6 = 5 + 2 \)
  - do not cross [3, 4]: \( 6 \in [4, 5] \Rightarrow 1 \neq 4 \)

For the start of task b (i.e., the third item \( \text{machine} - 2 \ \text{virtual} - 4 \ \text{ireal} - 5 \ \text{flagend} - 0 \)) of collection INSTANTS, we generate the following reified constraints:

- (if task b is assigned on machine 1)
  - before [2, 2]: \( 2 = 1 \land 5 \lessdot 2 \iff 2 = 1 \land 5 = 4 \)
  - between [2, 2] and [6, 7]: \( 2 = 1 \land 5 > 2 \land 5 \lessdot 6 \iff 2 = 1 \land 5 = 4 + 1 \)
  - after [6, 7]: \( 2 = 1 \land 5 \equiv 7 \iff 2 = 1 \land 5 = 4 + 3 \)
  - do not cross [2, 2], [6, 7]: \( 5 \in [2, 2] \Rightarrow 2 \neq 1, 5 \in [6, 7] \Rightarrow 2 \neq 1 \)

- (if task b is assigned on machine 2)
  - before [2, 2]: \( 2 = 2 \land 5 \lessdot 2 \iff 2 = 2 \land 5 = 4 \)
  - between [2, 2] and [6, 7]: \( 2 = 2 \land 5 > 2 \land 5 \lessdot 6 \iff 2 = 2 \land 5 = 4 + 1 \)
  - after [6, 7]: \( 2 = 2 \land 5 \equiv 7 \iff 2 = 2 \land 5 = 4 + 3 \)
  - do not cross [2, 2], [6, 7]: \( 5 \in [2, 2] \Rightarrow 2 \neq 2, 5 \in [6, 7] \Rightarrow 2 \neq 2 \)

- (if task b is assigned on machine 3)
  - no unavailability: \( 2 = 3 \iff 2 = 3 \land 5 = 4 \)

- (if task b is assigned on machine 4)
  - before [3, 4]: \( 2 = 4 \land 5 \lessdot 3 \iff 2 = 4 \land 5 = 4 \)
  - after [3, 4]: \( 2 = 4 \land 5 \equiv 4 \iff 2 = 4 \land 5 = 4 + 2 \)
  - do not cross [3, 4]: \( 5 \in [3, 4] \Rightarrow 2 \neq 4 \)
For the \textit{end of task} \(b\) (i.e., the fourth item \((\text{machine} - 2 \text{ virtual} - 6 \text{ ireal} - 9 \text{ flagend} - 1)\) of collection \textsc{instants}, we generate the following reified constraints:

- \textbf{(if task \(b\) is assigned on machine 1)}
  - before \([2, 2]\):
    \[ 2 = 1 \land 9 < 3 \iff 2 = 1 \land 9 = 6 \]
  - between \([2, 2] \text{ and } [6, 7]\):
    \[ 2 = 1 \land 9 > 3 \land 9 < 7 \iff 2 = 1 \land 9 = 6 + 1 \]
  - after \([6, 7]\):
    \[ 2 = 1 \land 9 > 8 \iff 2 = 1 \land 9 = 6 + 3 \]
  - do not cross \([2, 2], [6, 7]\):
    \[ 9 \in [3, 3] \Rightarrow 2 \neq 1, 9 \in [7, 8] \Rightarrow 2 \neq 1 \]

- \textbf{(if task \(b\) is assigned on machine 2)}
  - before \([2, 2]\):
    \[ 2 = 2 \land 9 < 3 \iff 2 = 2 \land 9 = 6 \]
  - between \([2, 2] \text{ and } [6, 7]\):
    \[ 2 = 2 \land 9 > 3 \land 9 < 7 \iff 2 = 2 \land 9 = 6 + 1 \]
  - after \([6, 7]\):
    \[ 2 = 2 \land 9 > 8 \iff 2 = 2 \land 9 = 6 + 3 \]
  - do not cross \([2, 2], [6, 7]\):
    \[ 9 \in [3, 3] \Rightarrow 2 \neq 2, 9 \in [7, 8] \Rightarrow 2 \neq 2 \]

- \textbf{(if task \(b\) is assigned on machine 3)}
  - no unavailability:
    \[ 2 = 3 \iff 2 = 3 \land 9 = 6 \]

- \textbf{(if task \(b\) is assigned on machine 4)}
  - before \([3, 4]\):
    \[ 2 = 4 \land 9 < 4 \iff 2 = 4 \land 9 = 6 \]
  - after \([3, 4]\):
    \[ 2 = 4 \land 9 > 5 \iff 2 = 4 \land 9 = 6 + 2 \]
  - do not cross \([3, 4]\):
    \[ 9 \in [4, 5] \Rightarrow 2 \neq 4 \]

For the \textit{start of task} \(c\) (i.e., the fifth item \((\text{machine} - 3 \text{ virtual} - 2 \text{ ireal} - 2 \text{ flagend} - 0)\) of collection \textsc{instants}, we generate the following reified constraints:

- \textbf{(if task \(c\) is assigned on machine 1)}
  - before \([2, 2]\):
    \[ 3 = 1 \land 2 < 2 \iff 3 = 1 \land 2 = 2 \]
  - between \([2, 2] \text{ and } [6, 7]\):
    \[ 3 = 1 \land 2 > 2 \land 2 < 6 \iff 3 = 1 \land 2 = 2 + 1 \]
  - after \([6, 7]\):
    \[ 3 = 1 \land 2 > 7 \iff 3 = 1 \land 2 = 2 + 3 \]
  - do not cross \([2, 2], [6, 7]\):
    \[ 2 \in [2, 2] \Rightarrow 3 \neq 1, 2 \in [6, 7] \Rightarrow 3 \neq 1 \]

- \textbf{(if task \(c\) is assigned on machine 2)}
  - before \([2, 2]\):
    \[ 3 = 2 \land 2 < 2 \iff 3 = 2 \land 2 = 2 \]
  - between \([2, 2] \text{ and } [6, 7]\):
    \[ 3 = 2 \land 2 > 2 \land 2 < 6 \iff 3 = 2 \land 2 = 2 + 1 \]
  - after \([6, 7]\):
    \[ 3 = 2 \land 2 > 7 \iff 3 = 2 \land 2 = 2 + 3 \]
  - do not cross \([2, 2], [6, 7]\):
    \[ 2 \in [2, 2] \Rightarrow 3 \neq 2, 2 \in [6, 7] \Rightarrow 3 \neq 2 \]

- \textbf{(if task \(c\) is assigned on machine 3)}
  - no unavailability:
    \[ 3 = 3 \iff 3 = 3 \land 2 = 2 \]

- \textbf{(if task \(c\) is assigned on machine 4)}
  - before \([3, 4]\):
    \[ 3 = 4 \land 2 < 3 \iff 3 = 4 \land 2 = 2 \]
  - after \([3, 4]\):
    \[ 3 = 4 \land 2 > 4 \iff 3 = 4 \land 2 = 2 + 2 \]
  - do not cross \([3, 4]\):
    \[ 2 \in [3, 4] \Rightarrow 3 \neq 4 \]
For the end of task c (i.e., the sixth item \{machine−3 virtual−5 ireal−5 flagend−1\}) of collection INSTANTS), we generate the following reified constraints:

- (if task c is assigned on machine 1)
  * before [2, 2]: \(3 = 1 \land 5 < 3 \Leftrightarrow 3 = 1 \land 5 = 5\)
  * between [2, 2] and [6, 7]: \(3 = 1 \land 5 > 3 \land 5 < 7 \Leftrightarrow 3 = 1 \land 5 = 5 + 1\)
  * after [6, 7]: \(3 = 1 \land 5 > 8 \Leftrightarrow 3 = 1 \land 5 = 5 + 3\)
  * do not cross [2, 2], [6, 7]: \(5 \in [3..3] \Rightarrow 3 \neq 1, 5 \in [7..8] \Rightarrow 3 \neq 1\)

- (if task c is assigned on machine 2)
  * before [2, 2]: \(3 = 2 \land 5 < 3 \Leftrightarrow 3 = 2 \land 5 = 5\)
  * between [2, 2] and [6, 7]: \(3 = 2 \land 5 > 3 \land 5 < 7 \Leftrightarrow 3 = 2 \land 5 = 5 + 1\)
  * after [6, 7]: \(3 = 2 \land 5 > 8 \Leftrightarrow 3 = 2 \land 5 = 5 + 3\)
  * do not cross [2, 2], [6, 7]: \(5 \in [3..3] \Rightarrow 3 \neq 2, 5 \in [7..8] \Rightarrow 3 \neq 2\)

- (if task c is assigned on machine 3)
  * no unavailability: \(3 = 3 \Leftrightarrow 3 = 3 \land 5 = 5\)

- (if task c is assigned on machine 4)
  * before [3, 4]: \(3 = 4 \land 5 < 4 \Leftrightarrow 3 = 4 \land 5 = 5\)
  * after [3, 4]: \(3 = 4 \land 5 > 5 \Leftrightarrow 3 = 4 \land 5 = 5 + 2\)
  * do not cross [3, 4]: \(5 \in [4..5] \Rightarrow 3 \neq 4\)

For the start of task d (i.e., the seventh item \{machine − 4 virtual − 2 ireal − 2 flagend − 0\}) of collection INSTANTS), we generate the following reified constraints:

- (if task d is assigned on machine 1)
  * before [2, 2]: \(4 = 1 \land 2 < 2 \Leftrightarrow 4 = 1 \land 2 = 2\)
  * between [2, 2] and [6, 7]: \(4 = 1 \land 2 > 2 \land 2 < 6 \Leftrightarrow 4 = 1 \land 2 = 2 + 1\)
  * after [6, 7]: \(4 = 1 \land 2 > 7 \Leftrightarrow 4 = 1 \land 2 = 2 + 3\)
  * do not cross [2, 2], [6, 7]: \(2 \in [2, 2] \Rightarrow 4 \neq 1, 2 \in [6, 7] \Rightarrow 4 \neq 1\)

- (if task d is assigned on machine 2)
  * before [2, 2]: \(4 = 2 \land 2 < 2 \Leftrightarrow 4 = 2 \land 2 = 2\)
  * between [2, 2] and [6, 7]: \(4 = 2 \land 2 > 2 \land 2 < 6 \Leftrightarrow 4 = 2 \land 2 = 2 + 1\)
  * after [6, 7]: \(4 = 2 \land 2 > 7 \Leftrightarrow 4 = 2 \land 2 = 2 + 3\)
  * do not cross [2, 2], [6, 7]: \(2 \in [2, 2] \Rightarrow 4 \neq 2, 2 \in [6, 7] \Rightarrow 4 \neq 2\)

- (if task d is assigned on machine 3)
  * no unavailability: \(4 = 3 \Leftrightarrow 4 = 3 \land 2 = 2\)

- (if task d is assigned on machine 4)
  * before [3, 4]: \(4 = 4 \land 2 < 3 \Leftrightarrow 4 = 4 \land 2 = 2\)
  * after [3, 4]: \(4 = 4 \land 2 > 4 \Leftrightarrow 4 = 4 \land 2 = 2 + 2\)
  * do not cross [3, 4]: \(2 \in [3, 4] \Rightarrow 4 \neq 4\)
For the end of task d (i.e., the eighth item \text{⟨machine−4 virtual−7 ireal−9 flagend−1⟩} of collection INSTANTS), we generate the following reified constraints:

- **(if task d is assigned on machine 1)**
  - Before \text{⟨2, 2⟩}:
    \[4 = 1 \land 9 < 3 \iff 4 = 1 \land 9 = 7\]
  - Between \text{⟨2, 2⟩} and \text{⟨6, 7⟩}:
    \[4 = 1 \land 9 > 3 \land 9 < 7 \iff 4 = 1 \land 9 = 7 + 1\]
  - After \text{⟨6, 7⟩}:
    \[4 = 1 \land 9 > 8 \iff 4 = 1 \land 9 = 7 + 3\]
  - Do not cross \text{⟨2, 2⟩, ⟨6, 7⟩}: \[9 \in [3, 3] \Rightarrow 4 \neq 1, 9 \in [7, 8] \Rightarrow 4 \neq 1\]

- **(if task d is assigned on machine 2)**
  - Before \text{⟨2, 2⟩}:
    \[4 = 2 \land 9 < 3 \iff 4 = 2 \land 9 = 7\]
  - Between \text{⟨2, 2⟩} and \text{⟨6, 7⟩}:
    \[4 = 2 \land 9 > 3 \land 9 < 7 \iff 4 = 2 \land 9 = 7 + 1\]
  - After \text{⟨6, 7⟩}:
    \[4 = 2 \land 9 > 8 \iff 4 = 2 \land 9 = 7 + 3\]
  - Do not cross \text{⟨2, 2⟩, ⟨6, 7⟩}: \[9 \in [3, 3] \Rightarrow 4 \neq 2, 9 \in [7, 8] \Rightarrow 4 \neq 2\]

- **(if task d is assigned on machine 3)**
  - No unavailability:
    \[4 = 3 \iff 4 = 3 \land 9 = 7\]

- **(if task d is assigned on machine 4)**
  - Before \text{⟨3, 4⟩}:
    \[4 = 4 \land 9 < 4 \iff 4 = 4 \land 9 = 7\]
  - After \text{⟨3, 4⟩}:
    \[4 = 4 \land 9 > 5 \iff 4 = 4 \land 9 = 7 + 2\]
  - Do not cross \text{⟨3, 4⟩}:
    \[9 \in [4, 5] \Rightarrow 4 \neq 4\]

See also

- **common keyword:** cumulative (scheduling constraint),
  - cumulatives (scheduling with machine choice, calendars and preemption),
  - diffn (multi-site employee scheduling with calendar constraints, scheduling with machine choice, calendars and preemption),
  - disjunctive (scheduling constraint),
  - geost (multi-site employee scheduling with calendar constraints, scheduling with machine choice, calendars and preemption).

**Keywords**

- **constraint type:** predefined constraint, temporal constraint, scheduling constraint.
- **modelling:** channelling constraint, multi-site employee scheduling with calendar constraints, scheduling with machine choice, calendars and preemption, assignment dimension.
- **modelling exercises:** multi-site employee scheduling with calendar constraints, scheduling with machine choice, calendars and preemption.
## 5.50 cardinality_atleast

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from global_cardinality.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>cardinality_atleast(ATLEAST, VARIABLES, VALUES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>ATLEAST : dvar</td>
<td>VARIABLES : collection(var−dvar)</td>
<td>VALUES : collection(val−int)</td>
</tr>
<tr>
<td>Restrictions</td>
<td>ATLEAST ≥ 0</td>
<td>ATLEAST ≤</td>
<td>VARIABLES</td>
</tr>
<tr>
<td>Purpose</td>
<td>ATLEAST is the minimum number of time that a value of VALUES is taken by the variables of the collection VARIABLES.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>(1, (3, 3, 8), (3, 8))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In this example, values 3 and 8 are respectively used 2, and 1 times. The cardinality_atleast constraint holds since its first argument ATLEAST = 1 is assigned to the minimum number of time that values 3 and 8 occur in the collection (3, 3, 8).

### Typical

| ATLEAST > 0 | ATLEAST < |VARIABLES| | | | | | | | |
| | [VARIABLES] > 1 | [VALUES] > 0 | [VARIABLES] > |VALUES| | | | | | |

### Symmetries

- Items of VARIABLES are **permutable**.
- Items of VALUES are **permutable**.
- An occurrence of a value of VARIABLES.var that does not belong to VALUES.val can be **replaced** by any other value that also does not belong to VALUES.val.
- All occurrences of two distinct values in VARIABLES.var or VALUES.val can be **swapped**; all occurrences of a value in VARIABLES.var or VALUES.val can be **renamed** to any unused value.

### Arg. properties

- **Functional dependency**: ATLEAST determined by VARIABLES and VALUES.

### Usage

An application of the cardinality_atleast constraint is to enforce a minimum use of values.
Remark

This is a restricted form of a variant of an `among` constraint and of the `global_cardinality` constraint. In the original `global_cardinality` constraint, one specifies for each value its minimum and maximum number of occurrences.

Algorithm

See `global_cardinality` [32].

See also

generalisation: `global_cardinality` (*single count variable replaced by an individual count variable for each value)*.

Keywords

application area: assignment.
characteristic of a constraint: automaton, automaton with array of counters.
constraint arguments: pure functional dependency.
constraint type: value constraint.
filtering: arc-consistency.
final graph structure: acyclic, bipartite, no loop.
modelling: functional dependency, at least.
Arc input(s)  VARIABLES VALUES
Arc generator  \( \text{PRODUCT} \rightarrow \text{collection}(\text{variables}, \text{values}) \)
Arc arity  2
Arc constraint(s)  \( \text{variables.var} \neq \text{values.val} \)
Graph property(ies)  \( \text{MAX.ID} = |\text{VARIABLES}| - \text{ATLEAST} \)
Graph class  • ACYCLIC
• BIPARTITE
• NO_LOOP

Graph model  Using directly the graph property \( \text{MIN.ID} = \text{ATLEAST} \), and replacing the disequality of the arc constraint by an equality does not work since it ignores values that are not assigned to any variable. This comes from the fact that isolated vertices are removed from the final graph.

Parts (A) and (B) of Figure 5.91 respectively show the initial and final graph associated with the \textbf{Example} slot. Since we use the \textbf{MAX.ID} graph property, the vertex with the maximum number of predecessor (i.e., namely two predecessors) is stressed with a double circle. As a consequence the first argument \( \text{ATLEAST} \) of the \texttt{cardinality\_atleast} constraint is assigned to the total number of variables 3 minus 2.

![Initial and final graph of the cardinality\_atleast constraint](image)

Figure 5.91: Initial and final graph of the \texttt{cardinality\_atleast} constraint
Automaton

Figure 5.92 depicts the automaton associated with the \texttt{cardinality\_atleast} constraint. To each variable \texttt{VAR}\textsubscript{i} of the collection \texttt{VARIABLES} corresponds a 0-1 signature variable \texttt{S}\textsubscript{i}. The following signature constraint links \texttt{VAR}\textsubscript{i} and \texttt{S}\textsubscript{i}: \texttt{VAR}\textsubscript{i} \in \texttt{VALUES} \iff \texttt{S}\textsubscript{i}.

\begin{center}
\begin{tikzpicture}
\node   (A) at (0,0) {not\_in(VAR\textsubscript{i},VALUES)};
\node   (B) at (0,-2) {\texttt{S}};
\node   (C) at (0,-3) {\texttt{M}\geq\texttt{ATLEAST}};
\node   (D) at (0,-5) {\texttt{minimum\_except\_0(M,C)}};
\node   (E) at (0,-7) {\texttt{in}(VAR\textsubscript{i},VALUES), \{C[VAR\textsubscript{i}]=C[VAR\textsubscript{i}]+1\}};
\node   (F) at (0,-9) {\texttt{C[\_]=0}};
\path[->] (A) edge (B) (B) edge (C) (C) edge (D) (D) edge (E) (E) edge (F);
\end{tikzpicture}
\end{center}

\textbf{Figure 5.92:} Automaton of the \texttt{cardinality\_atleast} constraint
5.51 cardinality_atmost

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from global_cardinality.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>cardinality_atmost(ATMOST, VARIABLES, VALUES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>ATMOST : dvar</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES : collection(var−dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VALUES : collection(val−int)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>ATMOST ≥ 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ATMOST ≤</td>
<td>VARIABLES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES, var)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES, val)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>distinct(VARIABLES, val)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>ATMOST is the maximum number of occurrences of each value of VALUES within the variables of the collection VARIABLES.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>In this example, values 5, 7, 2 and 9 occur respectively 0, 1, 2 and 0 times within the collection ⟨2, 1, 7, 1, 2⟩. As a consequence, the cardinality_atmost constraint holds since its first argument ATMOST is assigned to the maximum number of occurrences 2.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Typical</td>
<td>ATMOST &gt; 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ATMOST &lt;</td>
<td>VARIABLES</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>VALUES</td>
<td>&gt; 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VARIABLES</td>
<td>&gt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VALUES</td>
<td>&gt;</td>
</tr>
<tr>
<td>Symmetries</td>
<td>• Items of VARIABLES are permutable.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Items of VALUES are permutable.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• An occurrence of a value of VARIABLES.val that does not belong to VALUES.val can be replaced by any other value that also does not belong to VALUES.val.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• All occurrences of two distinct values in VARIABLES.var or VALUES.val can be swapped; all occurrences of a value in VARIABLES.var or VALUES.val can be renamed to any unused value.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arg. properties</td>
<td>Functional dependency: ATMOST determined by VARIABLES and VALUES.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Usage</td>
<td>An application of the cardinality_atmost constraint is to enforce a maximum use of values.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Remark
This is a restricted form of a variant of the *among* constraint and of the *global_cardinality* constraint. In the original *global_cardinality* constraint, one specifies for each value its minimum and maximum number of occurrences.

Algorithm
See *global_cardinality* [322].

See also
*generalisation: global_cardinality* *(single count variable replaced by an individual count variable for each value)*, *multi_inter_distance* *(window of size 1 replaced by window of DIST consecutive values)*.

implied by: among.

Keywords
application area: assignment.
characteristic of a constraint: automaton, automaton with array of counters.
constraint arguments: pure functional dependency.
constraint type: value constraint.
filtering: arc-consistency.
final graph structure: acyclic, bipartite, no loop.
modelling: at most, functional dependency.
Arc input(s) | VARIABLES VALUES
---|---
Arc generator | \( \text{PRODUCT} \rightarrow \text{collection}(\text{variables}, \text{values}) \)
Arc arity | 2
Arc constraint(s) | \text{variables.var} = \text{values.val}
Graph property(ies) | \text{MAX_ID} = \text{ATMOST}
Graph class | • ACYCLIC
• BIPARTITE
• NOLOOP

Graph model

Parts (A) and (B) of Figure 5.93 respectively show the initial and final graph associated with the Example slot. Since we use the MAX_ID graph property, the vertex that has the maximum number of predecessor is stressed with a double circle.

![Graph model diagram](image_url)

Figure 5.93: Initial and final graph of the cardinality_atmost constraint
Automaton

Figure 5.94 depicts the automaton associated with the \texttt{cardinality\_atmost} constraint. To each variable $\text{VAR}_i$ of the collection $\text{VARIABLES}$ corresponds a 0-1 signature variable $S_i$. The following signature constraint links $\text{VAR}_i$ and $S_i$: $\text{VAR}_i \in \text{VALUES} \iff S_i$.

\[
\{ C[\_] = 0 \}
\]

\[
\text{not\_in}(\text{VAR}_i, \text{VALUES}) \rightarrow \text{in}(\text{VAR}_i, \text{VALUES}),
\]

\[
\text{arith}(C, \leq, \text{ATMOST}),
\]

\[
\{ C[\text{VAR}_i] = C[\text{VAR}_i] + 1 \}
\]

Figure 5.94: Automaton of the \texttt{cardinality\_atmost} constraint
### 5.52 cardinality_atmost_partition

<table>
<thead>
<tr>
<th>Description</th>
<th>Links</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Derived from <code>global_cardinality</code>.</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td><code>cardinality_atmost_partition(ATMOST,VARIABLES,PARTITIONS)</code></td>
<td></td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td><code>VALUES : collection(val=int)</code></td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td><code>ATMOST : dvar</code></td>
<td><code>VARIABLES : collection(var=dvar)</code></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>`</td>
<td>VALUES</td>
</tr>
<tr>
<td></td>
<td><code>ATMOST ≥ 0</code></td>
<td>`ATMOST ≤</td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td><code>ATMOST</code> is the maximum number of time that values of a same partition of <code>PARTITIONS</code> are taken by the variables of the collection <code>VARIABLES</code>.</td>
<td></td>
</tr>
</tbody>
</table>
| **Example** | \[
\begin{pmatrix}
2, \\
\{\text{var} \rightarrow 2, \\
\text{var} \rightarrow 3, \\
\text{var} \rightarrow 7, \\
\text{var} \rightarrow 1, \\
\text{var} \rightarrow 6, \\
\text{var} \rightarrow 0 \}
\end{pmatrix}, \\
\begin{pmatrix}
\text{p} \rightarrow (1,3), \\
\text{p} \rightarrow (4), \\
\text{p} \rightarrow (2,6) 
\end{pmatrix}
\] | In this example, two variables of the collection `VARIABLES = \{2,3,7,1,6,0\}` are assigned values of the first partition, no variable is assigned a value of the second partition, and finally two variables are assigned values of the last partition. As a consequence, the `cardinality_atmost_partition` constraint holds since its first argument `ATMOST` is assigned to the maximum number of occurrences 2. |
| **Typical** | `ATMOST > 0` | `ATMOST < |VARIABLES|` | `|VARIABLES| > 1` | `|VARIABLES| > |PARTITIONS|` |
### Symmetries
- Items of VARIABLES are permutable.
- Items of PARTITIONS are permutable.
- Items of PARTITIONS.p are permutable.

### Arg. properties
Functional dependency: AT MOST determined by VARIABLES and PARTITIONS.

### See also
- **generalisation:** global_cardinality (single count variable replaced by an individual count variable for each value and variable replaced by variable ∈ partition).
  - used in graph description: in.

### Keywords
- characteristic of a constraint: partition.
- constraint arguments: pure functional dependency.
- constraint type: value constraint.
- filtering: arc-consistency.
- final graph structure: acyclic, bipartite, no loop.
- modelling: at most, functional dependency.
Arc input(s)  VARIABLES, PARTITIONS
Arc generator  \( \text{PRODUCT} \rightarrow \text{collection}(\text{variables}, \text{partitions}) \)
Arc arity  2
Arc constraint(s)  \( \text{in}(\text{variables.var}, \text{partitions.p}) \)
Graph property(ies)  \( \text{MAX}\_\text{ID} = \text{ATMOST} \)
Graph class
- ACYCLIC
- BIPARTITE
- NO_LOOP

Graph model
Parts (A) and (B) of Figure 5.95 respectively show the initial and final graph associated with the Example slot. Since we use the MAX_ID graph property, a vertex with the maximum number of predecessor is stressed with a double circle.

Figure 5.95: Initial and final graph of the cardinality_ATMOST_partition constraint
### 5.53 change

**Description**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>CHIP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>change(NCHANGE, VARIABLES, CTR)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>nbchanges, similarity.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Arguments   | NCHANGE : dvar
VARIABLES : collection(var−dvar)
CTR : atom |       |           |
| Restrictions| NCHANGE ≥ 0
NCHANGE < |VARIABLES|
required(VARIABLES, var)
CTR ∈ [ =, ≠, <, ≥, >, ≤] |       |           |
| Purpose     | NCHANGE is the number of times that constraint CTR holds on consecutive variables of the collection VARIABLES. |       |           |

**Example**

\[(3, (4, 4, 3, 4, 1), \neq)\]
\[(1, (1, 2, 4, 3, 7), >)\]

In the first example the changes are located between values 4 and 3, 3 and 4 and 4 and 1. Consequently, the corresponding change constraint holds since its first argument NCHANGE is fixed to value 3.

In the second example the unique change occurs between values 4 and 3. Consequently, the corresponding change constraint holds since its first argument NCHANGE is fixed to 1.

**Typical**

| NCHANGE > 0
| | | |
| | | | |
| | | | |

**Symmetry**

One and the same constant can be added to the var attribute of all items of VARIABLES.

**Arg. properties**

- **Functional dependency**: NCHANGE determined by VARIABLES and CTR.
- **Contractible wrt. VARIABLES** when CTR ∈ [ ≠, <, ≥, >, ≤] and NCHANGE = 0.
- **Contractible wrt. VARIABLES** when CTR ∈ [ =, <, ≥, >, ≤] and NCHANGE = |VARIABLES| − 1.

**Usage**

This constraint can be used in the context of timetabling problems in order to put an upper limit on the number of changes of job types during a given period.
Remark

A similar constraint appears in [283, page 338] under the name of similarity constraint. The difference consists of replacing the arithmetic constraint CTR by a binary constraint. When CTR is equal to ≠ this constraint is called nbchanges in [380].

Algorithm

A first incomplete algorithm is described in [29]. The sketch of a filtering algorithm for the conjunction of the change and the stretch constraints based on dynamic programming achieving arc-consistency is mentioned by Lars Hellsten in [191, page 56].

Reformulation

The change constraint can be reformulated with the seq_bin constraint [290] that we now introduce. Given N a domain variable, X a sequence of domain variables, and C and B two binary constraints, seq_bin(N, X, C, B) holds if (1) N is equal to the number of C-stretches in the sequence X, and (2) B holds on any pair of consecutive variables in X. A C-stretch is a generalisation of the notion of stretch introduced by G. Pesant [285], where the equality constraint is made explicit by replacing it by a binary constraint C, i.e., a C-stretch is a maximal length subsequence of X for which the binary constraint C is satisfied on consecutive variables. change(NCHANGE, VARIABLES, CTR) can be reformulated as N = N1 − 1 ∧ seq_bin(N1, X, ¬CTR, true), where true is the universal constraint.

Used in

pattern.

See also

common keyword: change_partition, circular_change(number of changes in a sequence of variables with respect to a binary constraint), cyclic_change, cyclic_change_joker(number of changes), smooth(number of changes in a sequence of variables with respect to a binary constraint).

generalisation: change_pair(variable replaced by pair of variables), change_vectors(variable replaced by vector).

shift of concept: distance_change, longest_change.

Keywords

characteristic of a constraint: automaton, automaton with counters, non-deterministic automaton.

constraint arguments: pure functional dependency.

constraint network structure: sliding cyclic(1) constraint network(2), sliding cyclic(1) constraint network(3), Berge-acyclic constraint network.

constraint type: timetabling constraint.

filtering: dynamic programming.

final graph structure: acyclic, bipartite, no loop.

modelling: number of changes, functional dependency.
### Arc input(s)

VARIABLES

### Arc generator

\[ \text{PATH} \mapsto \text{collection}(\text{variables1}, \text{variables2}) \]

### Arc arity

2

### Arc constraint(s)

\[ \text{variables1}.\text{var} \text{CTR} \text{variables2}.\text{var} \]

### Graph property(ies)

\[ \text{NARC} = \text{NCHANGE} \]

### Graph class

- ACYCLIC
- BIPARTITE
- NO LOOP

### Graph model

Since we are only interested by the constraints linking two consecutive items of the collection VARIABLES we use \text{PATH} to generate the arcs of the initial graph.

Parts (A) and (B) of Figure 5.96 respectively show the initial and final graph of the first example of the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

![Graph Model](image)

**Figure 5.96: Initial and final graph of the change constraint**
Automaton

Figure 5.97 depicts a first automaton that only accepts all the solutions of the change constraint. This automaton uses a counter in order to record the number of satisfied constraints of the form \( \text{VAR}_i \text{CTR} \text{VAR}_{i+1} \) already encountered. To each pair of consecutive variables \((\text{VAR}_i, \text{VAR}_{i+1})\) of the collection \(\text{VARIABLES}\) corresponds a 0-1 signature variable \(S_i\). The following signature constraint links \(\text{VAR}_i, \text{VAR}_{i+1}\) and \(S_i; \text{VAR}_i \text{CTR} \text{VAR}_{i+1} \iff S_i\).

\[
\begin{align*}
\text{VAR}_i & \text{CTR} \text{VAR}_{i+1}, & \{ S_i : \text{NCHANGE}=C \} & \text{VAR}_i \text{ not} \text{CTR} \text{VAR}_{i+1} \\
\{C=C+1\} & \end{align*}
\]

Figure 5.97: Automaton (with counter) of the change constraint

Since the reformulation associated with the previous automaton is not Berge-acyclic, we now describe a second counter free automaton that also only accepts all the solutions of the change constraint. Without loss of generality, assume that the collection of variables \(\text{VARIABLES}\) contains at least two variables (i.e., \(|\text{VARIABLES}| \geq 2\)). Let \(n\) and \(D\) respectively denote the number of variables of the collection \(\text{VARIABLES}\), and the union of the domains of the variables of \(\text{VARIABLES}\). Clearly, the maximum number of changes (i.e., the number of times the constraint \(\text{VAR}_i \text{CTR} \text{VAR}_{i+1} \) \(1 \leq i < n\) holds) cannot exceed the quantity \(m = \min(n - 1, \text{NCHANGE})\). The \((m + 1) \cdot |D| + 2\) states of the automaton that only accepts all the solutions of the change constraint are defined in the following way:

- We have an initial state labelled by \(s_I\).
- We have \(m \cdot |D|\) intermediate states labelled by \(s_{ij}\) \((i \in D, j \in [0, m])\). The first subscript \(i\) of state \(s_{ij}\) corresponds to the value currently encountered. The second subscript \(j\) denotes the number of already encountered satisfied constraints of the form \(\text{VAR}_i \text{CTR} \text{VAR}_{i+1}\) from the initial state \(s_I\) to the state \(s_{ij}\).
- We have a final state labelled by \(s_F\).

Four classes of transitions are respectively defined in the following way:

1. There is a transition, labelled by \(i\) from the initial state \(s_I\) to the state \(s_{i0}\), \((i \in D)\).
2. There is a transition, labelled by \(j\), from every state \(s_{ij}\), \((i \in D, j \in [0, m])\), to the final state \(s_F\).
3. \( \forall i \in D, \forall j \in [0, m], \forall k \in D \cap \{ k | i \not\sim \text{CTR} \ k \} \) there is a transition labelled by \( k \) from \( s_{ij} \) to \( s_{kj} \) (i.e., the counter \( j \) does not change for values \( k \) such that constraint \( i \ \text{CTR} \ k \) does not hold).

4. \( \forall i \in D, \forall j \in [0, m - 1], \forall k \in D \setminus \{ k | i \not\sim \text{CTR} \ k \} \) there is a transition labelled by \( k \) from \( s_{ij} \) to \( s_{kj+1} \) (i.e., the counter \( j \) is incremented by +1 for values \( k \) such that constraint \( i \ \text{CTR} \ k \) holds).

We have \(|D|\) transitions of type 1, \(|D| \cdot (m + 1)\) transitions of type 2, and at least \(|D|^2 \cdot m\) transitions of types 3 and 4. Since the maximum value of \( m \) is equal to \( n - 1 \), in the worst case we have at least \(|D|^2 \cdot (n - 1)\) transitions. This leads to a worst case time complexity of \(O(|D|^2 \cdot n^2)\) if we use Pesant’s algorithm for filtering the regular constraint [286].

Figure 5.99 depicts the corresponding counter free non deterministic automaton associated with the change constraint under the hypothesis that (1) all variables of VARIABLES are assigned a value in \( \{0, 1, 2, 3\} \), (2) \(|\text{VARIABLES}|\) is equal to 4, and (3) \( \text{CTR} \) is equal to \( \neq \).
The sequence of variables \( \text{VAR}_1, \text{VAR}_2, \text{VAR}_3, \text{VAR}_4 \) \text{NCHANGE} is passed to the automaton.

Figure 5.99: Counter free non-deterministic automaton of the change(\text{NCHANGE}, (\text{VAR}_1, \text{VAR}_2, \text{VAR}_3, \text{VAR}_4), \neq) constraint assuming \( \text{VAR}_i \in [0, 3] \) (\( 1 \leq i \leq 3 \)), with initial state \( s_I \) and final state \( s_F \).
5.54 change_continuity

**DESCRIPTION**

**LINKS**

**GRAPH**

**AUTOMATON**

**Origin**

N. Beldiceanu

**Constraint**

change_continuity : (NB_PERIOD_CHANGE, NB_PERIOD_CONTINUITY, MIN_SIZE_CHANGE, MAX_SIZE_CHANGE, MIN_SIZE_CONTINUITY, MAX_SIZE_CONTINUITY, NB_CHANGE, NB_CONTINUITY, VARIABLES, CTR)

**Arguments**

NB_PERIOD_CHANGE : dvar
NB_PERIOD_CONTINUITY : dvar
MIN_SIZE_CHANGE : dvar
MAX_SIZE_CHANGE : dvar
MIN_SIZE_CONTINUITY : dvar
MAX_SIZE_CONTINUITY : dvar
NB_CHANGE : dvar
NB_CONTINUITY : dvar
VARIABLES : collection(var—dvar)
CTR : atom

**Restrictions**

NB_PERIOD_CHANGE ≥ 0
NB_PERIOD_CONTINUITY ≥ 0
MIN_SIZE_CHANGE ≥ 0
MAX_SIZE_CHANGE ≥ MIN_SIZE_CHANGE
MIN_SIZE_CONTINUITY ≥ 0
MAX_SIZE_CONTINUITY ≥ MIN_SIZE_CONTINUITY
NB_CHANGE ≥ 0
NB_CONTINUITY ≥ 0
required(VARIABLES, var)
CTR ∈ [=, ≠, <, ≥, >, ≤]
On the one hand a change is defined by the fact that constraint $\text{VARIABLES[i]}.\text{var CTR VARIABLES[i+1]}.\text{var}$ holds. On the other hand a continuity is defined by the fact that constraint $\text{VARIABLES[i]}.\text{var CTR VARIABLES[i+1]}.\text{var}$ does not hold.

A period of change on variables

$$\text{VARIABLES[i]}.\text{var}, \text{VARIABLES[i+1]}.\text{var}, \ldots, \text{VARIABLES[j]}.\text{var} \ (i < j)$$

is defined by the fact that all constraints $\text{VARIABLES[k]}.\text{var CTR VARIABLES[k+1]}.\text{var}$ hold for $k \in [i, j - 1]$.

A period of continuity on variables

$$\text{VARIABLES[i]}.\text{var}, \text{VARIABLES[i+1]}.\text{var}, \ldots, \text{VARIABLES[j]}.\text{var} \ (i < j)$$

is defined by the fact that all constraints $\text{VARIABLES[k]}.\text{var CTR VARIABLES[k+1]}.\text{var}$ do not hold for $k \in [i, j - 1]$.

The constraint change_continuity holds if and only if:

- $\text{NB}_\text{PERIOD}_\text{CHANGE}$ is equal to the number of periods of change,
- $\text{NB}_\text{PERIOD}_\text{CONTINUITY}$ is equal to the number of periods of continuity,
- $\text{MIN}_\text{SIZE}_\text{CHANGE}$ is equal to the number of variables of the smallest period of change,
- $\text{MAX}_\text{SIZE}_\text{CHANGE}$ is equal to the number of variables of the largest period of change,
- $\text{MIN}_\text{SIZE}_\text{CONTINUITY}$ is equal to the number of variables of the smallest period of continuity,
- $\text{MAX}_\text{SIZE}_\text{CONTINUITY}$ is equal to the number of variables of the largest period of continuity,
- $\text{NB}_\text{CHANGE}$ is equal to the total number of changes,
- $\text{NB}_\text{CONTINUITY}$ is equal to the total number of continuities.

**Example**

$$\begin{cases} \text{var} - 1, \\ \text{var} - 3, \\ \text{var} - 1, \\ \text{var} - 8, \\ \text{var} - 8, \\ \text{var} - 4, \end{cases} \neq \begin{cases} 3, 2, 2, 4, 2, 4, 6, 4, \\ \text{var} - 7, \\ \text{var} - 7, \\ \text{var} - 7, \\ \text{var} - 7, \\ \text{var} - 2 \end{cases}$$

Figure 5.100 makes clear the different parameters that are associated with the given example for the collection $\text{VARIABLES} = \{1, 3, 1, 8, 8, 4, 7, 7, 7, 2\}$. We place character $|$ for representing a change and a blank for a continuity. On top of the solution we represent the different periods of change, while below we show the different periods of continuity. The change_continuity constraint holds since:
• Its number of periods of change `NB_PERIOD_CHANGE` is equal to 3 (i.e., the 3 periods depicted on top of Figure 5.100),

• Its number of periods of continuity `NB_PERIOD_CONTINUITY` is equal to 2 (i.e., the 2 periods depicted below Figure 5.100),

• The number of variables of its smallest period of change `MIN_SIZE_CHANGE` is equal to 2 (i.e., the number of variables involved in the third period of change 7 2 depicted on top of Figure 5.100),

• The number of variables of the largest period of change `MAX_SIZE_CHANGE` is equal to 4 (i.e., the number of variables involved in the first period of change 1 3 1 8 depicted on top of Figure 5.100),

• The number of variables of the smallest period of continuity `MIN_SIZE_CONTINUITY` is equal to 2 (i.e., the number of variables involved in the first period 8 8 depicted below Figure 5.100),

• The number of variables of the largest period of continuity `MAX_SIZE_CONTINUITY` is equal to 4 (i.e., the number of variables involved in the second period 7 7 7 7 depicted below Figure 5.100),

• The total number of changes `NB_CHANGE` is equal to 6 (i.e., the number of occurrences of character | in Figure 5.100),

• The total number of continuities `NB_CONTINUITY` is equal to 4.

```
<-----> <----> <->
1|3|1|8 8|4|7 7|7|2
<->  <----->
```

Figure 5.100: Periods of changes and periods of continuities

**Typical**

- `NB_PERIOD_CHANGE` > 0
- `NB_PERIOD_CONTINUITY` > 0
- `MIN_SIZE_CHANGE` > 0
- `MIN_SIZE_CONTINUITY` > 0
- `NB_CHANGE` > 0
- `NB_CONTINUITY` > 0
- `|VARIABLES|` > 1
- `range(VARIABLES.var)` > 1
- `CTR` ∈ [≠]

**Symmetry**

One and the same constant can be added to the `var` attribute of all items of `VARIABLES`.
Arg. properties

- Functional dependency: NB_PERIOD_CHANGE determined by VARIABLES and CTR.
- Functional dependency: NB_PERIOD_CONTINUITY determined by VARIABLES and CTR.
- Functional dependency: MIN_SIZE_CHANGE determined by VARIABLES and CTR.
- Functional dependency: MAX_SIZE_CHANGE determined by VARIABLES and CTR.
- Functional dependency: MIN_SIZE_CONTINUITY determined by VARIABLES and CTR.
- Functional dependency: MAX_SIZE_CONTINUITY determined by VARIABLES and CTR.
- Functional dependency: NB_CHANGE determined by VARIABLES and CTR.
- Functional dependency: NB_CONTINUITY determined by VARIABLES and CTR.

Remark

If the variables of the collection VARIABLES have to take distinct values between 1 and the total number of variables, we have what is called a permutation. In this case, if we choose the binary constraint <, then MAX_SIZE_CHANGE gives the size of the longest run of the permutation; A run is a maximal increasing contiguous subsequence in a permutation.

See also

- common keyword: group, group_skip, isolated_item, stretch_path (timetabling constraint).

Keywords

- characteristic of a constraint: automaton, automaton with counters.
- combinatorial object: sequence, run of a permutation, permutation.
- constraint network structure: sliding cyclic(1) constraint network(2), sliding cyclic(1) constraint network(3).
- constraint type: timetabling constraint.
- final graph structure: connected component, partition, acyclic, bipartite, no loop.
- miscellaneous: obscure.
### Arc input(s)

VARIABLES

### Arc generator

\[ \text{PATH} \rightarrow \text{collection}(\text{variables1}, \text{variables2}) \]

### Arc arity

2

### Arc constraint(s)

\[ \text{variables1}.\text{var} \text{CTR} \text{variables2}.\text{var} \]

### Graph property(ies)

- \text{NCC} = \text{NB}_{\text{PERIOD}\_\text{CHANGE}}
- \text{MIN}_{\text{NCC}} = \text{MIN}_{\text{SIZE}\_\text{CHANGE}}
- \text{MAX}_{\text{NCC}} = \text{MAX}_{\text{SIZE}\_\text{CHANGE}}
- \text{NARC} = \text{NB}_{\text{CHANGE}}

### Graph class

- ACYCLIC
- BIPARTITE
- NO LOOP

---

### Arc input(s)

VARIABLES

### Arc generator

\[ \text{PATH} \rightarrow \text{collection}(\text{variables1}, \text{variables2}) \]

### Arc arity

2

### Arc constraint(s)

\[ \text{variables1}.\text{var} \sim \text{CTR} \text{variables2}.\text{var} \]

### Graph property(ies)

- \text{NCC} = \text{NB}_{\text{PERIOD}\_\text{CONTINUITY}}
- \text{MIN}_{\text{NCC}} = \text{MIN}_{\text{SIZE}\_\text{CONTINUITY}}
- \text{MAX}_{\text{NCC}} = \text{MAX}_{\text{SIZE}\_\text{CONTINUITY}}
- \text{NARC} = \text{NB}_{\text{CONTINUITY}}

### Graph class

- ACYCLIC
- BIPARTITE
- NO LOOP

---

### Graph model

We use two graph constraints to respectively catch the constraints on the period of changes and of the period of continuities. In both case each period corresponds to a connected component of the final graph.

Parts (A) and (B) of Figure 5.101 respectively show the initial and final graph associated with the first graph constraint of the Example slot.
Figure 5.101: Initial and final graph of the change_continuity constraint
Automaton

Figures 5.102, 5.103, 5.106, 5.110, 5.111, and 5.114 depict the automata associated with the different graph parameters of the change_continuity constraint. For the automata that respectively compute NB_PERIOD_CHANGE, NB_PERIOD_CONTINUITY MIN_SIZE_CHANGE, MIN_SIZE_CONTINUITY MAX_SIZE_CHANGE, MAX_SIZE_CONTINUITY NB_CHANGE and NB_CONTINUITY we have a 0-1 signature variable $S_i$ for each pair of consecutive variables $(VAR_i, VAR_{i+1})$ of the collection VARIABLES. The following signature constraint links $VAR_i$, $VAR_{i+1}$ and $S_i$: $VAR_i$ CTR $VAR_{i+1} \Leftrightarrow S_i$.

Figure 5.102: Automaton for the NB_PERIOD_CHANGE parameter of the change_continuity constraint
Figure 5.103: Automaton for the NB_PERIOD_CONTINUITY parameter of the change_continuity constraint

Figure 5.104: Hypergraph of the reformulation corresponding to the automaton of the NB_PERIOD_CHANGE parameter of the change_continuity constraint

Figure 5.105: Hypergraph of the reformulation corresponding to the automaton of the NB_PERIOD_CONTINUITY parameter of the change_continuity constraint
Figure 5.106: Automaton for the MIN_SIZE_CHANGE parameter of the change_continuity constraint

Figure 5.107: Automaton for the MIN_SIZE_CONTINUITY parameter of the change_continuity constraint

Figure 5.108: Hypergraph of the reformulation corresponding to the automaton of the MIN_SIZE_CHANGE parameter of the change_continuity constraint
Figure 5.109: Hypergraph of the reformulation corresponding to the automaton of the MIN_SIZE_CONTINUITY parameter of the change_continuity constraint

Figure 5.110: Automaton for the MAX_SIZE_CHANGE parameter of the change_continuity constraint

Figure 5.111: Automaton for the MAX_SIZE_CONTINUITY parameter of the change_continuity constraint
Figure 5.112: Hypergraph of the reformulation corresponding to the automaton of the \textsc{MAX\_SIZE\_CHANGE} parameter of the \textsc{change\_continuity} constraint

Figure 5.113: Hypergraph of the reformulation corresponding to the automaton of the \textsc{MAX\_SIZE\_CONTINUITY} parameter of the \textsc{change\_continuity} constraint

Figure 5.114: Automata for the \textsc{NB\_CHANGE} and \textsc{NB\_CONTINUITY} parameters of the \textsc{change\_continuity} constraint
Figure 5.115: Hypergraph of the reformulation corresponding to the automaton of the NB_CHANGE parameter of the change_continuity constraint

Figure 5.116: Hypergraph of the reformulation corresponding to the automaton of the NB_CONTINUITY parameter of the change_continuity constraint
5.55 change_pair

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from change.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>change_pair(NCHANGE, PAIRS, CTRX, CTRY)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Arguments   | NCHANGE : dvar  
PAIRS : collection(x−dvar,y−dvar)  
CTRX : atom  
CTRY : atom |       |           |
| Restrictions| NCHANGE ≥ 0  
NCHANGE < |PAIRS|  
required(PAIRS,[x,y])  
CTRX ∈ [=,≠,⟨,⟩]  
CTRY ∈ [=,≠,⟨,⟩] |       |           |
| Purpose     | NCHANGE is the number of times that the following disjunction holds: (X₁ CTRX X₂) ∨  
(Y₁ CTRY Y₂), where (X₁, Y₁) and (X₂, Y₂) correspond to consecutive pairs of variables  
of the collection PAIRS. |       |           |
| Example     | \[
\begin{pmatrix}
  x - 3 & y - 5, \\
  x - 3 & y - 7, \\
  x - 3 & y - 7, \\
  x - 3 & y - 8, \\
  x - 3 & y - 4, \\
  x - 3 & y - 7, \\
  x - 1 & y - 3, \\
  x - 1 & y - 6, \\
  x - 1 & y - 6, \\
  x - 3 & y - 7
\end{pmatrix}
\] \(, \neq, >\) |       |           |

In the example we have the following 3 changes:

- One change between pairs \(x - 3 y - 8\) and \(x - 3 y - 4\) since \(3 \neq 3 \lor 8 > 4\).
- One change between pairs \(x - 3 y - 7\) and \(x - 1 y - 3\) since \(3 \neq 1 \lor 7 > 3\).
- One change between pairs \(x - 1 y - 6\) and \(x - 3 y - 7\) since \(1 \neq 3 \lor 6 > 7\).

Consequently the change_pair constraint holds since its first argument NCHANGE is assigned value 3.

Typical

\[
\begin{align*}
NCHANGE & > 0 \\
|PAIRS| & > 1 \\
range(PAIRS.x) & > 1 \\
range(PAIRS.y) & > 1
\end{align*}
\]
Symmetries

- One and the same constant can be added to the x attribute of all items of PAIRS.
- One and the same constant can be added to the y attribute of all items of PAIRS.

Arg. properties

Functional dependency: NCHANGE determined by PAIRS, CTRX and CTRY.

Usage

Here is a typical example where this constraint is useful. Assume we have to produce a set of cables. A given quality and a given cross-section that respectively correspond to the x and y attributes of the previous pairs of variables characterise each cable. The problem is to sequence the different cables in order to minimise the number of times two consecutive wire cables $C_1$ and $C_2$ verify the following property: $C_1$ and $C_2$ do not have the same quality or the cross section of $C_1$ is greater than the cross section of $C_2$.

See also

generalisation: change_vectors (pair of variables replaced by vector).
specialisation: change (pair of variables replaced by variable).

Keywords

characteristic of a constraint: pair, automaton, automaton with counters.
constraint arguments: pure functional dependency.
constraint network structure: sliding cyclic(2) constraint network(2).
constraint type: timetabling constraint.
final graph structure: acyclic, bipartite, no loop.
modelling: number of changes, functional dependency.
Arc input(s) | PAIRS
---|---
Arc generator | \( \text{PATH} \rightarrow \text{collection}(\text{pairs1}, \text{pairs2}) \)
Arc arity | 2
Arc constraint(s) | \( \text{pairs1}.x \text{CTR} \text{pairs2}.x \lor \text{pairs1}.y \text{CTR} \text{pairs2}.y \)
Graph property(ies) | \( \text{NARC} = \text{NCHANGE} \)
Graph class | • ACYCLIC
| • BIPARTITE
| • NO LOOP

Graph model
Same as change, except that each item has two attributes \( x \) and \( y \).

Parts (A) and (B) of Figure 5.117 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.
Figure 5.117: Initial and final graph of the change_pair constraint
Automaton

Figure 5.118 depicts the automaton associated with the \texttt{change\_pair} constraint. To each pair of consecutive pairs \((X_i, Y_i), (X_{i+1}, Y_{i+1})\) of the collection \texttt{PAIRS} corresponds a 0-1 signature variable \(S_i\). The following signature constraint links \(X_i, Y_i, X_{i+1}, Y_{i+1}\) and \(S_i\):

\[
(X_i \text{ CTRX } X_{i+1}) \lor (Y_i \text{ CTRY } Y_{i+1}) \iff S_i.
\]

\[\begin{array}{c}
\{C=0\} \\
\{C=C+1\}
\end{array}\]

\[
(X_i \text{ not CTRX } X_{i+1}) \land (Y_i \text{ not CTRY } Y_{i+1})
\]

\[\text{NCHANGE}\]

Figure 5.118: Automaton of the \texttt{change\_pair} constraint

Figure 5.119: Hypergraph of the reformulation corresponding to the automaton of the \texttt{change\_pair} constraint
5.56  change_partition

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
</table>

### Origin
Derived from change.

### Constraint
`change_partition(NCHANGE, VARIABLES, PARTITIONS)`

### Type
```
VALUES : collection(val=int)
```

### Arguments
```
NCHANGE : dvar
VARIABLES : collection(var=dvar)
PARTITIONS : collection(p=VALUES)
```

### Restrictions
- `|VALUES| ≥ 1`
- `required(VALUE, val)`
- `distinct(VALUE, val)`
- `NCHANGE ≥ 0`
- `NCHANGE < |VARIABLES|`
- `required(VALUE, var)`
- `required(PARTITIONS, p)`
- `|PARTITIONS| ≥ 2`

### Purpose
`NCHANGE` is the number of times that the following constraint holds: `X` and `Y` do not belong to the same partition of the collection `PARTITIONS`, where `X` and `Y` correspond to consecutive variables of the collection `VARIABLES`.

### Example
```

\[
\begin{align*}
\var - 6, \\
\var - 6, \\
\var - 2, \\
\var - 1, \\
\var - 3, \\
\var - 3, \\
\var - 1, \\
\var - 6, \\
\var - 2, \\
\var - 2, \\
\var - 2, \\
p = \{1, 3\}, \\
p = \{4\}, \\
p = \{2, 6\}
\end{align*}
\]
```

In the example we have the following two changes:

- One change between values 2 and 1 (since 2 and 1 respectively belong to the third and the first partition).
- One change between values 1 and 6 (since 1 and 6 respectively belong to the first and the third partition).
Consequently the change_partition constraint holds since its first argument NCHANGE is assigned to 2.

### Typical

| NCHANGE > 0 |
| [VARIABLES] > 1 |
| range(VARIABLES.var) > 1 |
| [VARIABLES] > |PARTITIONS|

### Symmetries

- Items of VARIABLES can be reversed.
- Items of PARTITIONS are permutable.
- Items of PARTITIONS p are permutable.
- An occurrence of a value of VARIABLES.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.

### Arg. properties

**Functional dependency:** NCHANGE determined by VARIABLES and PARTITIONS.

### Usage

This constraint is useful for the following problem: Assume you have to produce a set of orders, each order belonging to a given family. In the context of the Example slot we have three families that respectively correspond to values 1, 3, to value 4 and to values 2, 6. We would like to sequence the orders in such a way that we minimise the number of times two consecutive orders do not belong to the same family.

### Algorithm

[29].

### See also

- **common keyword:** change (number of changes in a sequence of variables with respect to a binary constraint).
- **used in graph description:** in_same_partition.

### Keywords

- **characteristic of a constraint:** partition.
- **constraint arguments:** pure functional dependency.
- **constraint type:** timetabling constraint.
- **final graph structure:** acyclic, bipartite, no loop.
- **modelling:** number of changes, functional dependency.
Arc input(s) | VARIABLES
---|---
Arc generator | $PATH \mapsto \text{collection}(\text{variables1}, \text{variables2})$
Arc arity | 2
Arc constraint(s) | $\text{in\_same\_partition}(\text{variables1.var}, \text{variables2.var}, \text{PARTITIONS})$
Graph property(ies) | $\text{NARC} = \text{NCHANGE}$
Graph class | • ACYCLIC
| • BIPARTITE
| • NO LOOP

Graph model

Parts (A) and (B) of Figure 5.120 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.
Figure 5.120: Initial and final graph of the change_partition constraint
5.57 change_vectors

**DESCRIPTION**

**Origin**
Derived from change

**Constraint**
change_vectors(NCHANGE, VECTORS, CTRS)

**Types**
VECTOR : collection(var − dvar)
CTR : atom

**Arguments**
NCHANGE : dvar
VECTORS : collection(vec − VECTOR)
CTRS : collection(ctr − CTR)

**Restrictions**
|VECTOR| ≥ 1
required(VECTOR, var)
CTR ∈ [=, ≠, <, ≥, >, ≤]
NCHANGE ≥ 0
NCHANGE < |VECTORS|
required(VECTORS, vec)
same size(VECTORS, vec)
required(CTRS, ctr)
|CTRS| = |VECTOR|

**Purpose**
Let us note VECTOR₁, VECTOR₂, …, VECTORₙ the vectors of the VECTORS collection, and d the number of components of each vector (all vectors have the same size). NCHANGE is the number of times that the following disjunctions holds where i ∈ [1, n − 1]


... ... ...

(VECTORᵢ, vecᵢ[d] CTRSᵢ[d] VECTORᵢ₊₁.vecᵢ[d]).

**Example**

\[
\begin{pmatrix}
\text{vec} - (4, 0), \\
\text{vec} - (4, 0), \\
\text{vec} - (4, 5), \\
3, \text{ vec} - (3, 4), \\
\text{vec} - (3, 4), \\
\text{vec} - (3, 4), \\
\text{vec} - (4, 0)
\end{pmatrix},
\]

In the example we have the following 3 changes:

- One change between (4, 0) and (4, 5) since 4 ≠ 4 ∨ 0 ≠ 5,
- One change between (4, 5) and (3, 4) since 4 ≠ 3 ∨ 5 ≠ 4,
- One change between (3, 4) and (4, 0) since 3 ≠ 4 ∨ 4 ≠ 0.
Consequently the `change_vectors` constraint holds since its first argument `NCHANGE` is assigned value 3.

**Typical**

- $\text{CTR} \in \{\neq\}$
- $|\text{VECTOR}| > 1$
- $NCHANGE > 0$
- $|\text{VECTORS}| > 1$

**Arg. properties**

- Functional dependency: `NCHANGE` determined by `VECTORS` and `CTRS`.

**See also**

- Specialisation: `change` (*vector replaced by variable*), `change_pair` (*vector replaced by pair of variables*).

**Keywords**

- **Characteristic of a constraint**: automaton, automaton with counters, vector.
- **Constraint arguments**: pure functional dependency.
- **Constraint network structure**: Berge-acyclic constraint network.
- **Modelling**: number of changes, functional dependency.
### 5.58 circuit

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>[238]</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>circuit(NODES)</td>
<td></td>
</tr>
<tr>
<td><strong>Synonyms</strong></td>
<td>atour, cycle.</td>
<td></td>
</tr>
<tr>
<td><strong>Argument</strong></td>
<td>NODES : collection(index−int, succ−dvar)</td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>required(NODES,[index, succ])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.index ≥ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.index ≤</td>
<td>NODES</td>
</tr>
<tr>
<td></td>
<td>distinct(NODES, index)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.succ ≥ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.succ ≤</td>
<td>NODES</td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>Enforce to cover a digraph G described by the NODES collection with one circuit visiting once all vertices of G.</td>
<td></td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Typical</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Symmetry</strong></td>
<td>Items of NODES are permutable.</td>
<td></td>
</tr>
<tr>
<td><strong>Remark</strong></td>
<td>In the original circuit constraint of CHIP the index attribute was not explicitly present. It was implicitly defined as the position of a variable in a list.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Within the context of linear programming [4] this constraint was introduced under the name atour. In the same context [198, page 380] provides continuous relaxations of the circuit constraint.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Within the KOALOG constraint system this constraint is called cycle.</td>
<td></td>
</tr>
<tr>
<td><strong>Algorithm</strong></td>
<td>Since all succ variables of the NODES collection have to take distinct values one can reuse the algorithms associated with the alldifferent constraint. A second necessary condition is to have no more than one strongly connected component. Pruning for enforcing this condition can be done by forcing all strong bridges to belong to the final solution, since otherwise the strongly connected component would be broken apart. A third necessary condition is that, if the graph is bipartite then the number of vertices of each class</td>
<td></td>
</tr>
</tbody>
</table>
should be identical. Consequently if the number of vertices is odd (i.e., \(|V|\) is odd) the graph should not be bipartite. Further necessary conditions (useful when the graph is sparse) combining the fact that we have a perfect matching and one single strongly connected component can be found in [360]. These conditions forget about the orientation of the arcs of the graph and characterise new required elementary chains. A typical pattern involving four vertices is depicted by Figure 5.121 where we assume that:

- There is an elementary chain between \(c\) and \(d\) (depicted by a dashed edge),
- \(b\) has exactly 3 neighbours.

In this context the edge between \(a\) and \(b\) is mandatory in any covering (i.e., the arc from \(a\) to \(b\) or the arc from \(b\) to \(a\)) since otherwise a small circuit involving \(b\), \(c\) and \(d\) would be created.

Figure 5.121: Reasoning about elementary chains and degrees: if we have an elementary chain between \(c\) and \(d\) and if \(b\) has 3 neighbours then the edge \((a, b)\) is mandatory.

When the graph is planar [200][127] one can also use as a necessary condition discovered by Grinberg [184] for pruning.

Finally, another approach based on the notion of 1-toughness [110] was proposed in [218] and evaluated for small graphs (i.e., graphs with up to 15 vertices).

**Systems**

- circuit in Gecode, circuit in JaCoP, circuit in SICStus.

**See also**

- common keyword: alldifferent (permutation), circuit_cluster (graph constraint, one_succ), path (graph partitioning constraint, one succ), tour (graph partitioning constraint, Hamiltonian).

**generalisation:** cycle (introduce a variable for the number of circuits).

**implies:** alldifferent.

**implies (items to collection):** lex_alldifferent.

**related:** strongly_connected.

**Keywords**

- combinatorial object: permutation.
- constraint type: graph constraint, graph partitioning constraint.
- filtering: linear programming, planarity test, strong bridge, DFS-bottleneck.
- final graph structure: circuit, one_succ.
- problems: Hamiltonian.
Arc input(s) \( \text{NODES} \)

Arc generator \( \text{CLIQUE} \mapsto \text{collection}(\text{nodes}_1, \text{nodes}_2) \)

Arc arity 2

Arc constraint(s) \( \text{nodes}_1.\text{succ} = \text{nodes}_2.\text{index} \)

Graph property(ies)
- \( \text{MIN}_\text{NSCC} = |\text{NODES}| \)
- \( \text{MAX}_\text{ID} \leq 1 \)

Graph class \( \text{ONE}_\text{SUCC} \)

Graph model

The first graph property enforces to have one single strongly connected component containing \( |\text{NODES}| \) vertices. The second graph property imposes to only have circuits. Since each vertex of the final graph has only one successor we do not need to use set variables for representing the successors of a vertex.

Parts (A) and (B) of Figure 5.122 respectively show the initial and final graph associated with the Example slot. The circuit constraint holds since the final graph consists of one circuit mentioning once every vertex of the initial graph.

![Graph Diagram](A)

![Graph Diagram](B)

Figure 5.122: Initial and final graph of the circuit constraint
5.59 circuit_cluster

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inspired by [234].</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>circuit_cluster(NCIRCUIT, NODES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NCIRCUIT : dvar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NODES : collection(index-int, cluster-int, succ-dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NCIRCUIT ≥ 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NCIRCUIT ≤</td>
<td>NODES</td>
<td></td>
</tr>
<tr>
<td>required(NODES[index, cluster, succ])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NODES.index ≥ 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NODES.index ≤</td>
<td></td>
<td>NODES</td>
</tr>
<tr>
<td>distinct(NODES, index)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NODES.succ ≥ 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NODES.succ ≤</td>
<td></td>
<td>NODES</td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Consider a digraph $G$, described by the NODES collection, such that its vertices are partitioned among several clusters. NCIRCUIT is the number of circuits containing more than one vertex used for covering $G$ in such a way that each cluster is visited by exactly one circuit of length greater than 1.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example

\[
\begin{align*}
1, & \{ \text{index} - 1, \text{cluster} - 1, \text{succ} - 1, \\
  & \text{index} - 2, \text{cluster} - 1, \text{succ} - 4, \\
  & \text{index} - 3, \text{cluster} - 2, \text{succ} - 3, \\
  & \text{index} - 4, \text{cluster} - 2, \text{succ} - 5, \\
  & \text{index} - 5, \text{cluster} - 3, \text{succ} - 8, \\
  & \text{index} - 6, \text{cluster} - 3, \text{succ} - 6, \\
  & \text{index} - 7, \text{cluster} - 3, \text{succ} - 7, \\
  & \text{index} - 8, \text{cluster} - 4, \text{succ} - 2, \\
  & \text{index} - 9, \text{cluster} - 4, \text{succ} - 9, \\
  & \text{index} - 1, \text{cluster} - 1, \text{succ} - 1, \\
  & \text{index} - 2, \text{cluster} - 1, \text{succ} - 4, \\
  & \text{index} - 3, \text{cluster} - 2, \text{succ} - 3, \\
  & \text{index} - 4, \text{cluster} - 2, \text{succ} - 2, \\
  & \text{index} - 5, \text{cluster} - 3, \text{succ} - 5, \\
  & \text{index} - 6, \text{cluster} - 3, \text{succ} - 9, \\
  & \text{index} - 7, \text{cluster} - 3, \text{succ} - 7, \\
  & \text{index} - 8, \text{cluster} - 4, \text{succ} - 8, \\
  & \text{index} - 9, \text{cluster} - 4, \text{succ} - 6 \}
\]

2.

Both examples involve 9 vertices 1, 2, \ldots, 9 such that vertices 1 and 2 belong to cluster number 1, vertices 3 and 4 belong to cluster number 2, vertices 5, 6 and 7 belong to cluster number 3, and vertices 8 and 9 belong to cluster number 4.
The first example involves only one single circuit containing more than one vertex (i.e., see in Figure 5.123 the circuit $2 \rightarrow 4 \rightarrow 5 \rightarrow 8 \rightarrow 2$). The corresponding circuit_cluster constraint holds since exactly one vertex of each cluster (i.e., vertex 2 for cluster 1, vertex 4 for cluster 2, vertex 5 for cluster 3, vertex 8 for cluster 4) belongs to this circuit.

The second example contains the two circuits $2 \rightarrow 4 \rightarrow 2$ and $6 \rightarrow 9 \rightarrow 6$ that both involve more than one vertex. The corresponding circuit_cluster constraint holds since exactly one vertex of each cluster (i.e., see in Figure 5.124 vertex 2 in $2 \rightarrow 4 \rightarrow 2$ for cluster 1, vertex 4 in $2 \rightarrow 4 \rightarrow 2$ for cluster 2, vertex 6 in $6 \rightarrow 9 \rightarrow 6$ for cluster 3, vertex 9 in $6 \rightarrow 9 \rightarrow 6$ for cluster 4) belongs to these two circuits.

**Typical**

<table>
<thead>
<tr>
<th>NCIRCUIT $&lt;$</th>
<th></th>
<th>NODES</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td>$</td>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td>range(NODES.cluster) $&gt;$</td>
<td>$1$</td>
<td></td>
</tr>
</tbody>
</table>

**Symmetry**

Items of NODES are permutable.

**Usage**

A related abstraction in Operations Research was introduced in [234]. It was reported as the Generalised Travelling Salesman Problem (GTSP). The circuit_cluster constraint differs from the GTSP because of the two following points:

- Each node of our graph belongs to one single cluster,
- We do not constrain the number of circuits to be equal to 1: The number of circuits should be equal to one of the values of the domain of the variable NCIRCUIT.

**See also**

common keyword: alldifferent(permutation), circuit, cycle(graph constraint, one_succ).

used in graph description: alldifferent, nvalues.

**Keywords**

combinatorial object: permutation.

constraint type: graph constraint.

final graph structure: strongly connected component, one_succ.

modelling: cluster.
Figure 5.123: Four clusters and a covering with one circuit corresponding to the first example

Figure 5.124: The same clusters as in the first example and a covering with two circuits corresponding to the second example
Arc input(s) NODES
Arc generator $CLIQUE \rightarrow \text{collection}(\text{nodes1},\text{nodes2})$
Arc arity 2
Arc constraint(s)
- nodes1.succ $\neq$ nodes1.index
- nodes1.succ $=$ nodes2.index
Graph property(ies)
- NTREE $= 0$
- NSCC $= \text{NCIRCUIT}$
Graph class ONE_SUCC
Sets
ALL_VERTICES $\mapsto$
- \[
\text{variables} - \text{col}\left(\text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \text{item}(\text{var} - \text{NODES.cluster})\right)
\]
Constraint(s) on sets
- \text{alldifferent}(\text{variables})
- \text{nvalues}(\text{variables}, =, \text{size}(\text{NODES.cluster}))

Graph model
In order to express the binary constraint linking two vertices one has to make explicit the identifier of each vertex as well as the cluster to which belongs each vertex. This is why the circuit cluster constraint considers objects that have the following three attributes:
- The attribute index that is the identifier of a vertex.
- The attribute cluster that is the cluster to which belongs a vertex.
- The attribute succ that is the unique successor of a vertex.

The partitioning of the clusters by different circuits is expressed in the following way:
- First note the condition nodes1.succ $\neq$ nodes1.index prevents the final graph of containing any loop. Moreover the condition nodes1.succ $=$ nodes2.index imposes no more than one successor for each vertex of the final graph.
- The graph property NTREE $= 0$ enforces that all vertices of the final graph belong to one circuit.
- The graph property NSCC $= \text{NCIRCUIT}$ express the fact that the number of strongly connected components of the final graph is equal to NCIRCUIT.
- The constraint alldifferent(variables) on the set ALL_VERTICES (i.e., all the vertices of the final graph) states that the cluster attributes of the vertices of the final graph should be pairwise distinct. This concretely means that no cluster should be visited more than once.
- The constraint nvalues(variables, =, size(NODES.cluster)) on the set ALL_VERTICES conveys the fact that the number of distinct values of the cluster attribute of the vertices of the final graph should be equal to the total number of clusters. This implies that each cluster is visited at least one time.

Parts (A) and (B) of Figure 5.125 respectively show the initial and final graph associated with the second example of the Example slot. Since we use the NSCC graph property, we show the two strongly connected components of the final graph. They respectively correspond to the two circuits $2 \rightarrow 4 \rightarrow 2$ and $6 \rightarrow 9 \rightarrow 6$. Since all the vertices belongs to a circuit we have that NTREE $= 0$. 
Figure 5.125: Initial and final graph of the circuit cluster constraint
5.60  circular_change

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Derived from change.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Constraint**

```
circular_change(NCHANGE, VARIABLES, CTR)
```

**Arguments**

- **NCHANGE**: dvar
- **VARIABLES**: collection(var−dvar)
- **CTR**: atom

**Restrictions**

- \( NCHANGE \geq 0 \)
- \( NCHANGE \leq |VARIABLES| \)
- \( required(VARIABLES, var) \)
- \( CTR \in [\neq, <, >, \leq] \)

**Purpose**

\( NCHANGE \) is the number of times that \( CTR \) holds on consecutive variables of the collection \( VARIABLES \). The last and the first variables of the collection \( VARIABLES \) are also considered to be consecutive.

**Example**

\((4, \langle 4, 4, 3, 4, 1 \rangle, \neq)\)

In the example the changes within the \( VARIABLES = \langle 4, 4, 3, 4, 1 \rangle \) collection are located between values 4 and 3, 3 and 4, 4 and 1, and 1 and 4 (i.e., since the third argument \( CTR \) of the circular_change constraint is set to \( \neq \), we count one change for each disequality constraint between two consecutive variables that holds). Consequently, the corresponding circular_change constraint holds since its first argument \( NCHANGE \) is fixed to 4.

**Typical**

- \( NCHANGE > 0 \)
- \( |VARIABLES| > 1 \)
- \( \text{range}(VARIABLES.var) > 1 \)
- \( CTR \in [\neq] \)

**Symmetries**

- Items of \( VARIABLES \) can be shifted.
- One and the same constant can be added to the \( var \) attribute of all items of \( VARIABLES \).

**Arg. properties**

- Functional dependency: \( NCHANGE \) determined by \( VARIABLES \) and \( CTR \).

**See also**

- **common keyword**: change (number of changes).

**Keywords**

- characteristic of a constraint: cyclic, automaton, automaton with counters.
- constraint arguments: pure functional dependency.
- constraint network structure: circular sliding cyclic(1) constraint network(2).
- constraint type: timetabling constraint.
- modelling: number of changes, functional dependency.
Graph model

Since we are also interested in the constraint that links the last and the first variable we use the arc generator *CIRCUIT* to produce the arcs of the initial graph.

Parts (A) and (B) of Figure 5.126 respectively show the initial and final graph associated with the Example slot. Since we use the *NARC* graph property, the arcs of the final graph are stressed in bold.

![Graphs showing initial and final graph of the circular_change constraint](attachment:image.png)
Automaton

Figure 5.127 depicts the automaton associated with the circular_change constraint. To each pair of consecutive variables \( (\text{VAR}_i, \text{VAR}_{(i \mod |\text{VARIABLES}|)+1}) \) of the collection \( \text{VARIABLES} \) corresponds a 0-1 signature variable \( S_i \). The following signature constraint links \( \text{VAR}_i, \text{VAR}_{(i \mod |\text{VARIABLES}|)+1} \) and \( S_i \): \( \text{VAR}_i \text{CTR VAR}_{(i \mod |\text{VARIABLES}|)+1} \Leftrightarrow S_i \).

Figure 5.127: Automaton of the circular_change constraint

Figure 5.128: Hypergraph of the reformulation corresponding to the automaton of the circular_change constraint
5.61 clause_and

### DESCRIPTION

**Origin**
Logic

**Constraint**
clause_and(POSVARS, NEGVARS, VAR)

**Synonym**
clause.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSVARS</td>
<td>collection(var−dvar)</td>
</tr>
<tr>
<td>NEGVARS</td>
<td>collection(var−dvar)</td>
</tr>
<tr>
<td>VAR</td>
<td>dvar</td>
</tr>
</tbody>
</table>

**Restrictions**

- \(|\text{POSVARS}| + |\text{NEGVARS}| > 0\)
- \(\text{required}(\text{POSVARS}, \text{var})\)
- \(\text{POSVARS}.\text{var} \geq 0\)
- \(\text{POSVARS}.\text{var} \leq 1\)
- \(\text{required}(\text{NEGVARS}, \text{var})\)
- \(\text{NEGVARS}.\text{var} \geq 0\)
- \(\text{NEGVARS}.\text{var} \leq 1\)
- \(\text{VAR} \geq 0\)
- \(\text{VAR} \leq 1\)

**Purpose**

Given a first collection of 0-1 variables \(\text{POSVARS} = U_1, U_2, \ldots, U_p\), a second collection of 0-1 variables \(\text{NEGVARS} = V_1, V_2, \ldots, V_n\), and a variable \(\text{VAR}\), enforce \(\text{VAR} = (U_1 \land U_2 \land \ldots \land U_p) \land (\neg V_1 \land \neg V_2 \land \ldots \land \neg V_n)\).

**Example**

\(((1,0),(0),0)\)

**Typical**

\(|\text{POSVARS}| + |\text{NEGVARS}| > 1\)

**Symmetries**

- Items of POSVARS are permutable.
- Items of NEGVARS are permutable.

**Arg. properties**

- Extensible wrt. \(\text{POSVARS}\) when \(\text{VAR} = 0\).
- Extensible wrt. \(\text{NEGVARS}\) when \(\text{VAR} = 0\).

**Remark**
The clause_or constraint is called clause in Gecode (http://www.gecode.org/).

**Systems**

reifiedAnd in Choco, clause in Choco, clause in Gecode.

**See also**

common keyword: and, clause_or (Boolean constraint).
Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint,

constraint network structure: Berge-acyclic constraint network.

constraint type: Boolean constraint.

filtering: arc-consistency.
Figure 5.129 depicts the automaton associated with the clause and constraint:

- To the argument \( \text{VAR} \) of the clause and constraint corresponds the first signature variable.
- To each variable of the argument \( \text{POSVARS} \) corresponds a signature variable.
- Finally, to each variable \( \text{VAR}_i \) of the argument \( \text{NEGVARS} \) corresponds a signature variable that is the negation of \( \text{VAR}_i \).

\[
\begin{align*}
\text{VAR} = 0 & \quad \text{PVAR}_i = 1 \\
\text{VAR} = 1 & \quad \text{NVAR}_i = 0 \\
\text{PVAR}_i = 1 & \quad \text{PVAR}_i = 0 \\
\text{NVAR}_i = 0 & \quad \text{NVAR}_i = 1
\end{align*}
\]

Figure 5.129: Automaton of the clause and constraint (PVAR\(_i\) and NVAR\(_i\) respectively denote variables of POSVARS and NEGVARS)

Figure 5.130: Hypergraph of the reformulation corresponding to the automaton of the clause and constraint
### 5.62 clause_or

**Origin**
Logic

**Constraint**
clause_or(POSVARS, NEGVARS, VAR)

**Synonym**
clause.

**Arguments**
- **POSVARS** : collection(var−dvar)
- **NEGVARS** : collection(var−dvar)
- **VAR** : dvar

**Restrictions**
\[
|\text{POSVARS}| + |\text{NEGVARS}| > 0 \\
|\text{POSVARS}.var| \geq 0 \\
|\text{POSVARS}.var| \leq 1 \\
|\text{NEGVARS}.var| \geq 0 \\
|\text{NEGVARS}.var| \leq 1 \\
\text{VAR} \geq 0 \\
\text{VAR} \leq 1
\]

**Purpose**
Given a first collection of 0-1 variables \(\text{POSVARS} = U_1, U_2, \ldots, U_p\), a second collection of 0-1 variables \(\text{NEGVARS} = V_1, V_2, \ldots, V_n\), and a variable \(\text{VAR}\), enforce \(\text{VAR} = (U_1 \lor U_2 \lor \ldots \lor U_p) \lor (\neg V_1 \lor \neg V_2 \lor \ldots \lor \neg V_n)\).

**Example**
\((0, 0), (0), 1\)

**Typical**
\( |\text{POSVARS}| + |\text{NEGVARS}| > 1 \)

**Symmetries**
- Items of \(\text{POSVARS}\) are permutable.
- Items of \(\text{NEGVARS}\) are permutable.

**Arg. properties**
- Extensible wrt. \(\text{POSVARS}\) when \(\text{VAR} = 1\).
- Extensible wrt. \(\text{NEGVARS}\) when \(\text{VAR} = 1\).

**Remark**
The `clause_or` constraint is called `clause` in Gecode (http://www.gecode.org/).

**Systems**
reifiedOr in Choco, clause in Choco, clause in Gecode.

**See also**
common keyword: clause_and, or (Boolean constraint).
Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

costRAINT network structure: Berge-acyclic constraint network.

costRAINT type: Boolean constraint.

filtering: arc-consistency.

modelling: disjunction.
Automaton

Figure 5.131 depicts the automaton associated with the `clause_or` constraint:

- To the argument `VAR` of the `clause_or` constraint corresponds the first signature variable.
- To each variable of the argument `POSVARS` corresponds a signature variable.
- Finally, to each variable `VAR_i` of the argument `NEGVARS` corresponds a signature variable that is the negation of `VAR_i`.

![Automaton Diagram](image)

Figure 5.131: Automaton of the `clause_or` constraint (PVAR_i and NVAR_i respectively denote variables of POSVARS and NEGVARS)

![Hypergraph Diagram](image)

Figure 5.132: Hypergraph of the reformulation corresponding to the automaton of the `clause_or` constraint
5.63 clique

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[146]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>clique(SIZE_CLIQUE, NODES)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>SIZE_CLIQUE : dvar</td>
<td>NODES : collection(index−int, succ−svar)</td>
</tr>
<tr>
<td>Restrictions</td>
<td>SIZE_CLIQUE ≥ 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SIZE_CLIQUE ≤</td>
<td>NODES</td>
</tr>
<tr>
<td></td>
<td>required(NODES,[index, succ])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.index ≥ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.index ≤</td>
<td>NODES</td>
</tr>
<tr>
<td></td>
<td>distinct(NODES,index)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.succ ≥ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.succ ≤</td>
<td>NODES</td>
</tr>
</tbody>
</table>

**Purpose**

Consider a digraph G described by the NODES collection: to the i^{th} item of the NODES collection corresponds the i^{th} vertex of G; To each value j of the i^{th} succ variable corresponds an arc from the i^{th} vertex to the j^{th} vertex. Select a subset S of the vertices of G that forms a clique of size SIZE_CLIQUE (i.e., there is an arc between each pair of distinct vertices of S).

**Example**

\[
\begin{pmatrix}
\text{index} - 1 & \text{succ} - \emptyset, \\
\text{index} - 2 & \text{succ} - \{3, 5\}, \\
\text{index} - 3 & \text{succ} - \{2, 5\}, \\
\text{index} - 4 & \text{succ} - \emptyset, \\
\text{index} - 5 & \text{succ} - \{2, 3\}
\end{pmatrix}
\]

The clique constraint holds since the NODES collection depicts a clique involving 3 vertices (namely vertices 2, 3 and 5) and since its first argument SIZE_CLIQUE is set to the number of vertices of this clique.

**Typical**

\[
\begin{align*}
\text{SIZE_CLIQUE} & \geq 2 \\
\text{SIZE_CLIQUE} & < |\text{NODES}| \\
|\text{NODES}| & > 2
\end{align*}
\]

**Symmetry**

Items of NODES are permutable.

**Arg. properties**

Functional dependency: SIZE_CLIQUE determined by NODES.

**Algorithm**

[146], [327], [328]. The algorithm for finding maximum cliques in an undirected graph of C. Bron and J. Kerbosch [83] was adapted by J.-C. Régin to the context of constraint programming in his papers.
See also

common keyword: link_set_to_booleans (constraint involving set variables, can be used for channelling).
used in graph description: in_set.

Keywords

constraint arguments: constraint involving set variables.
constraint type: graph constraint.
final graph structure: symmetric.
modelling: functional dependency.
problems: maximum clique.
Arc input(s) NODES
Arc generator \textit{CLIQUE}(\#) \rightarrow \textit{collection}(\text{nodes1, nodes2})
Arc arity 2
Arc constraint(s) \textit{in_set}(\text{nodes2.index, nodes1.succ})
Graph property(ies)
\begin{itemize}
  \item \text{NARC} = \text{SIZE}_{\text{CLIQUE}} \times \text{SIZE}_{\text{CLIQUE}} - \text{SIZE}_{\text{CLIQUE}}
  \item \text{NVERTEX} = \text{SIZE}_{\text{CLIQUE}}
\end{itemize}
Graph class SYMMETRIC

Graph model
Note the use of \textit{set variables} for modelling the fact that the vertices of the final graph have more than one successor: The successor variable associated with each vertex contains the successors of the corresponding vertex.

Part (A) of Figure 5.133 shows the initial graph from which we start. It is derived from the set associated with each vertex. Each set describes the potential values of the \textit{succ} attribute of a given vertex. Part (B) of Figure 5.133 gives the final graph associated with the \textbf{Example} slot. Since we both use the \text{NARC} and \text{NVERTEX} graph properties, the arcs and the vertices of the final graph are stressed in bold. The final graph corresponds to a clique containing three vertices.

Figure 5.133: Initial and final graph of the \textit{clique} set constraint
## 5.64 colored_matrix

### Description

**Origin**
KOALOG

**Constraint**

\[
\text{colored_matrix}(C, L, K, \text{MATRX}, \text{CPROJ}, \text{LPROJ})
\]

**Synonyms**

coloured_matrix, cardinality_matrix, card_matrix.

### Arguments

- \( C \) : int
- \( L \) : int
- \( K \) : int
- \( \text{MATRX} \) : collection(column−int, line−int, var−dvar)
- \( \text{CPROJ} \) : collection(column−int, val−int, nocc−dvar)
- \( \text{LPROJ} \) : collection(line−int, val−int, nocc−dvar)

### Restrictions

- \( C \geq 0 \)
- \( L \geq 0 \)
- \( K \geq 0 \)
- \( \text{required}(\text{MATRX}, [\text{column}, \text{line}, \text{var}]) \)
- \( \text{increasing_seq}(\text{MATRX}, \text{[column, line]}) \)
- \( |\text{MATRX}| = C \cdot L + C + L + 1 \)
- \( \text{MATRX.column} \geq 0 \)
- \( \text{MATRX.column} \leq C \)
- \( \text{MATRX.line} \geq 0 \)
- \( \text{MATRX.line} \leq L \)
- \( \text{MATRX.var} \geq 0 \)
- \( \text{MATRX.var} \leq K \)
- \( \text{required}(\text{CPROJ}, [\text{column}, \text{val}, \text{nocc}]) \)
- \( \text{increasing_seq}(\text{CPROJ}, \text{[column, val]}) \)
- \( |\text{CPROJ}| = C \cdot K + C + K + 1 \)
- \( \text{CPROJ.column} \geq 0 \)
- \( \text{CPROJ.column} \leq C \)
- \( \text{CPROJ.val} \geq 0 \)
- \( \text{CPROJ.val} \leq K \)
- \( \text{required}(\text{LPROJ}, [\text{line}, \text{val}, \text{nocc}]) \)
- \( \text{increasing_seq}(\text{LPROJ}, \text{[line, val]}) \)
- \( |\text{LPROJ}| = L \cdot K + L + K + 1 \)
- \( \text{LPROJ.line} \geq 0 \)
- \( \text{LPROJ.line} \leq L \)
- \( \text{LPROJ.val} \geq 0 \)
- \( \text{LPROJ.val} \leq K \)

### Purpose

Given a matrix of domain variables, imposes a **global_cardinality** constraint involving cardinality variables on each column and each row of the matrix.
The filtering algorithm described in [330] is based on network flow and does not achieve arc-consistency in general. However, when the number of values is restricted to two, the algorithm [330] achieves arc-consistency on the variables of the matrix. This corresponds in fact to a generalisation of the problem called "Matrices composed of 0's and 1's" presented by Ford and Fulkerson [209].
See also
common keyword: `k_alldifferent` *(system of constraints)*.
part of system of constraints: `global_cardinality`.
related to a common problem: `same` *(matrix reconstruction problem)*.

Keywords
constraint arguments: pure functional dependency.
constraint type: system of constraints, predefined constraint, timetabling constraint.
modelling: functional dependency, matrix, matrix model.
5.65 coloured_cumulative

**DESCRIPTION**

Origin

Derived from cumulative and nvalues.

Constraint

coloured_cumulative(TASKS, LIMIT)

Synonym

colored_cumulative.

Arguments

\[
\text{TASKS} : \text{collection} \left( \begin{array}{c}
\text{origin} - \text{dvar}, \\
\text{duration} - \text{dvar}, \\
\text{end} - \text{dvar}, \\
\text{colour} - \text{dvar}
\end{array} \right)
\]

\[
\text{LIMIT} : \text{int}
\]

Restrictions

\[
\text{require_at_least}(2, \text{TASKS}, [\text{origin}, \text{duration}, \text{end}])
\]

\[
\text{required}(\text{TASKS}, \text{colour})
\]

\[
\text{TASKS}.\text{duration} \geq 0
\]

\[
\text{TASKS}.\text{origin} \leq \text{TASKS}.\text{end}
\]

\[
\text{LIMIT} \geq 0
\]

**Purpose**

Consider the set \( \mathcal{T} \) of tasks described by the TASKS collection. The coloured_cumulative constraint enforces that, at each point in time, the number of distinct colours of the set of tasks that overlap that point, does not exceed a given limit.

A task overlaps a point \( i \) if and only if (1) its origin is less than or equal to \( i \), and (2) its end is strictly greater than \( i \). For each task of \( \mathcal{T} \) it also imposes the constraint \( \text{origin} + \text{duration} = \text{end} \).

**Example**

\[
\begin{pmatrix}
\text{origin} - 1 & \text{duration} - 2 & \text{end} - 3 & \text{colour} - 1, \\
\text{origin} - 2 & \text{duration} - 9 & \text{end} - 11 & \text{colour} - 2, \\
\text{origin} - 3 & \text{duration} - 10 & \text{end} - 13 & \text{colour} - 3, \\
\text{origin} - 6 & \text{duration} - 6 & \text{end} - 12 & \text{colour} - 2, \\
\text{origin} - 7 & \text{duration} - 2 & \text{end} - 9 & \text{colour} - 3
\end{pmatrix}
\]

Figure 5.134 shows the solution associated with the example. Each rectangle of the figure corresponds to a task of the coloured_cumulative constraint. Tasks that have their colour attribute set to 1, 2 and 3 are respectively coloured in yellow, blue and pink. The coloured_cumulative constraint holds since at each point in time we do not have more than \( \text{LIMIT} = 2 \) distinct colours.

**Typical**

\[
|\text{TASKS}| > 1
\]

\[
\text{range}(\text{TASKS}.\text{origin}) > 1
\]

\[
\text{range}(\text{TASKS}.\text{duration}) > 1
\]

\[
\text{range}(\text{TASKS}.\text{end}) > 1
\]

\[
\text{range}(\text{TASKS}.\text{colour}) > 1
\]

\[
\text{LIMIT} < \text{nval}(\text{TASKS}.\text{colour})
\]
Symmetries

- Items of TASKS are permutable.
- One and the same constant can be added to the origin and end attributes of all items of TASKS.
- All occurrences of two distinct values of TASKS.colour can be swapped: all occurrences of a value of TASKS.colour can be renamed to any unused value.
- LIMIT can be increased.

Arg. properties

Contractible wrt. TASKS.

Usage

Useful for scheduling problems where a machine can only proceed in parallel a maximum number of tasks of distinct type. This condition cannot be modelled by the classical cumulative constraint.

Reformulation

The coloured_cumulative constraint can be expressed in term of a set of reified constraints and of |TASKS| nvalue constraints:

1. For each pair of tasks TASKS[i], TASKS[j] (i, j ∈ [1,|TASKS|]) of the TASKS collection we create a variable \( C_{ij} \) which is set to the colour of task TASKS[j] if task TASKS[j] overlaps the origin attribute of task TASKS[i], and to the colour of task TASKS[i] otherwise:
   - If \( i = j \):
     - \( C_{ij} = \text{TASKS}[i].\text{colour} \)
   - If \( i \neq j \):
     - \( C_{ij} = \text{TASKS}[i].\text{colour} \lor C_{ij} = \text{TASKS}[j].\text{colour} \)
     - \((\text{TASKS}[j].\text{origin} \leq \text{TASKS}[i].\text{origin} \land \text{TASKS}[j].\text{end} > \text{TASKS}[i].\text{origin}) \lor (C_{ij} = \text{TASKS}[j].\text{colour})) \lor \n     - ((\text{TASKS}[j].\text{origin} > \text{TASKS}[i].\text{origin} \lor \text{TASKS}[j].\text{end} \leq \text{TASKS}[i].\text{origin}) \land (C_{ij} = \text{TASKS}[i].\text{colour}))

2. For each task TASKS[i] (i ∈ [1,|TASKS|]) we create a variable \( N_i \) which gives the number of distinct colours associated with the tasks that overlap the origin of task TASKS[i] (TASKS[i] overlaps its own origin) and we impose \( N_i \) to not exceed the maximum number of distinct colours LIMIT allowed at each instant:
   - \( N_i \geq 1 \land N_i \leq \text{LIMIT} \).
   - \text{nvalue}(N_i, \{C_{i1}, C_{i2}, \ldots, C_{i|\text{TASKS}|}\}).

Figure 5.134: A coloured cumulative solution with at most two distinct colours in parallel
See also

- **assignment dimension added:** coloured_cumulatives.
- **common keyword:** cumulative, track (resource constraint).
- **implied by:** cumulative.
- **related:** nvalue.
- **specialisation:** disjoint_tasks (a colour is assigned to each collection of tasks of constraint disjoint_tasks and a limit of one single colour is enforced).
- **used in graph description:** nvalues.

**Keywords**

- **characteristic of a constraint:** coloured.
- **constraint type:** scheduling constraint, resource constraint, temporal constraint.
- **filtering:** compulsory part.
- **modelling:** number of distinct values, zero-duration task.
Arc input(s) | TASKS
---|---
Arc generator | SELF $\mapsto$ collection(tasks)
Arc arity | 1
Arc constraint(s) | tasks.origin + tasks.duration = tasks.end
Graph property(ies) | NARC $=$ |TASKS|

Arc input(s) | TASKS TASKS
---|---
Arc generator | PRODUCT $\mapsto$ collection(tasks1,tasks2)
Arc arity | 2
Arc constraint(s) | • tasks1.duration $>$ 0
• tasks2.origin $\leq$ tasks1.origin
• tasks1.origin $<$ tasks2.end
Graph class | • ACYCLIC
• BIPARTITE
• NO LOOP
Sets | SUCC $\mapsto$
| | source,
| | variables $-$ col(VARIABLES $-$ collection(var $-$ dvar),
| | item[var $-$ TASKS.colour])
Constraint(s) on sets | nvalues(variables, $\leq$, LIMIT)

Graph model | Same as cumulative, except that we use another constraint for computing the resource consumption at each time point.
Parts (A) and (B) of Figure 5.135 respectively show the initial and final graph associated with the second graph constraint of the Example slot. On the one hand, each source vertex of the final graph can be interpreted as a time point. On the other hand the successors of a source vertex correspond to those tasks that overlap that time point. The coloured_cumulative constraint holds since for each successor set $S$ of the final graph the number of distinct colours of the tasks in $S$ does not exceed the LIMIT 2.

Signature | Since TASKS is the maximum number of vertices of the final graph of the first graph constraint we can rewrite NARC $=$ |TASKS| to NARC $\geq$ |TASKS|. This leads to simplify NARC to NARC.
Figure 5.135: Initial and final graph of the coloured_cumulative constraint
### 5.66 coloured_cumulatives

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from cumulatives and nvalues.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>coloured_cumulatives(TASKS, MACHINES)</td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td>colored_cumulatives.</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>TASKS : collection (machine=dvar, origin=dvar, duration=dvar, end=dvar, colour=dvar)</td>
<td>MACHINES : collection (id=int, capacity=int)</td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(TASKS, [machine, colour])</td>
<td>required_at_least(2, TASKS, [origin, duration, end])</td>
</tr>
<tr>
<td></td>
<td>TASKS.duration ≥ 0</td>
<td>TASKS.origin ≤ TASKS.end</td>
</tr>
<tr>
<td></td>
<td>required(MACHINES, [id, capacity])</td>
<td>distinct(MACHINES, id)</td>
</tr>
<tr>
<td></td>
<td>MACHINES.capacity ≥ 0</td>
<td></td>
</tr>
</tbody>
</table>

Consider a set $T$ of tasks described by the TASKS collection. The coloured_cumulatives constraint enforces for each machine $m$ of the MACHINES collection the following condition: at each point in time $p$, the numbers of distinct colours of the set of tasks that both overlap that point $p$ and are assigned to machine $m$ does not exceed the capacity of machine $m$. A task overlaps a point $i$ if and only if (1) its origin is less than or equal to $i$, and (2) its end is strictly greater than $i$. It also imposes for each task of $T$ the constraint origin + duration = end.

**Example**

```
(machine−1 origin−6 duration−6 end−12 colour−1, machine−1 origin−2 duration−9 end−11 colour−2, machine−2 origin−7 duration−3 end−10 colour−2, machine−1 origin−1 duration−2 end−3 colour−1, machine−2 origin−4 duration−5 end−9 colour−2, machine−1 origin−3 duration−10 end−13 colour−1, (id−1 capacity−2, id−2 capacity−1))
```

Figure 5.136 shows the solution associated with the example. Each rectangle of the figure corresponds to a task of the coloured_cumulatives constraint. Tasks that have their colour attribute set to 1 and 2 are respectively coloured in blue and pink. The coloured_cumulatives constraint holds since for machine 1 we have at most two distinct colours in parallel (which is the maximum capacity for machine 1), while for machine 2 we have no more than one single colour in parallel (which is actually the maximum capacity for machine 2).
**Typical**

\[
\begin{align*}
|\text{TASKS}| &> 1 \\
\text{range}(\text{TASKS}.\text{machine}) &> 1 \\
\text{range}(\text{TASKS}.\text{origin}) &> 1 \\
\text{range}(\text{TASKS}.\text{duration}) &> 1 \\
\text{range}(\text{TASKS}.\text{end}) &> 1 \\
\text{range}(\text{TASKS}.\text{colour}) &> 1 \\
\text{TASKS}.\text{duration} &> 0 \\
|\text{MACHINES}| &> 1 \\
\text{MACHINES}.\text{capacity} &> 0 \\
\text{MACHINES}.\text{capacity} &< \text{ncval}(\text{TASKS}.\text{colour}) \\
|\text{TASKS}| &> |\text{MACHINES}|
\end{align*}
\]

**Symmetries**

- Items of TASKS are permutable.
- Items of MACHINES are permutable.
- MACHINES.capacity can be increased.
- All occurrences of two distinct values in TASKS.machine or MACHINES.id can be swapped; all occurrences of a value in TASKS.machine or MACHINES.id can be renamed to any unused value.

**Arg. properties**

Contractible wrt. TASKS.

**Usage**

Useful for scheduling problems where several machines are available and where you have to assign each task to a specific machine. In addition each machine can only proceed in parallel a maximum number of tasks of distinct types.

**Reformulation**

The coloured_cumulatives constraint can be expressed in term of a set of reified constraints and of |TASKS| nvalue constraints:

1. For each pair of tasks \(\text{TASKS}[i], \text{TASKS}[j]\) \((i, j \in [1, |\text{TASKS}|])\) of the TASKS collection we create a variable \(C_{ij}\) which is set to the colour of task \(\text{TASKS}[j]\) if both tasks are assigned to the same machine and if task \(\text{TASKS}[j]\) overlaps the origin attribute of task \(\text{TASKS}[i]\), and to the colour of task \(\text{TASKS}[i]\) otherwise:

   - If \(i = j\):
     \[
     C_{ij} = \text{TASKS}[i].\text{colour}.
     \]

Figure 5.136: Assignment of the tasks on the two machines
• If \( i \neq j \):
  - \( C_{ij} = \text{TASKS}[i].\text{colour} \vee C_{ij} = \text{TASKS}[j].\text{colour} \).
  - \( (\text{TASKS}[j].\text{machine} = \text{TASKS}[i].\text{machine} \land \text{TASKS}[j].\text{origin} \leq \text{TASKS}[i].\text{origin} \land \text{TASKS}[j].\text{end} > \text{TASKS}[i].\text{origin}) \lor (C_{ij} = \text{TASKS}[j].\text{colour}) \) \lor
  - \( (\text{TASKS}[j].\text{machine} \neq \text{TASKS}[i].\text{machine} \lor \text{TASKS}[j].\text{origin} > \text{TASKS}[i].\text{origin} \land \text{TASKS}[j].\text{end} \leq \text{TASKS}[i].\text{origin}) \lor (C_{ij} = \text{TASKS}[i].\text{colour}) \)

2. For each task \( \text{TASKS}[i] \) \((i \in [1, |\text{TASKS}|])\) we create a variable \( N_i \) which gives the number of distinct colours associated with the tasks that both are assigned to the same machine as task \( \text{TASKS}[i] \) and overlap the origin of task \( \text{TASKS}[i] \) (\( \text{TASKS}[i] \) overlaps its own origin) and we impose \( N_i \) to not exceed the maximum number of distinct colours \( \text{LIMIT} \) allowed at each instant:
   - \( N_i \geq 1 \land N_i \leq \text{LIMIT} \).
   - \( \text{nvalue}(N_i, (C_{i1}, C_{i2}, \ldots, C_{i|\text{TASKS}|})). \)

**See also**
- assignment dimension removed: coloured_cumulative (machine attribute removed), cumulative (machine attribute removed and number of distinct colours replaced by sum of task heights).
- common keyword: cumulative, cumulatives (resource constraint).
- related: nvalue.
- used in graph description: nvalues.

**Keywords**
- characteristic of a constraint: coloured.
- constraint type: scheduling constraint, resource constraint, temporal constraint.
- filtering: compulsory part.
- modelling: number of distinct values, assignment dimension, zero-duration task.
Arc input(s)  TASKS
Arc generator  $SELF \rightarrow \text{collection}(\text{tasks})$
Arc arity  1
Arc constraint(s)  $\text{tasks}.\text{origin} + \text{tasks}.\text{duration} = \text{tasks}.\text{end}$
Graph property(ies)  $\text{NARC} = |\text{TASKS}|$

For all items of $\text{MACHINES}$:

Arc input(s)  TASKS TASKS
Arc generator  $\text{PRODUCT} \rightarrow \text{collection}(\text{tasks1}, \text{tasks2})$
Arc arity  2
Arc constraint(s)  
  - $\text{tasks1}.\text{machine} = \text{MACHINES}.\text{id}$
  - $\text{tasks1}.\text{machine} = \text{tasks2}.\text{machine}$
  - $\text{tasks1}.\text{duration} > 0$
  - $\text{tasks2}.\text{origin} \leq \text{tasks1}.\text{origin}$
  - $\text{tasks1}.\text{origin} < \text{tasks2}.\text{end}$
Graph class  
  - ACYCLIC
  - BIPARTITE
  - NO LOOP
Sets  
  $\text{SUCC} \mapsto$
  
  \[
  \left[ \begin{array}{c}
  \text{source}, \\
  \text{variables} \in \text{col}( \text{VARIABLES} \rightarrow \text{collection}([\text{VAR} \leftarrow \text{dvar}], [\text{item}([\text{VAR} \leftarrow \text{TASKS}.\text{colour}])]) )
  \end{array} \right]
  \]
Constraint(s) on sets  $\text{nvalues}(\text{variables}, \leq \text{MACHINES}.\text{capacity})$

Graph model  Parts (A) and (B) of Figure 5.137 respectively shows the initial and final graph associated with machines 1 and 2 involved in the Example slot. On the one hand, each source vertex of the final graph can be interpreted as a time point $p$ on a specific machine $m$. On the other hand the successors of a source vertex correspond to those tasks that both overlap that time point $p$ and are assigned to machine $m$. The coloured_cumulatives constraint holds since for each successor set $S$ of the final graph the number of distinct colours in $S$ does not exceed the capacity of the machine corresponding to the time point associated with $S$.

Signature  Since $\text{TASKS}$ is the maximum number of vertices of the final graph of the first graph constraint we can rewrite $\text{NARC} = |\text{TASKS}|$ to $\text{NARC} \geq |\text{TASKS}|$. This leads to simplify $\text{NARC}$ to $\text{NARC}$. 
Figure 5.137: Initial and final graph of the coloured_cumulatives constraint
5.67 common

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>N. Beldiceanu</td>
<td>common(NCOMMON1, NCOMMON2, VARIABLES1, VARIABLES2)</td>
<td></td>
</tr>
</tbody>
</table>

**Arguments**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NCOMMON1</td>
<td>dvar</td>
</tr>
<tr>
<td>NCOMMON2</td>
<td>dvar</td>
</tr>
<tr>
<td>VARIABLES1</td>
<td>collection(var–dvar)</td>
</tr>
<tr>
<td>VARIABLES2</td>
<td>collection(var–dvar)</td>
</tr>
</tbody>
</table>

**Restrictions**

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NCOMMON1 ≥ 0</td>
</tr>
<tr>
<td>NCOMMON1 ≤</td>
</tr>
<tr>
<td>NCOMMON2 ≥ 0</td>
</tr>
<tr>
<td>NCOMMON2 ≤</td>
</tr>
<tr>
<td>required(VARIABLES1, var)</td>
</tr>
<tr>
<td>required(VARIABLES2, var)</td>
</tr>
</tbody>
</table>

**Purpose**

NCOMMON1 is the number of variables of the collection of variables VARIABLES1 taking a value in VARIABLES2.

NCOMMON2 is the number of variables of the collection of variables VARIABLES2 taking a value in VARIABLES1.

**Example**

\[
\begin{pmatrix}
3, 4, (1, 9, 1, 5), \\
\text{var} - 2, \\
\text{var} - 1, \\
\text{var} - 9, \\
\text{var} - 9, \\
\text{var} - 6, \\
\text{var} - 9
\end{pmatrix}
\]

The common constraint holds since:

- Its first argument NCOMMON1 = 3 corresponds to the number of values of the collection \(1, 9, 1, 5\) that occur within \(2, 1, 9, 6, 9\).
- Its second argument NCOMMON2 = 4 corresponds to the number of values of the collection \(2, 1, 9, 6, 9\) that occur within \(1, 9, 1, 5\).

**Typical**

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>range(VARIABLES1.var) &gt; 1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>range(VARIABLES2.var) &gt; 1</td>
</tr>
</tbody>
</table>
Symmetries

- Arguments are permutable w.r.t. permutation \((NCOMMON1, NCOMMON2)\) \((VARIABLES1, VARIABLES2)\).
- Items of \(VARIABLES1\) are permutable.
- Items of \(VARIABLES2\) are permutable.
- All occurrences of two distinct values in \(VARIABLES1\).var or \(VARIABLES2\).var can be swapped; all occurrences of a value in \(VARIABLES1\).var or \(VARIABLES2\).var can be renamed to any unused value.

Arg. properties

- Functional dependency: \(NCOMMON1\) determined by \(VARIABLES1\) and \(VARIABLES2\).
- Functional dependency: \(NCOMMON2\) determined by \(VARIABLES1\) and \(VARIABLES2\).

Remark

It was shown in [66] that, finding out whether the common constraint has a solution or not is NP-hard. This was achieved by reduction from 3-SAT.

See also

- common keyword: alldifferent_on_intersection, nvalue_on_intersection, same_intersection (constraint on the intersection).
- generalisation: common_interval (variable replaced by variable/constant), common_modulo (variable replaced by variable mod constant), common_partition (variable replaced by variable ∈ partition).
- related: among_var.roots.
- root concept: among.
- specialisation: uses \((NCOMMON2=|VARIABLES2|)\).

Keywords

- complexity: 3-SAT.
- constraint arguments: constraint between two collections of variables, pure functional dependency.
- constraint type: constraint on the intersection.
- final graph structure: acyclic, bipartite, no loop.
Graph model

Parts (A) and (B) of Figure 5.138 respectively show the initial and final graph associated with the Example slot. Since we use the NSOURCE and NSINK graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since the final graph has only 3 sources and 4 sinks the variables NCOMMON1 and NCOMMON2 are respectively equal to 3 and 4. Note that all the vertices corresponding to the variables that take values 5, 2 or 6 were removed from the final graph since there is no arc for which the associated equality constraint holds.

Figure 5.138: Initial and final graph of the common constraint
5.68 common_interval

**Description**

Derived from common.

**Constraint**

\[
\begin{align*}
\text{common_interval} ( & N_{\text{COMMON1}}, \\
& N_{\text{COMMON2}}, \\
& \text{VARIABLES1,} \\
& \text{VARIABLES2}, \\
& \text{SIZE\_INTERVAL})
\end{align*}
\]

**Arguments**

- \(N_{\text{COMMON1}}\) : dvar
- \(N_{\text{COMMON2}}\) : dvar
- \(\text{VARIABLES1}\) : collection(var–dvar)
- \(\text{VARIABLES2}\) : collection(var–dvar)
- \(\text{SIZE\_INTERVAL}\) : int

**Restrictions**

- \(N_{\text{COMMON1}} \geq 0\)
- \(N_{\text{COMMON1}} \leq |\text{VARIABLES1}|\)
- \(N_{\text{COMMON2}} \geq 0\)
- \(N_{\text{COMMON2}} \leq |\text{VARIABLES2}|\)
- \(\text{required} (\text{VARIABLES1}, \text{var})\)
- \(\text{required} (\text{VARIABLES2}, \text{var})\)
- \(\text{SIZE\_INTERVAL} > 0\)

**Purpose**

\(N_{\text{COMMON1}}\) is the number of variables of the collection of variables \(\text{VARIABLES1}\) taking a value in one of the intervals derived from the values assigned to the variables of the collection \(\text{VARIABLES2}\): To each value \(v\) assigned to a variable of the collection \(\text{VARIABLES2}\) we associate the interval \([\text{SIZE\_INTERVAL} \cdot \lfloor v/\text{SIZE\_INTERVAL} \rfloor, \text{SIZE\_INTERVAL} \cdot \lfloor v/\text{SIZE\_INTERVAL} \rfloor + \text{SIZE\_INTERVAL} - 1]\).

\(N_{\text{COMMON2}}\) is the number of variables of the collection of variables \(\text{VARIABLES2}\) taking a value in one of the intervals derived from the values assigned to the variables of the collection \(\text{VARIABLES1}\): To each value \(v\) assigned to a variable of the collection \(\text{VARIABLES1}\) we associate the interval \([\text{SIZE\_INTERVAL} \cdot \lfloor v/\text{SIZE\_INTERVAL} \rfloor, \text{SIZE\_INTERVAL} \cdot \lfloor v/\text{SIZE\_INTERVAL} \rfloor + \text{SIZE\_INTERVAL} - 1]\).

**Example**

\[
\begin{align*}
3, 2, (8, 6, 6, 0), \\
\text{var} - 7, \\
\text{var} - 3, \\
\text{var} - 3, \\
\text{var} - 3, \\
\text{var} - 3, \\
\text{var} - 7
\end{align*}
\]

In the example, the last argument \(\text{SIZE\_INTERVAL} = 3\) defines the following family of intervals \([3 \cdot k, 3 \cdot k + 2]\), where \(k\) is an integer. As a consequence the items of
collection \(\langle 8, 6, 6, 0 \rangle\) respectively correspond to intervals \([6, 8], [6, 8], [6, 8]\) and \([0, 2]\). Similarly the items of collection \(\langle 7, 3, 3, 3, 3, 7 \rangle\) respectively correspond to intervals \([6, 8], [3, 5], [3, 5], [3, 5], [3, 5], [6, 8]\). The common interval constraint holds since:

- Its first argument \(N_{\text{COMMON}_1} = 3\) is the number of intervals associated with the items of collection \(\langle 8, 6, 6, 0 \rangle\) that also correspond to intervals associated with \(\langle 7, 3, 3, 3, 3, 7 \rangle\).
- Its second argument \(N_{\text{COMMON}_2} = 2\) is the number of intervals associated with the items of collection \(\langle 7, 3, 3, 3, 3, 7 \rangle\) that also correspond to intervals associated with \(\langle 8, 6, 6, 0 \rangle\).

**Typical**

\[
\begin{align*}
\text{range}(\text{VARIABLES1}.\text{var}) & > 1 \\
\text{range}(\text{VARIABLES2}.\text{var}) & > 1 \\
\text{SIZE}\_\text{INTERVAL} & > 1 \\
\text{SIZE}\_\text{INTERVAL} & < \text{range}(\text{VARIABLES1}.\text{var}) \\
\text{SIZE}\_\text{INTERVAL} & < \text{range}(\text{VARIABLES2}.\text{var})
\end{align*}
\]

**Symmetries**

- Arguments are **permutable** w.r.t. permutation \((N_{\text{COMMON}_1}, N_{\text{COMMON}_2}) (\text{VARIABLES1}, \text{VARIABLES2}) (\text{SIZE}\_\text{INTERVAL})\).
- Items of \(\text{VARIABLES1}\) are **permutable**.
- Items of \(\text{VARIABLES2}\) are **permutable**.
- An occurrence of a value of \(\text{VARIABLES1}.\text{var}\) that belongs to the \(k\)-th interval, of size \(\text{SIZE}\_\text{INTERVAL}\), can be **replaced** by any other value of the same interval.
- An occurrence of a value of \(\text{VARIABLES2}.\text{var}\) that belongs to the \(k\)-th interval, of size \(\text{SIZE}\_\text{INTERVAL}\), can be **replaced** by any other value of the same interval.

**Arg. properties**

- **Functional dependency**: \(N_{\text{COMMON}_1}\) determined by \(\text{VARIABLES1}, \text{VARIABLES2}\) and \(\text{SIZE}\_\text{INTERVAL}\).
- **Functional dependency**: \(N_{\text{COMMON}_2}\) determined by \(\text{VARIABLES1}, \text{VARIABLES2}\) and \(\text{SIZE}\_\text{INTERVAL}\).

**See also**

**specialisation**: common (variable/constant replaced by variable).

**Keywords**

- **constraint arguments**: constraint between two collections of variables, pure functional dependency.
- **final graph structure**: acyclic, bipartite, no loop.
- **modelling**: interval, functional dependency.
Arc input(s)  VARIABLES1 VARIABLES2
Arc generator  \( PRODUCT \rightarrow \text{collection}(\text{variables1}, \text{variables2}) \)
Arc arity  2
Arc constraint(s)  \( \text{variables1} \cdot \text{var} / \text{SIZE} \_\text{INTERVAL} = \text{variables2} \cdot \text{var} / \text{SIZE} \_\text{INTERVAL} \)
Graph property(ies)  
- \( \text{NSOURCE} = \text{NCOMMON1} \)
- \( \text{NSINK} = \text{NCOMMON2} \)
Graph class  
- \( \text{ACYCLIC} \)
- \( \text{BIPARTITE} \)
- \( \text{NO \_LOOP} \)

Graph model

Parts (A) and (B) of Figure 5.139 respectively show the initial and final graph associated with the Example slot. Since we use the \( \text{NSOURCE} \) and \( \text{NSINK} \) graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since the graph has only 3 sources and 2 sinks the variables \( \text{NCOMMON1} \) and \( \text{NCOMMON2} \) are respectively equal to 3 and 2. Note that the vertices corresponding to the variables that take values 0 or 3 were removed from the final graph since there is no arc for which the associated arc constraint holds.

![Graph](image)

Figure 5.139: Initial and final graph of the \text{common} \_\text{interval} constraint
5.69 common_modulo

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from common.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td><code>common_modulo(NCOMMON1, NCOMMON2, VARIABLES1, VARIABLES2, M)</code></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>NCOMMON1 : dvar</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NCOMMON2 : dvar</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES1 : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES2 : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M : int</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>NCOMMON1 ≥ 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NCOMMON1 ≤</td>
<td>VARIABLES1</td>
</tr>
<tr>
<td></td>
<td>NCOMMON2 ≥ 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NCOMMON2 ≤</td>
<td>VARIABLES2</td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES1, var)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES2, var)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M &gt; 0</td>
<td></td>
</tr>
</tbody>
</table>
| Purpose     | NCOMMON1 is the number of variables of the collection of variables VARIABLES1 taking a value situated in an equivalence class (congruence modulo a fixed number M) derived from the values assigned to the variables of VARIABLES2 and from M.  
NCOMMON2 is the number of variables of the collection of variables VARIABLES2 taking a value situated in an equivalence class (congruence modulo a fixed number M) derived from the values assigned to the variables of VARIABLES1 and from M. |       |
| Example     | `{3, 4, ⟨0, 4, 0, 8⟩,  
     var − 7,  
     var − 5,  
     ⟨var − 4,  
     var − 9,  
     var − 2,  
     var − 4⟩, 5}` |       |

In the example, the last argument M = 5 defines the equivalence classes a ≡ 0 (mod 5), a ≡ 1 (mod 5), a ≡ 2 (mod 5), a ≡ 3 (mod 5), and a ≡ 4 (mod 5) where a is an integer. As a consequence the items of collection ⟨0, 4, 0, 8⟩ respectively correspond to the equivalence classes a ≡ 0 (mod 5), a ≡ 4 (mod 5), a ≡ 0 (mod 5), and a ≡ 3 (mod 5). Similarly the items of collection ⟨7, 5, 4, 9, 2, 4⟩ respectively correspond to the equivalence classes a ≡ 2 (mod 5), a ≡ 0 (mod 5), a ≡ 4 (mod 5), a ≡ 4 (mod 5), a ≡ 2 (mod 5), and a ≡ 4 (mod 5). The common_modulo constraint holds since:

- Its first argument NCOMMON1 = 3 is the number of equivalence classes associated with the items of collection ⟨0, 4, 0, 8⟩ that also correspond to equivalence classes associated with ⟨7, 5, 4, 9, 2, 4⟩.
- Its second argument \( N_{\text{COMMON2}} = 4 \) is the number of equivalence classes associated with the items of collection \( \langle 7, 5, 4, 9, 2, 4 \rangle \) that also correspond to equivalence classes associated with \( \langle 0, 4, 0, 8 \rangle \).

**Typical**

\[
\begin{align*}
|\text{VARIABLES1}| & > 1 \\
\text{range}(\text{VARIABLES1}.\text{var}) & > 1 \\
|\text{VARIABLES2}| & > 1 \\
\text{range}(\text{VARIABLES2}.\text{var}) & > 1 \\
M & > 1 \\
M & < \text{maxval}(\text{VARIABLES1}.\text{var}) \\
M & < \text{maxval}(\text{VARIABLES2}.\text{var})
\end{align*}
\]

**Symmetries**

- Arguments are permutable w.r.t. permutation \((N_{\text{COMMON1}}, N_{\text{COMMON2}})\) \((\text{VARIABLES1}, \text{VARIABLES2})\) \(M\).
- Items of \text{VARIABLES1} are permutable.
- Items of \text{VARIABLES2} are permutable.
- An occurrence of a value \( u \) of \text{VARIABLES1}.\text{var} can be replaced by any other value \( v \) such that \( v \) is congruent to \( u \) modulo \( M \).
- An occurrence of a value \( u \) of \text{VARIABLES2}.\text{var} can be replaced by any other value \( v \) such that \( v \) is congruent to \( u \) modulo \( M \).

**Arg. properties**

- Functional dependency: \( N_{\text{COMMON1}} \) determined by \text{VARIABLES1}, \text{VARIABLES2} and \( M \).
- Functional dependency: \( N_{\text{COMMON2}} \) determined by \text{VARIABLES1}, \text{VARIABLES2} and \( M \).

**See also**

specialisation: common \((\text{variable mod constant replaced by variable})\).

**Keywords**

characteristic of a constraint: modulo.

constraint arguments: constraint between two collections of variables, pure functional dependency.

final graph structure: acyclic, bipartite, no loop.

modelling: functional dependency.
Arc input(s) | VARIABLES1 VARIABLES2
---|---
Arc generator | \( \text{PRODUCT} \rightarrow \text{collection}(\text{variables1}, \text{variables2}) \)
Arc arity | 2
Arc constraint(s) | \text{variables1}.\text{var} \text{mod} M = \text{variables2}.\text{var} \text{mod} M
Graph property(ies) | • NSOURCE = \text{NCOMMON1}
• NSINK = \text{NCOMMON2}
Graph class | • ACYCLIC
• BIPARTITE
• NO_LOOP

Graph model

Parts (A) and (B) of Figure 5.140 respectively show the initial and final graph associated with the Example slot. Since we use the NSOURCE and NSINK graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since the graph has only 3 sources and 4 sinks the variables NCOMMON1 and NCOMMON2 are respectively equal to 3 and 4. Note that the vertices corresponding to the variables that take values 8, 7 or 2 were removed from the final graph since there is no arc for which the associated arc constraint holds.

![Graph Diagram](image)

Figure 5.140: Initial and final graph of the common modulo constraint
5.70 common_partition

**DESCRIPTION**

Derived from `common`.

**Constraint**

\[
\text{common} \_\text{partition} \left( \begin{array}{c}
\text{NCOMMON1}, \\
\text{NCOMMON2}, \\
\text{VARIABLES1}, \\
\text{VARIABLES2}, \\
\text{PARTITIONS}
\end{array} \right)
\]

**Type**

VALUES : `collection(val-int)`

**Arguments**

- \(\text{NCOMMON1} : \text{dvar}\)
- \(\text{NCOMMON2} : \text{dvar}\)
- \(\text{VARIABLES1} : \text{collection(var-dvar)}\)
- \(\text{VARIABLES2} : \text{collection(var-dvar)}\)
- \(\text{PARTITIONS} : \text{collection(p-VALUES)}\)

**Restrictions**

\[
\begin{align*}
|\text{VALUES}| & \geq 1 \\
\text{required}(\text{VALUES}, \text{val}) \\
\text{distinct}(\text{VALUES}, \text{val}) \\
\text{NCOMMON1} & \geq 0 \\
\text{NCOMMON1} & \leq |\text{VARIABLES1}| \\
\text{NCOMMON2} & \geq 0 \\
\text{NCOMMON2} & \leq |\text{VARIABLES2}| \\
\text{required}(\text{VARIABLES1}, \text{var}) \\
\text{required}(\text{VARIABLES2}, \text{var}) \\
\text{required}(\text{PARTITIONS}, \text{p}) \\
|\text{PARTITIONS}| & \geq 2
\end{align*}
\]

**Purpose**

`NCOMMON1` is the number of variables of the \(\text{VARIABLES1}\) collection taking a value in a partition derived from the values assigned to the variables of \(\text{VARIABLES2}\) and from \(\text{PARTITIONS}\).

`NCOMMON2` is the number of variables of the \(\text{VARIABLES2}\) collection taking a value in a partition derived from the values assigned to the variables of \(\text{VARIABLES1}\) and from \(\text{PARTITIONS}\).
In the example, the last argument \texttt{PARTITIONS} defines the partitions \(p - \langle 1,3 \rangle\), \(p - \langle 4 \rangle\) and \(p - \langle 2,6 \rangle\). As a consequence the first three items of collection \(\langle 2,3,6,0 \rangle\) respectively correspond to the partitions \(p - \langle 2,6 \rangle\), \(p - \langle 1,3 \rangle\), and \(p - \langle 2,6 \rangle\). Similarly the items of collection \(\langle 0,6,3,3,7,1 \rangle\) (from which we remove items 0 and 7 since they do not belong to any partition) respectively correspond to the partitions \(p - \langle 2,6 \rangle\), \(p - \langle 1,3 \rangle\), \(p - \langle 1,3 \rangle\), and \(p - \langle 1,3 \rangle\). The \texttt{common\_partition} constraint holds since:

- Its first argument \(\texttt{NCOMMON1} = 3\) is the number of partitions associated with the items of collection \(\langle 2,3,6,0 \rangle\) that also correspond to partitions associated with \(\langle 0,6,3,3,7,1 \rangle\).

- Its second argument \(\texttt{NCOMMON2} = 4\) is the number of partitions associated with the items of collection \(\langle 0,6,3,3,7,1 \rangle\) that also correspond to partitions associated with \(\langle 2,3,6,0 \rangle\).

\begin{itemize}
  \item \texttt{Typical} \hspace{100pt} \texttt{Symmetries} \\
  \texttt{|VARIABLES1| > 1} \hspace{100pt} \texttt{\textbullet Arguments are permutabl} \\
  \texttt{range(VARIABLES1.var) > 1} \hspace{100pt} \texttt{e \ w.r.t. permutation (\texttt{NCOMMON1, NCOMMON2})} \\
  \texttt{|VARIABLES2| > 1} \hspace{100pt} \texttt{(VARIABLES1, VARIABLES2) (PARTITIONS).} \\
  \texttt{range(VARIABLES2.var) > 1} \hspace{100pt} \texttt{\textbullet Items of VARIABLES1 are permutabl} \\
  \texttt{|VARIABLES1| > |PARTITIONS|} \hspace{100pt} \texttt{e.} \\
  \texttt{|VARIABLES2| > |PARTITIONS|} \hspace{100pt} \texttt{\textbullet Items of VARIABLES2 are permutabl} \\
  \texttt{} \hspace{100pt} \texttt{e.} \\
  \texttt{} \hspace{100pt} \texttt{\textbullet Items of PARTITIONS are permutabl} \\
  \texttt{} \hspace{100pt} \texttt{e.} \\
  \texttt{} \hspace{100pt} \texttt{\textbullet Items of PARTITIONS.p are permutabl} \\
  \texttt{} \hspace{100pt} \texttt{e.} \\
  \texttt{} \hspace{100pt} \texttt{\textbullet An occurrence of a value of VARIABLES1.var can be replaced by any other value} \\
  \texttt{} \hspace{100pt} \texttt{that also belongs to the same partition of PARTITIONS.} \\
  \texttt{} \hspace{100pt} \texttt{\textbullet An occurrence of a value of VARIABLES2.var can be replaced by any other value} \\
  \texttt{} \hspace{100pt} \texttt{that also belongs to the same partition of PARTITIONS.} \\
\end{itemize}

\begin{itemize}
  \item \texttt{Arg. properties} \\
  \texttt{} \hspace{100pt} \texttt{\textbullet Functional dependency: NCOMMON1 determined by VARIABLES1, VARIABLES} \\
  \texttt{} \hspace{100pt} \texttt{2 and PARTITIONS.} \\
  \texttt{} \hspace{100pt} \texttt{\textbullet Functional dependency: NCOMMON2 determined by VARIABLES1, VARIABLES} \\
  \texttt{} \hspace{100pt} \texttt{2 and PARTITIONS.} \\
\end{itemize}
See also

specialisation: common \((\text{variable } \in \text{partition replaced by variable})\).
used in graph description: in_same_partition.

Keywords

characteristic of a constraint: partition.
constraint arguments: constraint between two collections of variables, pure functional dependency.
final graph structure: acyclic, bipartite, no loop.
modelling: functional dependency.
Graph model

Parts (A) and (B) of Figure 5.141 respectively show the initial and final graph associated with the Example slot. Since we use the NSOURCE and NSINK graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since the graph has only 3 sources and 4 sinks the variables NCOMMON1 and NCOMMON2 are respectively equal to 3 and 4. Note that the vertices corresponding to the variables that take values 0 or 7 were removed from the final graph since there is no arc for which the associated in_same_partition constraint holds.

Figure 5.141: Initial and final graph of the common_partition constraint
## 5.71 compare_and_count

### DESCRIPTION

**Origin**  
Generalise discrepancy

**Constraint**  
```plaintext
compare_and_count(VARIABLES1, VARIABLES2, COMPARE, COUNT, LIMIT)
```  
**Arguments**  
- `VARIABLES1`: collection(var-dvar)
- `VARIABLES2`: collection(var-dvar)
- `COMPARE`: atom
- `COUNT`: atom
- `LIMIT`: dvar

**Restrictions**  
- `|VARIABLES1| = |VARIABLES2|`
- `required(VARIABLES1, var)`
- `required(VARIABLES2, var)`
- `COMPARE ∈ {=, ≠, <, ≥, >, ≤}`
- `COUNT ∈ {=, ≠, <, ≥, >, ≤}`
- `LIMIT ≥ 0`

**Purpose**  
Enforce the condition  
```plaintext
\( \sum_{i=1}^{|VARIABLES1|} VARIABLES1[i].var \) COMPARE VARIABLES2[i].var) COUNT LIMIT.
```  

**Example**  
```plaintext
(⟨4, 5, 5, 4, 5⟩, ⟨4, 2, 5, 1, 5⟩, =, ≤, 3)\)
```

The `compare_and_count` constraint holds since no more than \( LIMIT = 3 \) pairs of variables are equal, i.e., the first, third and fifth pairs.

**Typical**  
- `|VARIABLES1| > 1`
- `range(VARIABLES1.var) > 1`
- `range(VARIABLES2.var) > 1`
- `COMPARE ∈ [=]`
- `COUNT ∈ [=, <, ≥, >, ≤]`
- `LIMIT > 0`
- `LIMIT < |VARIABLES1|`

**Arg. properties**  
- **Contractible** wrt. `VARIABLES1` and `VARIABLES2` (remove items from same position) when `COUNT ∈ [<, ≤]`.
- **Extensible** wrt. `VARIABLES1` and `VARIABLES2` (add items at same position) when `COUNT ∈ [≥, >]`.

**See also**  
- **common keyword**: `count` (counting constraint).

**Keywords**  
- **constraint type**: predefined constraint, counting constraint.
5.72 cond_lex_cost

### Description

**Origin**

Inspired by [412].

**Constraint**

`cond_lex_cost(VECTOR, PREFERENCE_TABLE, COST)`

**Type**

`TUPLE_OF_VALS : collection(val−int)`

**Arguments**

- `VECTOR : collection(var−dvar)`
- `PREFERENCE_TABLE : collection(tuple − TUPLE_OF_VALS)`
- `COST : dvar`

**Restrictions**

- `|TUPLE_OF_VALS| ≥ 1`
- `required(TUPLE_OF_VALS, val)`
- `required(VECTOR, var)`
- `|VECTOR| = |TUPLE_OF_VALS|`
- `required(PREFERENCE_TABLE, tuple)`
- `same_size(PREFERENCE_TABLE, tuple)`
- `distinct(PREFERENCE_TABLE, [])`
- `in_relation(VECTOR, PREFERENCE_TABLE)`
- `COST ≥ 1`
- `COST ≤ |PREFERENCE_TABLE|`

**Purpose**

VECTOR is assigned to the CDST\textsuperscript{th} item of the collection PREFERENCE_TABLE.

**Example**

\[
\begin{pmatrix}
(0,1),
\text{tuple} = (1,0),
\text{tuple} = (0,1),
\text{tuple} = (0,0),
\text{tuple} = (1,1)
\end{pmatrix}, 2
\]

The cond_lex_cost constraint holds since VECTOR is assigned to the second item of the collection PREFERENCE_TABLE.

**Typical**

- `|TUPLE_OF_VALS| > 1`
- `|VECTOR| > 1`
- `|PREFERENCE_TABLE| > 1`

**Symmetries**

- Items of VECTOR and PREFERENCE_TABLE.tuple are permutable (same permutation used).
- All occurrences of two distinct tuples of values in VECTOR or PREFERENCE_TABLE.tuple can be swapped; all occurrences of a tuple of values in VECTOR or PREFERENCE_TABLE.tuple can be renamed to any unused tuple of values.
Usage

We consider an example taken from [412] where a customer has to decide among vacations. There are two seasons when he can travel (spring and summer) and two locations Naples and Helsinki. Furthermore assume that location is more important than season and the preferred period of the year depends on the selected location. The travel preferences of a customer are explicitly defined by stating the preferences ordering among the possible tuples of values \(\text{\langle Naples, spring\rangle, \langle Naples, summer\rangle, \langle Helsinki, spring\rangle, \langle Helsinki, summer\rangle}\). For instance we may state within the preference table \text{PREFERENCE\_TABLE} of the \text{cond\_lex\_cost} constraint the preference ordering \(\text{\langle Naples, spring\rangle} \succ \text{\langle Helsinki, summer\rangle} \succ \text{\langle Helsinki, spring\rangle} \succ \text{\langle Naples, summer\rangle}\), which denotes the fact that our customer prefers Naples in the spring and Helsinki in the summer, and a vacation in spring is preferred over summer. Finally a solution minimising the cost variable COST will match the preferences stated by our customer.

See also

attached to cost variant: in relation (COST parameter removed).

common keyword: cond\_lex\_greater, cond\_lex\_greatereq, cond\_lex\_less, cond\_lex\_lesseq (preferences).

specialisation: element (tuple of variables replaced by single variable).

Keywords

characteristic of a constraint: vector, automaton, automaton without counters, reified automaton constraint.

constraint network structure: Berge-acyclic constraint network.

constraint type: order constraint.

filtering: arc-consistency, cost filtering constraint.

modelling: preferences.

symmetry: lexicographic order.
Automaton

Figure 5.142 depicts the automaton associated with cond_lex_less_eq constraint. Let VAR_k denote the var attribute of the k^{th} item of the VECTOR collection. Figure 5.143 depicts the reformulation of the cond_lex_cost constraint.

Figure 5.142: Automaton of the cond_lex_cost constraint given in the example

Figure 5.143: Hypergraph of the reformulation corresponding to the automaton of the cond_lex_cost constraint
5.73  

cond_lex_greater

**DESCRIPTION**

- **Origin**: Inspired by [412].

- **Constraint**: 
  
  \[ \text{cond_lex_greater(VECTOR1, VECTOR2, PREFERENCE_TABLE)} \]

- **Type**: 
  
  \[ \text{TUPLE_OF_VALS : collection(val-int)} \]

- **Arguments**: 
  
  - VECTOR1 : \( \text{collection(var-dvar)} \)
  - VECTOR2 : \( \text{collection(var-dvar)} \)
  - PREFERENCE_TABLE : \( \text{collection(tuple - TUPLE_OF_VALS)} \)

- **Restrictions**: 
  
  \[ |\text{TUPLE_OF_VALS}| \geq 1 \]
  
  \[ \text{required(VECTOR1, var)} \]
  
  \[ \text{required(VECTOR2, var)} \]
  
  \[ |\text{VECTOR1}| = |\text{VECTOR2}| \]
  
  \[ |\text{VECTOR1}| = |\text{TUPLE_OF_VALS}| \]
  
  \[ \text{required(PREFERENCE_TABLE, tuple)} \]
  
  \[ \text{same_size(PREFERENCE_TABLE, tuple)} \]
  
  \[ \text{distinct(PREFERENCE_TABLE, [])} \]
  
  \[ \text{in_relation(VECTOR1, PREFERENCE_TABLE)} \]
  
  \[ \text{in_relation(VECTOR2, PREFERENCE_TABLE)} \]

- **Purpose**: 
  
  VECTOR1 and VECTOR2 are both assigned to the \( I^{th} \) and \( J^{th} \) items of the collection PREFERENCE_TABLE such that \( I > J \).

- **Example**: 
  
  \[
  \begin{pmatrix}
  (0,0), \\
  (1,0), \\
  \text{tuple} = (1,0), \\
  \text{tuple} = (0,1), \\
  \text{tuple} = (0,0), \\
  \text{tuple} = (1,1)
  \end{pmatrix}
  \]

  The \text{cond_lex_greater} constraint holds since VECTOR1 and VECTOR2 are respectively assigned to the third and first items of the collection PREFERENCE_TABLE.

- **Typical**: 
  
  \[ |\text{TUPLE_OF_VALS}| > 1 \]
  
  \[ |\text{VECTOR1}| > 1 \]
  
  \[ |\text{VECTOR2}| > 1 \]
  
  \[ |\text{PREFERENCE_TABLE}| > 1 \]
Symmetries

- Items of VECTOR1, VECTOR2 and PREFERENCE_TABLE.tuple are permutable (same permutation used).

- All occurrences of two distinct tuples of values in VECTOR1, VECTOR2 or PREFERENCE_TABLE.tuple can be swapped; all occurrences of a tuple of values in VECTOR1, VECTOR2 or PREFERENCE_TABLE.tuple can be renamed to any unused tuple of values.

Usage

See cond_lex_cost.

See also

| common keyword: | cond_lex_cost, cond_lex_greatereq, cond_lex_less, cond_lex_lesseq(preferences), lex_greater (lexicographic order). |
| implies: | cond_lex_greatereq. |

Keywords

| characteristic of a constraint: | vector, automaton. |
| constraint network structure: | Berge-acyclic constraint network. |
| constraint type: | order constraint. |
| filtering: | arc-consistency. |
| modelling: | preferences. |
| symmetry: | lexicographic order. |
Figure 5.144 depicts the automaton associated with the preference table of the `cond_lex_greater` constraint given in the example. Let \( \text{VAR}_1^k \) and \( \text{VAR}_2^k \) respectively be the \( \text{var} \) attributes of the \( k^{th} \) items of the \( \text{VECTOR}_1 \) and the \( \text{VECTOR}_2 \) collections. Figure 5.145 depicts the reformulation of the `cond_lex_greater` constraint. This reformulation uses:

- Two occurrences of the automaton depicted by Figure 5.144 for computing the positions \( I \) and \( J \) within the preference table corresponding to \( \text{VECTOR}_1 \) and \( \text{VECTOR}_2 \).
- The binary constraint \( I > J \).

Figure 5.144: Automaton associated with the preference table of the `cond_lex_greater` constraint given in the example

Figure 5.145: Hypergraph of the reformulation corresponding to the `cond_lex_greater` constraint: it uses two occurrences of the automaton of Figure 5.144 and the constraint \( I > J \)
5.74 cond_lex_greatereq

**DESCRIPTION**

**LINKS**

**AUTOMATON**

**Origin**

Inspired by [412].

**Constraint**

\[
\text{cond\_lex\_greatereq}(\text{VECTOR1}, \text{VECTOR2}, \text{PREFERENCE\_TABLE})
\]

**Type**

\[
\text{TUPLE\_OF\_VALS} : \text{collection}(\text{val\_int})
\]

**Arguments**

\[
\begin{align*}
\text{VECTOR1} & : \text{collection}(\text{var\_dvar}) \\
\text{VECTOR2} & : \text{collection}(\text{var\_dvar}) \\
\text{PREFERENCE\_TABLE} & : \text{collection}(\text{tuple\_TUPLE\_OF\_VALS})
\end{align*}
\]

**Restrictions**

\[
\begin{align*}
|\text{TUPLE\_OF\_VALS}| & \geq 1 \\
\text{required}(\text{TUPLE\_OF\_VALS}.\text{val}) \\
\text{required}(\text{VECTOR1}.\text{var}) \\
\text{required}(\text{VECTOR2}.\text{var}) \\
|\text{VECTOR1}| & = |\text{VECTOR2}| \\
|\text{VECTOR1}| & = |\text{TUPLE\_OF\_VALS}| \\
\text{required}(\text{PREFERENCE\_TABLE}.\text{tuple}) \\
\text{same\_size}(\text{PREFERENCE\_TABLE}.\text{tuple}) \\
\text{distinct}(\text{PREFERENCE\_TABLE}, []) \\
\text{in\_relation}(\text{VECTOR1}, \text{PREFERENCE\_TABLE}) \\
\text{in\_relation}(\text{VECTOR2}, \text{PREFERENCE\_TABLE})
\end{align*}
\]

**Purpose**

\[
\text{VECTOR1} \text{ and } \text{VECTOR2} \text{ are both assigned to the } I^{th} \text{ and } J^{th} \text{ items of the collection} \text{PREFERENCE\_TABLE} \text{ such that } I \geq J.
\]

**Example**

\[
\begin{pmatrix}
(0,0), \\
(1,0),
\end{pmatrix}
\begin{pmatrix}
tuple = (1,0), \\
tuple = (0,1), \\
tuple = (0,0), \\
tuple = (1,1)
\end{pmatrix}
\]

The \text{cond\_lex\_greatereq} constraint holds since \text{VECTOR1} and \text{VECTOR2} are respectively assigned to the third and first items of the collection \text{PREFERENCE\_TABLE}.

**Typical**

\[
\begin{align*}
|\text{TUPLE\_OF\_VALS}| & > 1 \\
|\text{VECTOR1}| & > 1 \\
|\text{VECTOR2}| & > 1 \\
|\text{PREFERENCE\_TABLE}| & > 1
\end{align*}
\]
Symmetries

- Items of \texttt{VECTOR1}, \texttt{VECTOR2} and \texttt{PREFERENCE\_TABLE\_tuple} are permutable (same permutation used).
- All occurrences of two distinct tuples of values in \texttt{VECTOR1}, \texttt{VECTOR2} or \texttt{PREFERENCE\_TABLE\_tuple} can be swapped; all occurrences of a tuple of values in \texttt{VECTOR1}, \texttt{VECTOR2} or \texttt{PREFERENCE\_TABLE\_tuple} can be renamed to any unused tuple of values.

Usage

See \texttt{cond\_lex\_cost}.

See also

\texttt{cond\_lex\_cost}, \texttt{cond\_lex\_greater}, \texttt{cond\_lex\_less}, \texttt{cond\_lex\_lesseq(preferences)}, \texttt{lex\_greater\_eq(\textit{lexicographic order})}.

implied by: \texttt{cond\_lex\_greater}.

Keywords

\texttt{characteristic of a constraint}: vector, automaton.
\texttt{constraint network structure}: Berge-acyclic constraint network.
\texttt{constraint type}: order constraint.
\texttt{filtering}: arc-consistency.
\texttt{modelling}: preferences.
\texttt{symmetry}: lexicographic order.
Automaton

Figure 5.146 depicts the automaton associated with the preference table of the **cond_lex.greatereq** constraint given in the example. Let **VAR1_k** and **VAR2_k** respectively be the **var** attributes of the **k**\(^{th}\) items of the **VECTOR1** and the **VECTOR2** collections. Figure 5.147 depicts the reformulation of the **cond_lex.greatereq** constraint. This reformulation uses:

- Two occurrences of the automaton depicted by Figure 5.146 for computing the positions **I** and **J** within the preference table corresponding to **VECTOR1** and **VECTOR2**.
- The binary constraint **I ≥ J**.

---

**Figure 5.146:** Automaton associated with the preference table of the **cond_lex.greatereq** constraint given in the example

**Figure 5.147:** Hypergraph of the reformulation corresponding to the **cond_lex.greatereq** constraint: it uses two occurrences of the automaton of Figure 5.146 and the constraint **I ≥ J**
5.75 cond_lex_less

**Description**

Inspired by [412].

**Constraint**

\[ \text{cond_lex_less}(\text{VECTOR1}, \text{VECTOR2}, \text{PREFERENCE\_TABLE}) \]

**Type**

\[ \text{TUPLE\_OF\_VALS} : \text{collection(val-int)} \]

**Arguments**

- \( \text{VECTOR1} \) : \( \text{collection(var-dvar)} \)
- \( \text{VECTOR2} \) : \( \text{collection(var-dvar)} \)
- \( \text{PREFERENCE\_TABLE} \) : \( \text{collection(tuple-TUPLE\_OF\_VALS)} \)

**Restrictions**

- \( |\text{TUPLE\_OF\_VALS}| \geq 1 \)
- \( \text{required}(\text{TUPLE\_OF\_VALS}, \text{val}) \)
- \( \text{required}(\text{VECTOR1}, \text{var}) \)
- \( \text{required}(\text{VECTOR2}, \text{var}) \)
- \( |\text{VECTOR1}| = |\text{VECTOR2}| \)
- \( |\text{VECTOR1}| = |\text{TUPLE\_OF\_VALS}| \)
- \( \text{required}(\text{PREFERENCE\_TABLE}, \text{tuple}) \)
- \( \text{same\_size}(\text{PREFERENCE\_TABLE}, \text{tuple}) \)
- \( \text{distinct}(\text{PREFERENCE\_TABLE}, []) \)
- \( \text{in\_relation}(\text{VECTOR1}, \text{PREFERENCE\_TABLE}) \)
- \( \text{in\_relation}(\text{VECTOR2}, \text{PREFERENCE\_TABLE}) \)

**Purpose**

\( \text{VECTOR1} \) and \( \text{VECTOR2} \) are both assigned to the \( I^{th} \) and \( J^{th} \) items of the collection \( \text{PREFERENCE\_TABLE} \) such that \( I < J \).

**Example**

\[
\langle (1,0), (0,0) \rangle,
\text{tuple} = (1,0),
\text{tuple} = (0,1),
\text{tuple} = (0,0),
\text{tuple} = (1,1)
\]

The \( \text{cond\_lex\_less} \) constraint holds since \( \text{VECTOR1} \) and \( \text{VECTOR2} \) are respectively assigned to the first and third items of the collection \( \text{PREFERENCE\_TABLE} \).

**Typical**

- \( |\text{TUPLE\_OF\_VALS}| > 1 \)
- \( |\text{VECTOR1}| > 1 \)
- \( |\text{VECTOR2}| > 1 \)
- \( |\text{PREFERENCE\_TABLE}| > 1 \)
Symmetries

- Items of VECTOR1, VECTOR2 and PREFERENCE_TABLE.tuple are permutable (same permutation used).
- All occurrences of two distinct tuples of values in VECTOR1, VECTOR2 or PREFERENCE_TABLE.tuple can be swapped; all occurrences of a tuple of values in VECTOR1, VECTOR2 or PREFERENCE_TABLE.tuple can be renamed to any unused tuple of values.

Usage

See cond_lex_cost.

See also

common keyword: cond_lex_cost, cond_lex_greater, cond_lex_greatereq, cond_lex_lessseq(preferences), lex_less(lexicographic order).
implies: cond_lex_lessseq.

Keywords

characteristic of a constraint: vector, automaton.
constraint network structure: Berge-acyclic constraint network.
constraint type: order constraint.
filtering: arc-consistency.
modelling: preferences.
symmetry: lexicographic order.
Automaton

Figure 5.148 depicts the automaton associated with the preference table of the $\text{cond\_lex\_less}$ constraint given in the example. Let $\text{VAR}_1^k$ and $\text{VAR}_2^k$ respectively be the var attributes of the $k^{th}$ items of the $\text{VECTOR}_1$ and the $\text{VECTOR}_2$ collections. Figure 5.149 depicts the reformulation of the $\text{cond\_lex\_less}$ constraint. This reformulation uses:

- Two occurrences of the automaton depicted by Figure 5.148 for computing the positions $I$ and $J$ within the preference table corresponding to $\text{VECTOR}_1$ and $\text{VECTOR}_2$.
- The binary constraint $I < J$.

![Automaton Diagram](image)

Figure 5.148: Automaton associated with the preference table of the $\text{cond\_lex\_less}$ constraint given in the example

![Reformulation Diagram](image)

Figure 5.149: Hypergraph of the reformulation corresponding to the $\text{cond\_lex\_less}$ constraint: it uses two occurrences of the automaton of Figure 5.148 and the constraint $I < J$
### 5.76 cond_lex_lesseq

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Inspired by [412].</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>$\text{cond}<em>\text{lex}</em>\text{lesseq}(\text{VECTOR1, VECTOR2, PREFERENCE_TABLE})$</td>
<td></td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>$\text{TUPLE_OF_VALS} : \text{collection}(\text{val-int})$</td>
<td></td>
</tr>
</tbody>
</table>
| **Arguments** | \begin{align*} \text{VECTOR1} & : \text{collection}(\text{var-dvar}) \\
| VECTOR2 & : \text{collection}(\text{var-dvar}) \\
| PREFERENCE\_TABLE & : \text{collection}(\text{tuple-TUPLE\_OF\_VALS}) \end{align*} |          |
| **Restrictions** | \begin{align*} |\text{TUPLE\_OF\_VALS}| & \geq 1 \\
| \text{required}(\text{TUPLE\_OF\_VALS}\_\text{val}) & \\
| \text{required}(\text{VECTOR1}\_\text{var}) & \\
| \text{required}(\text{VECTOR2}\_\text{var}) & \\
| |\text{VECTOR1}| & = |\text{VECTOR2}| \\
| |\text{VECTOR1}| & = |\text{TUPLE\_OF\_VALS}| \\
| \text{required}(\text{PREFERENCE\_TABLE}\_\text{tuple}) & \\
| \text{same\_size}(\text{PREFERENCE\_TABLE}\_\text{tuple}) & \\
| \text{distinct}(\text{PREFERENCE\_TABLE}, [\]) & \\
| \text{in\_relation}(\text{VECTOR1}, \text{PREFERENCE\_TABLE}) & \\
| \text{in\_relation}(\text{VECTOR2}, \text{PREFERENCE\_TABLE}) & \end{align*} |          |
| **Purpose** | \text{VECTOR1 and VECTOR2 are both assigned to the I\textsuperscript{th} and J\textsuperscript{th} items of the collection PREFERENCE\_TABLE such that I \leq J.} |          |
| **Example** | $\begin{pmatrix} (1,0), \\
| (0,0), \\
| \text{tuple} = (1,0), \\
| \text{tuple} = (0,1), \\
| \text{tuple} = (0,0), \\
| \text{tuple} = (1,1) \end{pmatrix}$ |          |
| **Typical** | $|\text{TUPLE\_OF\_VALS}| > 1 \\
| |\text{VECTOR1}| > 1 \\
| |\text{VECTOR2}| > 1 \\
| |\text{PREFERENCE\_TABLE}| > 1$ |          |

The $\text{cond}_\text{lex}_\text{lesseq}$ constraint holds since $\text{VECTOR1}$ and $\text{VECTOR2}$ are respectively assigned to the first and third items of the collection $\text{PREFERENCE\_TABLE}$. 
Symmetries

- Items of VECTOR1, VECTOR2 and PREFERENCE_TABLE.tuple are permutable (same permutation used).
- All occurrences of two distinct tuples of values in VECTOR1, VECTOR2 or PREFERENCE_TABLE.tuple can be swapped; all occurrences of a tuple of values in VECTOR1, VECTOR2 or PREFERENCE_TABLE.tuple can be renamed to any unused tuple of values.

Usage

See cond_lex_cost.

See also

common keyword: cond_lex_cost, cond_lex_greater, cond_lex_greatereq, cond_lex_less(preferences), lex_lesseq(lexicographic order).

implied by: cond_lex_less.

Keywords

characteristic of a constraint: vector, automaton.
constraint network structure: Berge-acyclic constraint network.
constraint type: order constraint.
filtering: arc-consistency.
modelling: preferences.
symmetry: lexicographic order.
Automaton

Figure 5.150 depicts the automaton associated with the preference table of the cond_lex_lesseq constraint given in the example. Let $VAR_{1k}$ and $VAR_{2k}$ respectively be the var attributes of the $k^{th}$ items of the VECTOR1 and the VECTOR2 collections. Figure 5.151 depicts the reformulation of the cond_lex_lesseq constraint. This reformulation uses:

- Two occurrences of the automaton depicted by Figure 5.150 for computing the positions $I$ and $J$ within the preference table corresponding to VECTOR1 and VECTOR2.
- The binary constraint $I \leq J$.

![Automaton Diagram](image)

Figure 5.150: Automaton associated with the preference table of the cond_lex_lesseq constraint given in the example

![Hypergraph Diagram](image)

Figure 5.151: Hypergraph of the reformulation corresponding to the cond_lex_lesseq constraint: it uses two occurrences of the automaton of Figure 5.150 and the constraint $I \leq J$
5.77 connect_points

**DESCRIPTION**

**LINKS**

**GRAPH**

**Origin**
N. Beldiceanu

**Constraint**
connect_points(SIZE1, SIZE2, SIZE3, NGROUP, POINTS)

**Arguments**

<table>
<thead>
<tr>
<th>SIZE1</th>
<th>int</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIZE2</td>
<td>int</td>
</tr>
<tr>
<td>SIZE3</td>
<td>int</td>
</tr>
<tr>
<td>NGROUP</td>
<td>dvar</td>
</tr>
<tr>
<td>POINTS</td>
<td>collection(p-dvar)</td>
</tr>
</tbody>
</table>

**Restrictions**

<table>
<thead>
<tr>
<th>SIZE1</th>
<th>&gt; 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIZE2</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>SIZE3</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>NGROUP</td>
<td>≥ 0</td>
</tr>
<tr>
<td>NGROUP</td>
<td>≤</td>
</tr>
<tr>
<td>SIZE1 + SIZE2 + SIZE3</td>
<td>=</td>
</tr>
<tr>
<td>required(POINTS,p)</td>
<td></td>
</tr>
</tbody>
</table>

**Purpose**

On a 3-dimensional grid of variables, number of groups, where a group consists of a connected set of variables that all have a same value distinct from 0.
Figure 5.152 corresponds to the solution where we describe separately each layer of the grid. The `connect_points` constraint holds since we have two groups (`NGROUP = 2`): a first one for the variables of the `POINTS` collection assigned to value 1, and a second one for the variables assigned to value 2.

Figure 5.152: The two layers of the solution
Typical

SIZE1 > 1
SIZE2 > 1
NGROUP > 0
NGROUP < |POINTS|
|POINTS| > 3

Symmetry

All occurrences of two distinct values of POINTS_p that are both different from 0 can be swapped; all occurrences of a value of POINTS_p that is different from 0 can be renamed to any unused value that is also different from 0.

Arg. properties

Functional dependency: NGROUP determined by SIZE1, SIZE2, SIZE3 and POINTS.

Usage

Wiring problems [361], [424].

Algorithm

Since the graph corresponding to the 3-dimensional grid is symmetric one could certainly use as a starting point the filtering algorithm associated with the number of connected components graph property described in [50] (see the paragraphs “Estimating NCC” and “Estimating NCC”). One may also try to take advantage of the fact that the considered initial graph is a grid in order to simplify the previous filtering algorithm.

Keywords

characteristic of a constraint: joker value.
final graph structure: strongly connected component, symmetric.
geometry: geometrical constraint.
modelling: functional dependency.
problems: channel routing.
Figure 5.153 gives the initial graph constructed by the *GRID* arc generator associated with the *Example* slot.
5.78 connected

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[131]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>connected(NODES)</td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>NODES : collection(index-int, succ-svar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(NODES,[index, succ])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.index ≥ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.index ≤</td>
<td>NODES</td>
</tr>
<tr>
<td></td>
<td>distinct(NODES, index)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.succ ≥ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.succ ≤</td>
<td>NODES</td>
</tr>
<tr>
<td>Purpose</td>
<td>Consider a digraph $G$ described by the NODES collection. Select a subset of arcs of $G$ so that the corresponding graph is symmetric (i.e., if there is an arc from $i$ to $j$, there is also an arc from $j$ to $i$) and connected (i.e., there is a path between any pair of vertices of $G$).</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>The connected constraint holds since the NODES collection depicts a symmetric graph involving one single connected component.</td>
<td></td>
</tr>
<tr>
<td>Typical</td>
<td>$</td>
<td>\text{NODES}</td>
</tr>
<tr>
<td>Symmetry</td>
<td>Items of NODES are permutable.</td>
<td></td>
</tr>
<tr>
<td>Algorithm</td>
<td>A filtering algorithm for the connected constraint is sketched in [131, page 88]. Beside the pruning associated with the fact that the final graph is symmetric, it is based on the fact that all bridges and cutvertices on a path between two vertices that should for sure belong to the final graph should also belong to the final graph.</td>
<td></td>
</tr>
<tr>
<td>See also</td>
<td>common keyword: symmetric(symmetric).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>implies: strongly_connected.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>used in graph description: in_set.</td>
<td></td>
</tr>
<tr>
<td>Keywords</td>
<td>constraint arguments: constraint involving set variables.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>constraint type: graph constraint.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>final graph structure: connected component, symmetric.</td>
<td></td>
</tr>
</tbody>
</table>
Arc input(s) NODES
Arc generator \( CLIQUE \mapsto \text{collection}(\text{nodes1, nodes2}) \)
Arcarity 2
Arc constraint(s) \( \text{in\_set}(\text{nodes2.index, nodes1.succ}) \)
Graph property(ies) NCC = 1
Graph class SYMMETRIC

Graph model
Part (A) of Figure 5.154 shows the initial graph from which we start. It is derived from the set associated with each vertex. Each set describes the potential values of the \text{succ} attribute of a given vertex. Part (B) of Figure 5.154 gives the final graph associated with the Example slot.

Figure 5.154: Initial and final graph of the connected set constraint
### 5.79 consecutive_groups_of_ones

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from <code>group</code></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td><code>consecutive_groups_of_ones(GROUP_SIZES, VARIABLES)</code></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td><code>GROUP_SIZES : collection(nb-int)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>VARIABLES : collection(var-dvar)</code></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td><code>required(GROUP_SIZES, nb)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>`</td>
<td>GROUP_SIZES</td>
</tr>
<tr>
<td></td>
<td><code>GROUP_SIZES.nb ≥ 1</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>`GROUP_SIZES.nb ≤</td>
<td>VARIABLES</td>
</tr>
<tr>
<td></td>
<td><code>required(VARIABLES, var)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>`</td>
<td>VARIABLES</td>
</tr>
<tr>
<td></td>
<td>`</td>
<td>VARIABLES</td>
</tr>
<tr>
<td></td>
<td><code>VARIABLES.var ≥ 0</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>VARIABLES.var ≤ 1</code></td>
<td></td>
</tr>
</tbody>
</table>

In order to define the meaning of the `consecutive_groups_of_ones` constraint, we first introduce the notions of stretch and span. Let \( n \) be the number of variables of the collection \( \text{VARIABLES} \) and let \( m \) be the number of items of the collection \( \text{GROUP_SIZES} \). Let \( X_1, \ldots, X_j \) \((1 ≤ i ≤ j ≤ n)\) be consecutive variables of the collection of variables \( \text{VARIABLES} \) such that the following conditions apply:

- All variables \( X_i, \ldots, X_j \) are assigned value 1,
- \( i = 1 \) or \( X_{i-1} \neq 1 \),
- \( j = n \) or \( X_{j+1} \neq 1 \).

We call such a set of variables a stretch. The span of the stretch is equal to \( j - i + 1 \). We now define the condition enforced by the `consecutive_groups_of_ones` constraint.

All variables of the \( \text{VARIABLES} \) collection should be assigned value 0 or 1. In addition there is \(|\text{GROUP_SIZES}|\) successive stretches of respective span \( \text{GROUP_SIZES}[1], \text{nb}, \) \( \text{GROUP_SIZES}[2], \text{nb}, \ldots, \text{GROUP_SIZES}[m], \text{nb} \).

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\langle 2, 1 \rangle,)</td>
</tr>
<tr>
<td>(\text{var} - 1,)</td>
</tr>
<tr>
<td>(\text{var} - 1,)</td>
</tr>
<tr>
<td>(\langle \text{var} - 0,)</td>
</tr>
<tr>
<td>(\text{var} - 0,)</td>
</tr>
<tr>
<td>(\text{var} - 0,)</td>
</tr>
<tr>
<td>(\text{var} - 0,)</td>
</tr>
<tr>
<td>(\text{var} - 1,)</td>
</tr>
<tr>
<td>(\text{var} - 0)</td>
</tr>
</tbody>
</table>

The `consecutive_groups_of_ones` constraint holds since the sequence 1 1 0 0 0 1 0
contains a first stretch (i.e., a maximum sequence of 1) of span 2 and a second stretch of span 1.

Typical

\begin{align*}
|\text{VARIABLES}| & > 1 \\
\text{range(}\text{VARIABLES}.\text{var}) & > 1
\end{align*}

Symmetry

Items of GROUP_SIZES and VARIABLES are simultaneously reversible.

Usage

The consecutive_groups_of_ones constraint can be used in order to model the logigraphe problem.

See also

root concept: group.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.
constraint network structure: Berge-acyclic constraint network.
filtering: arc-consistency.
modelling exercises: logigraphe.
puzzles: logigraphe.
Automaton

Figure 5.155 depicts the automaton associated with the `consecutive_groups_of_ones` constraint. To each variable \( \text{VAR}_i \) of the collection \( \text{VARIABLES} \) corresponds a signature variable that is equal to \( \text{VAR}_i \). There is no signature constraint.

Figure 5.155: Automaton of the `consecutive_groups_of_ones` constraint of the Example slot

Figure 5.156: Hypergraph of the reformulation corresponding to the automaton of the `consecutive_groups_of_ones` constraint of the Example slot
5.80 consecutive_values

### DESCRIPTION

**Origin**
Derived from `alldifferent_consecutive_values`.

**Constraint**
`consecutive_values(VARIABLES)`

**Argument**
`VARIABLES : collection(var−dvar)`

**Restriction**
`required(VARIABLES.var)`

**Purpose**
Constraint the difference between the largest and the smallest values of the `VARIABLES` collection to be equal to the number of distinct values assigned to the variables of the `VARIABLES` collection minus one (i.e., there is no holes at all within the used values).

**Example**

```
((5, 4, 3, 5))
```

The `consecutive_values` constraint holds since all values between value 3 and value 5 are effectively used.

**Typical**

```
|VARIABLES| > 1
range(VARIABLES.var) > 1
```

**Symmetries**

- Items of `VARIABLES` are `permutable`.
- One and the same constant can be `added` to the `var` attribute of all items of `VARIABLES`.

**See also**

- implied by: `all_equal, alldifferent_consecutive_values, global_contiguity`
- used in reformulation: `nvalue`

**Keywords**

- characteristic of a constraint: sort based reformulation.
- constraint type: value constraint, predefined constraint.
## 5.81 contains_sboxes

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>LOGIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Geometry, derived from [318]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>contains_sboxes(K, DIMS, OBJECTS, SBOXES)</td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td>contains.</td>
<td></td>
</tr>
<tr>
<td>Types</td>
<td>VARIABLES : (\text{collection}(v - \text{dvar}))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTEGERS : (\text{collection}(v - \text{int}))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>POSITIVES : (\text{collection}(v - \text{int}))</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>(K : \text{int})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{DIMS} : \text{sint})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{OBJECTS} : \text{collection}(\text{oid} - \text{int}, \text{sid} - \text{int}, x - \text{VARIABLES}))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{SBOXES} : \text{collection}(\text{sid} - \text{int}, t - \text{INTEGERS}, l - \text{POSITIVES}))</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>(</td>
<td>\text{VARIABLES}</td>
</tr>
<tr>
<td></td>
<td>(</td>
<td>\text{INTEGERS}</td>
</tr>
<tr>
<td></td>
<td>(</td>
<td>\text{POSITIVES}</td>
</tr>
<tr>
<td></td>
<td>(\text{required(VARIABLES,v)})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{required(INTEGERS,v)})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{required(POSITIVES,v)})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{POSITIVES} = K)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{POSITIVES}.v &gt; 0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(K &gt; 0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{DIMS} \geq 0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{DIMS} &lt; K)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{increasing seq(OBJECTS, [oid])})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{required(OBJECTS, [oid, sid, x])})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{OBJECTS.oid} \geq 1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{OBJECTS.oid} \leq</td>
<td>\text{OBJECTS}</td>
</tr>
<tr>
<td></td>
<td>(\text{OBJECTS.sid} \geq 1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{OBJECTS.sid} \leq</td>
<td>\text{SBOXES}</td>
</tr>
<tr>
<td></td>
<td>(\text{SBOXES} \geq 1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{required(SBOXES, [sid, t, l])})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{SBOXES.sid} \geq 1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{SBOXES.sid} \leq</td>
<td>\text{SBOXES}</td>
</tr>
<tr>
<td></td>
<td>(\text{do not overlap}(\text{SBOXES}))</td>
<td></td>
</tr>
</tbody>
</table>
Holds if, for each pair of objects $\left( O_i, O_j \right)$, $i < j$, $O_i$ contains $O_j$ with respect to a set of dimensions depicted by $\text{DIMS}$. $O_i$ and $O_j$ are objects that take a shape among a set of shapes. Each shape is defined as a finite set of shifted boxes, where each shifted box is described by a box in a $K$-dimensional space at a given offset (from the origin of the shape) with given sizes. More precisely, a shifted box is an entity defined by its shape id $\text{sid}$, shift offset $\text{t}$, and sizes $l$. Then, a shape is defined as the union of shifted boxes sharing the same shape id. An object is an entity defined by its unique object identifier $\text{oid}$, shape id $\text{sid}$ and origin $x$.

An object $O_i$ contains an object $O_j$ with respect to a set of dimensions depicted by $\text{DIMS}$ if and only if, for all shifted boxes $s_j$ associated with $O_j$, there exists a shifted box $s_i$ of $O_i$ such that $s_i$ contains $s_j$. A shifted box $s_i$ contains a shifted box $s_j$ if and only if, for all dimensions $d \in \text{DIMS}$, (1) the start of $s_j$ in dimension $d$ is strictly less than the start of $s_i$ in dimension $d$ and (2) the end of $s_j$ in dimension $d$ is strictly less than the end of $s_i$ in dimension $d$.

Figure 5.157 shows the objects of the example. Since $O_1$ contains both $O_2$ and $O_3$, and since $O_2$ contains $O_3$, the contains_sboxes constraint holds.

Figure 5.157: The three objects of the example

Typical $|\text{OBJECTS}| > 1$
Symmetries

- Items of SBOXES are permutable.
- Items of OBJECTS.x, SBOXES.t and SBOXES.l are permutable (same permutation used).

Arg. properties

Suffix-contractible wrt. OBJECTS.

Remark

One of the eight relations of the Region Connection Calculus [318]. The constraint contains_sboxes is a restriction of the original relation since it requires that each shifted box of an object is contained by one shifted box of the other object.

See also

common keyword: coveredby_sboxes, covers_sboxes, disjoint_sboxes, equal_sboxes, inside_sboxes, meet_sboxes(rcc8), non_overlap_sboxes (geometrical constraint,logic), overlap_sboxes (rcc8).

Keywords

constraint type: logic.
geometry: geometrical constraint, rcc8.
Logic

- \text{origin}(O_1, S_1, D) \overset{\text{def}}{=} O_1.x(D) + S_1.t(D)
- \text{end}(O_1, S_1, D) \overset{\text{def}}{=} O_1.x(D) + S_1.t(D) + S_1.1(D)
- \text{contains_sboxes}(\text{Dims}, O_1, S_1, O_2, S_2) \overset{\text{def}}{=} \forall D \in \text{Dims}
  \left( \begin{array}{c}
  \text{origin}(O_1, S_1, D) < \\
  \text{origin}(O_2, S_2, D) < \\
  \text{end}(O_2, S_2, D) < \\
  \text{end}(O_1, S_1, D)
  \end{array} \right)
- \text{contains_objects}(\text{Dims}, O_1, O_2) \overset{\text{def}}{=} \forall S_1 \in \text{sboxes}([O_1.\text{sid}])
  \exists S_2 \in \text{sboxes}([O_2.\text{sid}])
  \text{contains_sboxes}(\text{Dims}, O_1, S_1, O_2, S_2)
- \text{all_contains}(\text{Dims}, \text{OIDS}) \overset{\text{def}}{=} \forall O_1 \in \text{objects}(\text{OIDS})
  \forall O_2 \in \text{objects}(\text{OIDS})
  \begin{array}{c}
  O_1.\text{oid} < \\
  02.\text{oid}
  \end{array} \Rightarrow 
  \text{contains_objects}(\text{Dims}, O_1, O_2)
- \text{all_contains}($\text{DIMENSIONS, OIDS}$)
5.82 correspondence

**DESCRIPTION**

Origin

Derived from sort_permutation by removing the sorting condition.

Constraint

correspondence(FROM, PERMUTATION, TO)

Arguments

<table>
<thead>
<tr>
<th>FROM</th>
<th>collection(from-dvar)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERMUTATION</td>
<td>collection(var-dvar)</td>
</tr>
<tr>
<td>TO</td>
<td>collection(tvar-dvar)</td>
</tr>
</tbody>
</table>

Restrictions

\[|\text{PERMUTATION}| = |\text{FROM}|\]
\[|\text{PERMUTATION}| = |\text{TO}|\]
\[\text{PERMUTATION}.\text{var} \geq 1\]
\[\text{PERMUTATION}.\text{var} \leq |\text{PERMUTATION}|\]
\[\text{alldifferent}(|\text{PERMUTATION}|)\]
\[\text{required}(\text{FROM}, \text{from})\]
\[\text{required}(\text{PERMUTATION}, \text{var})\]
\[\text{required}(\text{TO}, \text{tvar})\]

Purpose

The variables of collection FROM correspond to the variables of collection TO according to the permutation PERMUTATION (i.e., FROM[i].from = TO[PERMUTATION[i]].var.tvar).

Example

\[
\begin{bmatrix}
\text{from} - 1, \\
\text{from} - 9, \\
\text{from} - 1, \\
\text{from} - 5, \\
\text{from} - 2, \\
\text{from} - 1 \\
\text{var} - 6, \\
\text{var} - 1, \\
\text{var} - 3, \\
\text{var} - 5, \\
\text{var} - 4, \\
\text{var} - 2 \\
\text{tvar} - 9, \\
\text{tvar} - 1, \\
\text{tvar} - 1, \\
\text{tvar} - 2, \\
\text{tvar} - 5, \\
\text{tvar} - 1
\end{bmatrix}
\]

As illustrated by Figure 5.158, the correspondence constraint holds since:

- The first item FROM[1].from = 1 of collection FROM corresponds to the PERMUTATION[1].var = 6th item of collection TO.
- The second item FROM[2].from = 9 of collection FROM corresponds to the PERMUTATION[2].var = 1\textsuperscript{st} item of collection TO.
- The third item FROM[3].from = 1 of collection FROM corresponds to the PERMUTATION[3].var = 3\textsuperscript{rd} item of collection TO.
- The fourth item FROM[4].from = 5 of collection FROM corresponds to the PERMUTATION[4].var = 5\textsuperscript{th} item of collection TO.
- The fifth item FROM[5].from = 2 of collection FROM corresponds to the PERMUTATION[5].var = 4\textsuperscript{th} item of collection TO.
- The sixth item FROM[6].from = 1 of collection FROM corresponds to the PERMUTATION[6].var = 2\textsuperscript{nd} item of collection TO.

![Figure 5.158: Illustration of the correspondence between the items of the FROM and the TO collections according to the permutation defined by the items of the PERMUTATION collection](image)

<table>
<thead>
<tr>
<th>Typical</th>
<th>FROM &gt; 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>range(FROM.from) &gt; 1</td>
<td></td>
</tr>
</tbody>
</table>

**Symmetry**

All occurrences of two distinct values in FROM.from or TO.tvar can be swapped; all occurrences of a value in FROM.from or TO.tvar can be renamed to any unused value.

**Remark**

Similar to the same constraint except that we also provide the permutation that allows to go from the items of collection FROM to the items of collection TO.

**See also**

implied by: sort permutation.
specialisation: same (PERMUTATION parameter removed).

**Keywords**

characteristic of a constraint: derived collection.
combinatorial object: permutation.
constraint arguments: constraint between three collections of variables.
final graph structure: acyclic, bipartite, no loop.
Derived Collection

\[
\text{col}(\text{FROM\_PERMUTATION}\rightarrow\text{collection(from-dvar, var-dvar)}, \\
\text{[item(from - FROM.from, var - PERMUTATION.var])})
\]

Arc input(s)
FROM\_PERMUTATION TO

Arc generator
\(PRODUCT\rightarrow\text{collection(from_permutation, to)}\)

Arc arity
2

Arc constraint(s)
- from_permutation.from = to.tvar
- from_permutation.var = to.key

Graph property(ies)
\(\text{NARC} = |\text{PERMUTATION}|\)

Graph class
- ACYCLIC
- BIPARTITE
- NO_LOOP

Graph model
Parts (A) and (B) of Figure 5.159 respectively show the initial and final graph associated with the Example slot. In both graphs the source vertices correspond to the derived collection FROM\_PERMUTATION, while the sink vertices correspond to the collection TO. Since the final graph contains exactly \(|\text{PERMUTATION}|\) arcs the correspondence constraint holds. As we use the \(\text{NARC}\) graph property, the arcs of the final graph are stressed in bold.

Signature
Because of the second condition from_permutation.var = to.key of the arc constraint and since both, the var attributes of the collection FROM\_PERMUTATION and the key attributes of the collection TO are all-distinct, the final graph contains at most \(|\text{PERMUTATION}|\) arcs. Therefore we can rewrite the graph property \(\text{NARC} = |\text{PERMUTATION}|\) to \(\text{NARC} \geq |\text{PERMUTATION}|\). This leads to simplify \(\text{NARC}\) to \(\text{NARC}\).
Figure 5.159: Initial and final graph of the correspondence constraint
5.83 count

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[94]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>count(VALUE, VARIABLES, RELOP, LIMIT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>occurrence,occurrence.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VALUE : int</td>
<td>VARIABLES : collection(var−dvar)</td>
<td>RELOP : atom</td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(VARIABLES,var)</td>
<td>RELOP ∈ [=, ≠, &lt;, ≥, &gt;, ≤]</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>Let $N$ be the number of variables of the VARIABLES collection assigned to value VALUE; Enforce condition $N$ RELOP LIMIT to hold.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>( 5, ⟨4,5,5,4,5⟩, ≥, 2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Typical</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symmetries</td>
<td>● Items of VARIABLES are permutable.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arg. properties</td>
<td>● Contractible wrt. VARIABLES when RELOP ∈ [&lt;, =].</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remark</td>
<td>Similar to the among constraint. Both, in JaCoP (<a href="http://www.jacop.eu/">http://www.jacop.eu/</a>) and in MiniZinc (<a href="http://www.g12.cs.mu.oz.au/minizinc/">http://www.g12.cs.mu.oz.au/minizinc/</a>) RELOP is implicitly set to =.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The count constraint holds since value VALUE = 5 occurs 3 times within the items of the collection VARIABLES = ⟨4, 5, 5, 4, 5⟩, which is greater than or equal to (RELOP is set to ≥) LIMIT = 2.
Reformulation
The count(VALUE, VARIABLES, RELOP, LIMIT) constraint can be expressed in terms of the conjunction \( \text{among}(N, \text{VARIABLES}, \{\text{VALUE}\}) \land N \text{ RELOP LIMIT} \).

Systems
occurrence in Choco, count in Gecode, count in JaCoP, count in MiniZinc, count in SICStus.

See also
assignment dimension added: assign_and_counts(variable=VALUE replaced by variable \( \in \) VALUES and assignment dimension introduced).
common keyword: among (value constraint, counting constraint),
arith (value constraint), compare_and_count (counting constraint),
global_cardinality, max_nvalue, min_nvalue (value constraint, counting constraint),
nvalue (counting constraint).
generalisation: counts (variable=VALUE replaced by variable \( \in \) VALUES).
related: roots.
used in reformulation: among.

Keywords
characteristic of a constraint: automaton, automaton with counters.
constraint network structure: alpha-acyclic constraint network(2).
constraint type: value constraint, counting constraint.
filtering: arc-consistency.
Graph model

Parts (A) and (B) of Figure 5.160 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the loops of the final graph are stressed in bold.

Figure 5.160: Initial and final graph of the count constraint
**Automaton**  

Figure 5.161 depicts the automaton associated with the count constraint. To each variable \( \text{VAR}_i \) of the collection \( \text{VARIABLES} \) corresponds a 0-1 signature variable \( S_i \). The following signature constraint links \( \text{VAR}_i \) and \( S_i \): \( \text{VAR}_i = \text{VALUE} \Leftrightarrow S_i \).

\[
\begin{align*}
\{ \text{C=0} \} & \quad \text{VAR}_i = \text{VALUE} \\
\{ \text{C=C+1} \} & \quad \text{C \ RELOP \ LIMIT} \\
\{ \text{C=0} \} & \quad \text{VAR}_i \Leftrightarrow \text{VALUE}
\end{align*}
\]

**Figure 5.161: Automaton of the count constraint**

Figure 5.162: Hypergraph of the reformulation corresponding to the automaton of the count constraint
## 5.84 counts

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Derived from count.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>counts(VALUE, VARIABLE, REL, LIMIT)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VALUES : collection(val-int)</td>
<td>VARIABLES : collection(var-dvar)</td>
<td>REL : atom</td>
<td>LIMIT : dvar</td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>required(VALUE, val)</td>
<td>distinct(VALUE, val)</td>
<td>required(VARIABLE, var)</td>
<td>REL ∈ {=, ≠, &lt;, ≥, &gt;, ≤}</td>
</tr>
<tr>
<td>Purpose</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Let (N) be the number of variables of the VARIABLE collection assigned to a value of the VALUE collection. Enforce condition (N \text{ REL} \text{ LIMIT}) to hold.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| \[
\begin{pmatrix}
1, 3, 4, 9, \\
\text{var} - 4, \\
\text{var} - 5, \\
\text{var} - 5, \\
\text{var} - 4, \\
\text{var} - 1, \\
\text{var} - 5
\end{pmatrix}, =, 3
\] |       |       |           |
| Values 1, 3, 4 and 9 of the VALUE collection are assigned to 3 items of the VARIABLE = \{4, 5, 5, 4, 1, 5\} collection. The counts constraint holds since this number is in fact equal (REL is set to =) to the last argument of the counts constraint. |       |       |           |
| Typical     |       |       |           |
| \|VALUES\| > 1 | \|VARIABLES\| > 1 | \text{range}(VARIABLES.var) > 1 | \|VARIABLES\| > \|VALUES\| | REL ∈ \{=, ≠, <, ≥, >, ≤\} | LIMIT > 0 | LIMIT < \|VARIABLES\| |
| Symmetries |       |       |           |
| • Items of VALUE are permutable. | • Items of VARIABLE are permutable. | • An occurrence of a value of VARIABLE.var that belongs to VALUE.val (resp. does not belong to VALUE.val) can be replaced by any other value in VALUE.val (resp. not in VALUE.val). |       |
Arg. properties

- **Contractible** wrt. VARIABLES when RELOP ∈ [<, ≤].
- **Extensible** wrt. VARIABLES when RELOP ∈ [≥, >].
- **Aggregate**: VALUES(union), VARIABLES(union), RELOP(id), LIMIT(+) when RELOP ∈ [<, ≤, ≥, >].

Usage

Used in the **Constraint(s) on sets** slot for defining some constraints like `assign_and_counts`.

Reformulation

The count(VALUE, VARIABLES, RELOP, LIMIT) constraint can be expressed in terms of the conjunction among (VALUE, VARIABLES, VALUES) ∧ VALUE RELOP LIMIT.

Systems

`count` in Gecode.

Used in

`assign_and_counts`.

See also

**assignment dimension added**: `assign_and_counts` (**assignment dimension introduced**).  
**common keyword**: among (**value constraint, counting constraint**).  
**specialisation**: count (variable ∈ VALUES replaced by variable=VALUE).

Keywords

**characteristic of a constraint**: automaton, automaton with counters.  
**constraint network structure**: alpha-acyclic constraint network(2).  
**constraint type**: value constraint, counting constraint.  
**filtering**: arc-consistency.  
**final graph structure**: acyclic, bipartite, no loop.
Arc input(s) | VARIABLES VALUES
---|---
Arc generator | \( \text{PRODUCT} \rightarrow \text{collection}(\text{variables}, \text{values}) \)
Arc arity | 2
Arc constraint(s) | \( \text{variables}.\text{var} = \text{values}.\text{val} \)
Graph property(ies) | NARC, RELOP, LIMIT
Graph class | • ACYCLIC
• BIPARTITE
• NO LOOP

Graph model

Because of the arc constraint \( \text{variables}.\text{var} = \text{values}.\text{val} \) and since each domain variable can take at most one value, NARC is the number of variables taking a value in the VALUES collection.

Parts (A) and (B) of Figure 5.163 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

![Diagram](image_url)

Figure 5.163: Initial and final graph of the counts constraint
Automaton

Figure 5.164 depicts the automaton associated with the counts constraint. To each variable \( \text{VAR}_i \) of the collection \( \text{VARIABLES} \) corresponds a 0-1 signature variable \( S_i \). The following signature constraint links \( \text{VAR}_i \) and \( S_i \): \( \text{VAR}_i \in \text{VALUES} \Leftrightarrow S_i \).

\[
\begin{align*}
\text{in} & (\text{VAR}_1, \text{VALUES}), \\
\{C=0\} \\
\{C=C+1\} \\
\text{not}_{\text{in}} & (\text{VAR}_1, \text{VALUES})
\end{align*}
\]

Figure 5.164: Automaton of the counts constraint

Figure 5.165: Hypergraph of the reformulation corresponding to the automaton of the counts constraint
### 5.85 coveredby_sboxes

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>LOGIC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometry, derived from [318]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>coveredby_sboxes(K, DIMS, OBJECTS, SBOXES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Synonym</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>coveredby.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Types</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARIABLES : collection(v−dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTEGERS : collection(v−int)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POSITIVES : collection(v−int)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K : int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIMS : sint</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OBJECTS : collection(oid−int,sid−int,x − VARIABLES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBOXES : collection(sid−int,t − INTEGERS,1 − POSITIVES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- | VARIABLES| ≥ 1 |
- | INTEGERS| ≥ 1 |
- | POSITIVES| ≥ 1 |
- required(VARIABLES,v) |
- | VARIABLES| = K |
- | INTEGERS| = K |
- | POSITIVES| = K |
- | POSITIVES.v| > 0 |
- | K| > 0 |
- | DIMS| ≥ 0 |
- | DIMS < K |
- increasing_seq(OBJECTS,[oid]) |
- required(OBJECTS,[oid,sid,x]) |
- | OBJECTS.oid| ≥ 1 |
- | OBJECTS.oid| ≤ |OBJECTS|
- | OBJECTS.sid| ≥ 1 |
- | OBJECTS.sid| ≤ |SBOXES|
- | SBOXES | ≥ 1 |
- | SBOXES.sid| ≥ 1 |
- | SBOXES.sid| ≤ |SBOXES|
- do_not_overlap(SBOXES) |
Holds if, for each pair of objects \((O_i, O_j)\), \(i < j\), \(O_i\) is covered by \(O_j\) with respect to a set of dimensions depicted by \(\text{DIMS}\). \(O_i\) and \(O_j\) are objects that take a shape among a set of shapes. Each \(\text{shape}\) is defined as a finite set of shifted boxes, where each shifted box is described by a box in a \(K\)-dimensional space at a given offset (from the origin of the shape) with given sizes. More precisely, a \(\text{shifted box}\) is an entity defined by its shape \(\text{id}\) \(\text{sid}\), shift offset \(\text{t}\), and sizes \(\text{l}\). Then, a shape is defined as the union of shifted boxes sharing the same shape \(\text{id}\). An \(\text{object}\) is an entity defined by its unique object identifier \(\text{oid}\), shape \(\text{id}\) \(\text{sid}\) and origin \(\text{x}\).

An object \(O_i\) is \(\text{covered by}\) an object \(O_j\) with respect to a set of dimensions depicted by \(\text{DIMS}\) if and only if, for all shifted box \(s_i\) of \(O_i\), there exists a shifted box \(s_j\) of \(O_j\) such that:

- For all dimensions \(d \in \text{DIMS}\), (1) the start of \(s_j\) in dimension \(d\) is less than or equal to the start of \(s_i\) in dimension \(d\), and (2) the end of \(s_j\) in dimension \(d\) is less than or equal to the end of \(s_i\) in dimension \(d\).
- There exists a dimension \(d\) where, (1) the start of \(s_j\) in dimension \(d\) coincide with the start of \(s_i\) in dimension \(d\), or (2) the end of \(s_j\) in dimension \(d\) coincide with the end of \(s_i\) in dimension \(d\).

\[
\begin{align*}
2, \{0, 1\}, \\
\text{oid} - 1 & \quad \text{sid} - 4 & \quad x = \{2, 3\}, \\
\text{oid} - 2 & \quad \text{sid} - 2 & \quad x = \{2, 2\}, \\
\text{oid} - 3 & \quad \text{sid} - 1 & \quad x = \{1, 1\} \\
\text{sid} - 1 & \quad \text{t} = \{0, 0\} & \quad l = \{3, 3\}, \\
\text{sid} - 1 & \quad \text{t} = \{3, 0\} & \quad l = \{2, 2\}, \\
\text{sid} - 2 & \quad \text{t} = \{0, 0\} & \quad l = \{2, 2\}, \\
\text{sid} - 2 & \quad \text{t} = \{2, 0\} & \quad l = \{1, 1\}, \\
\text{sid} - 3 & \quad \text{t} = \{0, 0\} & \quad l = \{2, 2\}, \\
\text{sid} - 3 & \quad \text{t} = \{2, 1\} & \quad l = \{1, 1\}, \\
\text{sid} - 4 & \quad \text{t} = \{0, 0\} & \quad l = \{1, 1\}
\end{align*}
\]

Figure 5.166 shows the objects of the example. Since \(O_1\) is covered by both \(O_2\) and \(O_3\), and since \(O_2\) is covered by \(O_3\), the \text{coveredby_sboxes} constraint holds.

Typical

\(|\text{OBJECTS}| > 1\)

Symmetries

- Items of \(\text{SBOXES}\) are \text{permutable}.
- Items of \(|\text{OBJECTS}.x, \text{SBOXES}.t\) and \(\text{SBOXES}.l\) are \text{permutable (same permutation used)}.

Remark

One of the eight relations of the \text{Region Connection Calculus} [318]. The constraint \text{coveredby_sboxes} is a restriction of the original relation since it requires that each shifted box of an object is covered by one shifted box of the other object.

See also

\begin{align*}
\text{common keyword:} & \quad \text{contains_sboxes, covers_sboxes,} \\
& \quad \text{disjoint_sboxes, equal_sboxes, inside_sboxes, meet_sboxes (\text{rcc8}),} \\
& \quad \text{non_overlap_sboxes (geometrical constraint, logic), overlap_sboxes (\text{rcc8}).}
\end{align*}
Keywords

- **constraint type:** logic.
- **geometry:** geometrical constraint, rcc8.
(A) Shape of the first object
(B) Shapes of the second object
(C) Shape of the third object
(D) Three objects O1, O2 and O3, where O1 is covered by both O2 and O3, and O2 is covered by O3

Figure 5.166: The three objects of the example
Logic

- \( \text{origin}(O_1, S_1, D) \overset{\text{def}}{=} O_1.x(D) + S_1.t(D) \)
- \( \text{end}(O_1, S_1, D) \overset{\text{def}}{=} O_1.x(D) + S_1.t(D) + S_1.l(D) \)
- \( \text{coveredby_sboxes}(\text{Dims}, O_1, S_1, O_2, S_2) \overset{\text{def}}{=} \)
  \[ \forall D \in \text{Dims} \]
  \[ \bigwedge \left( \begin{array}{c}
    \text{origin} \\
    O_2, S_2, D
  \end{array} \right) \leq \left( \begin{array}{c}
    \text{origin} \\
    O_1, S_1, D
  \end{array} \right),
\]
  \[ \bigwedge \left( \begin{array}{c}
    \text{end} \left( \begin{array}{c}
      O_1, S_1, D
    \end{array} \right) \leq \text{end} \left( \begin{array}{c}
      O_2, S_2, D
    \end{array} \right)
  \end{array} \right)
  \]
\[ \exists D \in \text{Dims} \]
\[ \left( \begin{array}{c}
    \text{origin} \\
    O_2, S_2, D
  \end{array} \right) = \left( \begin{array}{c}
    \text{origin} \\
    O_1, S_1, D
  \end{array} \right),
\]
\[ \bigvee \left( \begin{array}{c}
    \text{end} \left( \begin{array}{c}
      O_1, S_1, D
    \end{array} \right) = \text{end} \left( \begin{array}{c}
      O_2, S_2, D
    \end{array} \right)
  \end{array} \right)
  \]
- \( \text{coveredby_objects}(\text{Dims}, O_1, O_2) \overset{\text{def}}{=} \)
  \[ \forall S_1 \in \text{sboxes}(\text{[}O_1.\text{sid}\text{]}), \exists S_2 \in \text{sboxes}(\text{[}O_2.\text{sid}\text{]}),
\]
  \[ \text{coveredby_sboxes}(\text{Dims}, O_1, S_1, O_2, S_2) \]
- \( \text{all_coveredby}(\text{Dims}, OIDS) \overset{\text{def}}{=} \)
  \[ \forall O_1 \in \text{objects}(OIDS) \]
  \[ \forall O_2 \in \text{objects}(OIDS) \]
  \[ O_1.\text{oid} < \Rightarrow \]
  \[ O_2.\text{oid} \]
  \[ \text{coveredby_objects}(\text{Dims}, O_1, O_2) \]
- \( \text{all_coveredby}(\text{DIMENSIONS}, OIDS) \)
5.86 covers_sboxes

**DESCRIPTION**
Geometry, derived from [318]

**Constraint**
covers_sboxes(K, DIMS, OBJECTS, SBOXES)

**Synonym**
covers.

**Types**
- VARIABLES : collection(v−dvar)
- INTEGERS : collection(v−int)
- POSITIVES : collection(v−int)

**Arguments**
- K : int
- DIMS : sint
- OBJECTS : collection(oid−int,sid−int,x − VARIABLES)
- SBOXES : collection(sid−int,t − INTEGERS,l − POSITIVES)

**Restrictions**
- |VARIABLES| \(\geq 1\)
- |INTEGERS| \(\geq 1\)
- |POSITIVES| \(\geq 1\)
- required(VARIABLES.v)
- required(INTEGERS.v)
- required(POSITIVES.v)
- POSITIVES.v > 0
- K > 0
- DIMS \(\geq 0\)
- DIMS < K
- increasing_seq(OBJECTS,{oid})
- required(OBJECTS,{oid,sid,x})
- OBJECTS.oid \(\geq 1\)
- OBJECTS.oid \(\leq |\text{OBJECTS}|\)
- OBJECTS.sid \(\geq 1\)
- OBJECTS.sid \(\leq |\text{SBOXES}|\)
- SBOXES \(\geq 1\)
- required(SBOXES,{sid,t,l})
- SBOXES.sid \(\geq 1\)
- SBOXES.sid \(\leq |\text{SBOXES}|\)
- do_not_overlap(SBOXES)
Holds if, for each pair of objects \((O_i, O_j), i < j\), \(O_i\) covers \(O_j\) with respect to a set of dimensions depicted by \(\text{DIMS}\). \(O_i\) and \(O_j\) are objects that take a shape among a set of shapes. Each \textit{shape} is defined as a finite set of shifted boxes, where each shifted box is described by a box in a \(K\)-dimensional space at a given offset (from the origin of the shape) with given sizes. More precisely, a \textit{shifted box} is an entity defined by its shape \(\text{id}\), \(\text{sid}\), shift offset \(t\), and sizes \(l\). Then, a shape is defined as the union of shifted boxes sharing the same shape id. An \textit{object} is an entity defined by its unique object identifier \(\text{oid}\), \(\text{shape id}\) \(\text{sid}\) and origin \(x\).

**Purpose**

An object \(O_i\) covers an object \(O_j\) with respect to a set of dimensions depicted by \(\text{DIMS}\) if and only if, for all shifted box \(s_j\) of \(O_j\), there exists a shifted box \(s_i\) of \(O_i\) such that:

- For all dimensions \(d \in \text{DIMS}\), (1) the start of \(s_i\) in dimension \(d\) is less than or equal to the start of \(s_j\) in dimension \(d\), and (2) the end of \(s_j\) in dimension \(d\) is less than or equal to the end of \(s_i\) in dimension \(d\).

- There exists a dimension \(d\) where, (1) the start of \(s_i\) in dimension \(d\) coincide with the start of \(s_j\) in dimension \(d\), or (2) the end of \(s_i\) in dimension \(d\) coincide with the end of \(s_j\) in dimension \(d\).

\[
\begin{pmatrix}
2, \{0,1\}, \\
\text{oid} = 1 \quad \text{sid} = 1 \quad x = \{1,1\}, \\
\text{oid} = 2 \quad \text{sid} = 2 \quad x = \{2,2\}, \\
\text{oid} = 3 \quad \text{sid} = 4 \quad x = \{2,3\},
\end{pmatrix}
\]

**Example**

Figure 5.167 shows the objects of the example. Since \(O_1\) covers both \(O_2\) and \(O_3\), and since \(O_2\) covers \(O_3\), the \text{covers\_sboxes} constraint holds.

**Typical**

\(|\text{OBJECTS}| > 1\)

**Symmetries**

- Items of \text{SBOXES} are \text{permutable}.

- Items of \text{OBJECTS}\_x, \text{SBOXES}\_t and \text{SBOXES}\_l are \text{permutable} (same permutation used).

**Arg. properties**

\text{Suffix-contractible} wrt. \text{OBJECTS}.

**Remark**

One of the eight relations of the \text{Region Connection Calculus} [318]. The constraint \text{covers\_sboxes} is a relaxation of the original relation since it requires that each shifted box of an object is covered by one shifted box of the other object.

**See also**

\text{common keyword: contains\_sboxes, coveredby\_sboxes, disjoint\_sboxes, equal\_sboxes, inside\_sboxes, meet\_sboxes (rcc8), non\_overlap\_sboxes (geometrical constraint, logic), overlap\_sboxes (rcc8).}
<table>
<thead>
<tr>
<th>Keywords</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>constraint type:</strong> logic.</td>
</tr>
<tr>
<td></td>
<td><strong>geometry:</strong> geometrical constraint, rcc8.</td>
</tr>
</tbody>
</table>
(A) Shape of the first object

(B) Shapes of the second object

(C) Shape of the third object

(D) Three objects O1, O2 and O3, where O1 covers both O2 and O3, and O2 covers O3

Figure 5.167: The three objects of the example
• \( \text{origin}(O1, S1, D) \overset{\text{def}}{=} O1.x(D) + S1.t(D) \)
• \( \text{end}(O1, S1, D) \overset{\text{def}}{=} O1.x(D) + S1.t(D) + S1.l(D) \)
• \( \text{covers} \text{sboxes}(\text{Dims}, O1, S1, O2, S2) \overset{\text{def}}{=} \)
  \[
  \forall D \in \text{Dims} \left( \begin{array}{l}
  \text{origin}(O1, S1, D) \leq \\
  \text{origin}(O2, S2, D) \\
  \text{end}(O2, S2, D) \leq \\
  \text{end}(O1, S1, D)
  \end{array} \right), \\
  \exists D \in \text{Dims} \left( \begin{array}{l}
  \text{origin}(O1, S1, D) = \\
  \text{origin}(O2, S2, D) \\
  \text{end}(O1, S1, D) = \\
  \text{end}(O2, S2, D)
  \end{array} \right)
  \right) \)
• \( \text{covers} \text{objects}(\text{Dims}, O1, O2) \overset{\text{def}}{=} \forall S2 \in \text{sboxes}([O2.sid]) \exists S1 \in \text{sboxes}([O1.sid]) \text{covers} \text{sboxes}(\text{Dims}, O1, S1, O2, S2) \)
• \( \text{all} \text{covers}(\text{Dims, OIDS}) \overset{\text{def}}{=} \forall O1 \in \text{objects(OIDS)} \forall O2 \in \text{objects(OIDS)} O1.\text{oid} < \Rightarrow O2.\text{oid} \text{covers} \text{objects}(\text{Dims, O1, O2}) \)
• \( \text{all} \text{covers}(\text{DIMENSIONS, OIDS}) \)
## 5.87 crossing

### Description

<table>
<thead>
<tr>
<th>Link</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Inspired by [115].</td>
</tr>
</tbody>
</table>

### Constraint

\[ \text{crossing}(\text{NCROSS}, \text{SEGMENTS}) \]

### Arguments

- **NCROSS** : dvar
- **SEGMENTS** : collection \((\text{ox} - \text{dvar}, \text{oy} - \text{dvar}, \text{ex} - \text{dvar}, \text{ey} - \text{dvar})\)

### Restrictions

\[
\begin{align*}
\text{NCROSS} & \geq 0 \\
\text{NCROSS} & \leq (|\text{SEGMENTS}| \times |\text{SEGMENTS}| - |\text{SEGMENTS}|)/2 \\
\text{required} & (\text{SEGMENTS, [ox, oy, ex, ey]})
\end{align*}
\]

### Purpose

\(\text{NCROSS}\) is the number of line-segments intersections between the line-segments defined by the \(\text{SEGMENTS}\) collection. Each line-segment is defined by the coordinates \((\text{ox}, \text{oy})\) and \((\text{ex}, \text{ey})\) of its two extremities.

### Example

>\[
\begin{pmatrix}
\text{ox} - 1 & \text{oy} - 4 & \text{ex} - 9 & \text{ey} - 2, \\
\text{ox} - 1 & \text{oy} - 1 & \text{ex} - 3 & \text{ey} - 5, \\
\text{ox} - 3 & \text{oy} - 2 & \text{ex} - 7 & \text{ey} - 4, \\
\text{ox} - 9 & \text{oy} - 1 & \text{ex} - 9 & \text{ey} - 4
\end{pmatrix}
\]

Figure 5.168 provides a picture of the example with the corresponding four line-segments of the \(\text{SEGMENTS}\) collection. The crossing constraint holds since its first argument \(\text{NCROSS}\) is set to 3, which is actually the number of line-segments intersections.

### Typical

\(|\text{SEGMENTS}| > 1\)
Symmetries

- Items of SEGMENTS are **permutable**.
- Attributes of SEGMENTS are permutable w.r.t. permutation \((ox, oy) (ex, ey)\) (permutation applied to all items).
- One and the same constant can be **added** to the \(ox\) and \(ex\) attributes of all items of SEGMENTS.
- One and the same constant can be **added** to the \(oy\) and \(ey\) attributes of all items of SEGMENTS.

Arg. properties

Functional dependency: NCROSS determined by SEGMENTS.

See also

**common keyword**: graph_crossing, two_layer_edge_crossing (line-segments intersection).

Keywords

**constraint arguments**: pure functional dependency.

**final graph structure**: acyclic, no loop.

**geometry**: geometrical constraint, line-segments intersection.

**modelling**: functional dependency.
### Arc input(s)

**SEGMENTS**

### Arc generator

\[ \text{CLIQUE}(\prec) \rightarrow \text{collection}(s1, s2) \]

### Arc arity

2

### Arc constraint(s)

- \( \max(s1.ox, s1.ex) \geq \min(s2.ox, s2.ex) \)
- \( \max(s1.ox, s2.ex) \geq \min(s1.ox, s1.ex) \)
- \( \max(s2.ox, s1.ex) \geq \min(s2.ox, s2.ex) \)
- \( \max(s2.ox, s2.ex) \geq \min(s1.ox, s1.ex) \)

\[
\begin{align*}
\left( s2.ox - s1.ex \right) \star \left( s1.ey - s1.oy \right) - \prod \left( s1.ex - s1.ox, s2.ey - s1.ey \right) & = 0, \\
\left( s2.ex - s1.ex \right) \star \left( s2.oy - s1.oy \right) - \prod \left( s2.ex - s1.ex, s2.ey - s1.ey \right) & = 0, \\
\text{sign} \left( \left( s2.ox - s1.ex \right) \star \left( s1.ey - s1.oy \right) - \left( s1.ex - s1.ox \right) \star \left( s2.oy - s1.oy \right) - \left( s2.ex - s1.ex \right) \star \left( s2.oy - s1.oy \right) - \left( s2.ox - s1.ox \right) \star \left( s2.ey - s1.ey \right) \right) & \neq 0.
\end{align*}
\]

### Graph property(ies)

\[ \text{NARC} = \text{NCROSS} \]

### Graph class

- **ACYCLIC**
- **NO_LOOP**

### Graph model

Each line-segment is described by the \( x \) and \( y \) coordinates of its two extremities. In the arc generator we use the restriction \( \prec \) in order to generate one single arc for each pair of segments. This is required, since otherwise we would count more than once a given line-segments intersection.

Parts (A) and (B) of Figure 5.169 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. An arc constraint expresses the fact the two line-segments intersect. It is taken from [115, page 889]. Each arc of the final graph corresponds to a line-segments intersection.
Figure 5.169: Initial and final graph of the crossing constraint
### 5.88 cumulative

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>cumulative(TASKS, LIMIT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td>cumulative_max.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>TASKS : collection (origin−dvar, duration−dvar, end−dvar, height−dvar)</td>
<td>LIMIT : int</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>require_at_least(2, TASKS, [origin, duration, end])</td>
<td>required(TASKS, height)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TASKS.duration ≥ 0</td>
<td>TASKS.origin ≤ TASKS.end</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TASKS.height ≥ 0</td>
<td>LIMIT ≥ 0</td>
<td></td>
</tr>
</tbody>
</table>

**Purpose**
Cumulative scheduling constraint or scheduling under resource constraints. Consider a set $T$ of tasks described by the TASKS collection. The cumulative constraint enforces that at each point in time, the cumulated height of the set of tasks that overlap that point, does not exceed a given limit. A task overlaps a point $i$ if and only if (1) its origin is less than or equal to $i$, and (2) its end is strictly greater than $i$. It also imposes for each task of $T$ the constraint $\text{origin} + \text{duration} = \text{end}$.

**Example**

Figure 5.170 shows the cumulated profile associated with the example. To each task of the cumulative constraint corresponds a set of rectangles coloured with the same colour: the sum of the lengths of the rectangles corresponds to the duration of the task, while the height of the rectangles (i.e., all the rectangles associated with a task have the same height) corresponds to the resource consumption of the task. The cumulative constraint holds since at each point in time we do not have a cumulated resource consumption strictly greater than the upper limit 8 enforced by the last argument of the cumulative constraint.
Typical

\[
\begin{align*}
|\text{TASKS}| & > 1 \\
\text{range}(\text{TASKS}.\text{origin}) & > 1 \\
\text{range}(\text{TASKS}.\text{duration}) & > 1 \\
\text{range}(\text{TASKS}.\text{end}) & > 1 \\
\text{range}(\text{TASKS}.\text{height}) & > 1 \\
\text{TASKS}.\text{duration} & > 0 \\
\text{TASKS}.\text{height} & > 0 \\
\text{LIMIT} & < \sum(\text{TASKS}.\text{height})
\end{align*}
\]

Symmetries

- Items of \text{TASKS} are \textit{permutable}.
- \text{TASKS}.\text{duration} can be \textit{decreased} to any value \(\geq 0\).
- \text{TASKS}.\text{height} can be \textit{decreased} to any value \(\geq 0\).
- One and the same constant can be \textit{added} to the \text{origin} and \text{end} attributes of all items of \text{TASKS}.
- \text{LIMIT} can be \textit{increased}.

Arg. properties

\textit{Contractible} \textit{wrt. TASKS}.

Remark

In the original \textit{cumulative} constraint of \textit{CHIP} the \text{LIMIT} parameter was a domain variable corresponding to the \textit{maximum peak of the resource consumption profile}. Given a fixed time frame, this variable could be used as a cost in order to directly minimise the maximum resource consumption peak. Fixing this variable is potentially dangerous since it imposes the maximum peak to be equal to a given target value.

Some systems like Ilog CP Optimizer also assume that a zero-duration task overlaps a point \(i\) if and only if (1) its origin is less than or equal to \(i\), and (2) its end is greater than or equal to \(i\). Under this definition, the height of a zero-duration task is also taken into account in the resource consumption profile.

Note that the concept of cumulative is \textit{different} from the concept of rectangles non-overlapping even if, most of the time, each task of a ground solution of a cumulative constraint is simply drawn as a single rectangle. As illustrated by Figure 5.211, this is in fact not always possible (i.e., some rectangles may need to be broken apart). In fact the cumulative constraint is only a necessary condition for rectangles non-overlapping

![Figure 5.170: Resource consumption profile](image-url)
(see Figure 5.210 and the corresponding explanation in the Algorithm slot of the diffn constraint).

In MiniZinc (http://www.g12.cs.mu.oz.au/minizinc/) the tasks of cumulative constraint have no end attribute.

**Algorithm**

The first filtering algorithms were related to the notion of compulsory part of a task [232]. They compute a cumulated resource profile of all the compulsory parts of the tasks and prune the origins of the tasks with respect to this profile in order to not exceed the resource capacity. These methods are sometimes called time tabling. Even if these methods are quite local, i.e., a task has a non-empty compulsory part only when the difference between its latest start and its earliest start is strictly less than its duration, it scales well and is therefore widely used. Later on, more global algorithms based on the resource consumption of the tasks on specific intervals were introduced [141, 98, 246]. A popular variant, called edge finding, considers only specific intervals [264]. An efficient implementation of edge finding in $O(kn \log n)$, where $k$ is the number of distinct task heights and $n$ is the number of tasks, based on a specific data structure, so called a cumulative $\Phi$-tree [410], is provided in [409]. When the number of distinct task heights $k$ is not small, a usually almost faster implementation in $O(n^2)$ is described in [212]. A $O(n^2 \log n)$ filtering algorithm based on tasks that can not be the earliest (or not be the latest) is described in [354].

Within the context of linear programming, the reference [199] provides a relaxation of the cumulative constraint.

A necessary condition for the cumulative constraint is obtained by stating a disjunctive constraint on a subset of tasks $T$ such that, for each pair of tasks of $T$, the sum of the two corresponding minimum heights is strictly greater than LIMIT. This can be done by applying the following procedure:

- Let $h$ be the smallest minimum height strictly greater than $\lfloor \text{LIMIT} \rfloor$ of the tasks of the cumulative constraint. If no such task exists then the procedure is stopped without stating any disjunctive constraint.
- Let $T_h$ denote the set of tasks of the cumulative constraint for which the minimum height is greater than or equal to $h$. By construction, the tasks of $T_h$ cannot overlap. But we can eventually add one more task as shown by the next step.
- When it exists, we can add one task that does not belong to $T_h$ and such that its minimum height is strictly greater than $\text{LIMIT} - h$. Again, by construction, this task cannot overlap all the tasks of $T_h$.

When the tasks are involved in several cumulative constraints more sophisticated methods are available for extracting disjunctive constraints [16, 15].

In the context where, both the duration and height of all the tasks are fixed, [35] provides two kinds of additional filtering algorithms that are specially useful when the slack $\sigma$ (i.e., the difference between the available space and the sum of the surfaces of the tasks) is very small:

- The first one introduces bounds for the so called cumulative longest hole problem. Given an integer $\epsilon$ that does not exceed the resource limit, and a subset of tasks $T'$ for which the resource consumption is a most $\epsilon$, the cumulative longest hole

---

$^4$Even if these more global algorithms usually can prune more early in the search tree, these algorithms do not catch all deductions derived from the cumulated resource profile of compulsory parts.
The problem is to find the largest integer $l_{\text{max}} \epsilon \sigma(T')$ such that there is a cumulative placement of maximum height $\epsilon$ involving a subset of tasks of $T'$ where, on one interval $[i, i + l_{\text{max}} \epsilon (T') - 1]$ of the cumulative profile, the area of the empty space does not exceed $\sigma$.

- The second one used dynamic programming for filtering so called balancing knapsack constraints. When the slack is 0, such constraints express that the total height of tasks ending at instant $i$ must equal the total height of tasks starting at instant $i$. Such constraints can be generalized to non-zero slack.

**Systems**

- cumulativeMax in Choco, cumulative in Gecode, cumulative in JaCoP, cumulative in MiniZinc, cumulative in SICStus.

**See also**

- assignment dimension added: coloured_cumulatives (sum of task heights replaced by number of distinct colours, assignment dimension added), cumulatives (negative heights allowed and assignment dimension added).

**common keyword:**

- calendar (scheduling constraint).

**coloured_cumulative** (resource constraint, sum of task heights replaced by number of distinct values), coloured_cumulatives (resource constraint), cumulative_convex (resource constraint, task defined by a set of points), cumulative_product (resource constraint, sum of task heights replaced by product of task heights), cumulative_with_level_of_priority (resource constraint, a cumulative constraint for each set of tasks having a priority less than or equal to a given threshold).

**generalisation:** cumulative_two_d (task replaced by rectangle with a height).

**implied by:** diffn (cumulative is a necessary condition for each dimension of the diffn constraint).

**implies:** coloured_cumulative.

**related:** lex_chain_less, lex_chain_leq (lexicographic ordering on the origins of tasks, rectangles, ...), ordered_global_cardinality (controlling the shape of the cumulative profile for breaking symmetry).

**soft variant:** soft_cumulative.

**specialisation:** atmost (task replaced by variable), bin_packing (all tasks have a duration of 1 and a fixed height), disjunctive (all tasks have a height of 1), multi_inter_distance (all tasks have the same duration equal to DIST and the same height equal to 1).

**used in graph description:** sum_ctr.

**Keywords**

- characteristic of a constraint: core, automaton, automaton with array of counters.

- complexity: sequencing with release times and deadlines.

- constraint type: scheduling constraint, resource constraint, temporal constraint.

- filtering: linear programming, dynamic constraint, compulsory part, cumulative longest hole problems, Phi-tree.

- modelling: zero-duration task.

- problems: producer-consumer.

- puzzles: squared squares.
### Arc input(s)
| TASKS |

### Arc generator
- **SELF**: \( \rightarrow \) \( collection(\text{tasks}) \)

### Arc arity
1

### Arc constraint(s)
- \( \text{tasks}.\text{origin} + \text{tasks}.\text{duration} = \text{tasks}.\text{end} \)

### Graph property(ies)
- \( \text{NARC} = |\text{TASKS}| \)

### Arc input(s)
| TASKS, TASKS |

### Arc generator
- **PRODUCT**: \( \rightarrow \) \( collection(\text{tasks1}, \text{tasks2}) \)

### Arc arity
2

### Arc constraint(s)
- \( \text{tasks1}.\text{duration} > 0 \)
- \( \text{tasks2}.\text{origin} \leq \text{tasks1}.\text{origin} \)
- \( \text{tasks1}.\text{origin} < \text{tasks2}.\text{end} \)

### Graph class
- ACYCLIC
- BIPARTITE
- NO LOOP

### Sets
- \( \text{SUCC} \mapsto \begin{bmatrix} \text{source}, \\
\text{variables} \rightarrow \text{col}(\text{VARIABLES} \rightarrow \text{collection}(\text{var} \rightarrow \text{dvar}), \\
\text{item}(\text{var} \rightarrow \text{TASKS}.\text{height})) \end{bmatrix} \)

### Constraint(s) on sets
- \( \text{sum}_\text{ctr}(\text{variables}, \leq, \text{LIMIT}) \)

### Graph model
The first graph constraint enforces for each task the link between its origin, its duration and its end. The second graph constraint makes sure, for each time point \( t \) corresponding to the start of a task, that the cumulated heights of the tasks that overlap \( t \) does not exceed the limit of the resource.

Parts (A) and (B) of Figure 5.171 respectively show the initial and final graph associated with the second graph constraint of the Example slot. On the one hand, each source vertex of the final graph can be interpreted as a time point. On the other hand the successors of a source vertex correspond to those tasks that overlap that time point. The cumulative constraint holds since for each successor set \( S \) of the final graph the sum of the heights of the tasks in \( S \) does not exceed the limit \( \text{LIMIT} = 8 \).

### Signature
Since \( \text{TASKS} \) is the maximum number of vertices of the final graph of the first graph constraint we can rewrite \( \text{NARC} = |\text{TASKS}| \) to \( \text{NARC} \geq |\text{TASKS}| \). This leads to simplify \( \text{NARC} \) to \( \text{NARC} \).
Figure 5.171: Initial and final graph of the cumulative constraint
Figure 5.172: Automaton of the cumulative constraint
5.89 cumulative_convex

**DESCRIPTION**

Origin: Derived from cumulative

Constraint: cumulative_convex(TASKS, LIMIT)

Type: POINTS : collection(var–dvar)

Arguments:
- TASKS : collection(points – POINTS, height–dvar)
- LIMIT : int

Restrictions:
- required(POINTS.var)
- |POINTS| > 0
- required(TASKS,[points, height])
- TASKS.height ≥ 0
- LIMIT ≥ 0

Purpose: Cumulative scheduling constraint or scheduling under resource constraints. Consider a set \( T \) of tasks described by the TASKS collection where each task is defined by:

- A set of distinct points depicting the time interval where the task is actually running: the smallest and largest coordinates of these points respectively give the first and last instant of that time interval.

- A height that depicts the resource consumption used by the task from its first instant to its last instant.

The cumulative_convex constraint enforces that, at each point in time, the cumulated height of the set of tasks that overlap that point, does not exceed a given limit. A task overlaps a point \( i \) if and only if (1) its origin is less than or equal to \( i \), and (2) its end is strictly greater than \( i \).

**Example**

\[
\begin{align*}
\text{points} & \quad (2,1,5) & \quad \text{height} & \quad 1, \\
\text{points} & \quad (4,5,7) & \quad \text{height} & \quad 2, \\
\text{var} & \quad 14, \\
\text{var} & \quad 13, \\
\text{var} & \quad 9, \\
\text{var} & \quad 11, \\
\text{var} & \quad 10 \\
\end{align*}
\]

Figure 5.173 shows the cumulated profile associated with the example. To each set of points defining a task corresponds a rectangle. The height of each rectangle represents the resource consumption of the associated task. The cumulative_convex constraint holds since at each point in time we do not have a cumulated resource consumption strictly greater than the upper limit 3 enforced by the last argument of the cumulative_convex constraint.
**Typical**

| TASKS | > 1  
| TASKS.height | > 0  
| LIMIT | < \(\text{sum(TASKS.height)}\)

**Symmetries**

- Items of TASKS are **permutable**.
- Items of TASKS.points are **permutable**.
- TASKS.height can be **decreased** to any value \(\geq 0\).
- LIMIT can be **increased**.

**Arg. properties**

Contractible wrt. TASKS.

**Usage**

A natural use of the cumulative_convex constraint corresponds to problems where a task is defined as the convex hull of a set of distinct points \(P_1, \ldots, P_n\) that are not initially fixed. Note that, by explicitly introducing a start \(S\) and an end \(E\) variables, and by using a minimum\(S, (\text{var} - P_i, \ldots, \text{var} - P_n)\) and a maximum\(E, (\text{var} - P_i, \ldots, \text{var} - P_n)\) constraints, one could replace the cumulative_convex constraint by a cumulative constraint. However this hinders propagation.

As a concrete example of use of the cumulative_convex constraint we present a constraint model for a well-known pattern-sequencing problem \([153]\) (also known to be equivalent to the graph pathwidth \([244]\) problem) that is based on one single cumulative_convex constraint. The pattern sequencing problem can be described as follows: Given a 0-1 matrix in which each column \(j\) \((1 \leq j \leq p)\) corresponds to a product required by the customers and each row \(i\) \((1 \leq i \leq c)\) corresponds to the order of a particular customer (The entry \(c_{ij}\) is equal to 1 if and only if customer \(i\) has ordered some quantity of product \(j\)), the objective is to find a permutation of the products such that the maximum number of open orders at any point in the sequence is minimised. Order \(i\) is open at point \(k\) in the production sequence if there is a product required in order \(i\) that appears at or before position \(k\) in the sequence and also a product that appears at or after position \(k\) in the sequence.

Before giving the constraint model, let us first provide an instance of the pattern-sequencing problem. Consider the matrix \(M_1\) depicted by part (A1) of Fig. 5.174. Part (A2) gives its corresponding cumulated matrix \(M_2\) obtained by setting to 1 each 0 of \(M_1\) that is both preceded and followed by a 1. Part (A3) depicts the corresponding solution in term of the cumulative_convex constraint: to each row of the matrix \(M_1\) corresponds a task
Figure 5.174: An input matrix for the pattern sequencing problem (A1), its corresponding cumulated matrix (A2), a view in terms of tasks (A3) and the corresponding cumulative profile (A4). A second matrix (A2) where column 4 of (A1) is put at rightmost position.
defined as the convex hull of the different 1 located on that row. Finally part (A4) gives the cumulated profile associated with part (A3), namely the number of 1 in each column of \( M_2 \). The cost 3 of this solution is equal to the maximum number of 1 in the columns of the cumulated matrix \( M_2 \). As shown by parts (B1-B4), we can get a lower cost of 2 by pushing the fourth column to the rightmost position.

The idea of the model is to associate to each row (i.e., customer) \( i \) of the cumulated matrix a stack task that starts at the first 1 on row \( i \) and ends at the last 1 of row \( i \) (i.e., the task corresponds to the convex hull of the different 1 located on row \( i \)). Then the cost of a solution is simply the maximum height on the corresponding cumulated profile.

For each column \( j \) of the 0-1 matrix initially given there is a variable \( V_j \) ranging from 1 to the number of columns \( p \). The value of \( V_j \) gives the position of column \( j \) in a solution. We put all the stack tasks in a cumulative constraint, telling that each stack task uses one unit of the resource during all it execution. Since we want to have the same model for different limits on the maximum number of open stacks, and since all variables \( V_1, V_2, \ldots, V_p \) have to be distinct, we have an extra dummy task characterised as the convex hull of \( V_1, V_2, \ldots, V_p \). This extra dummy task has a height \( H \) that has to be maximised. For the matrix depicted by (A1) of Fig. 5.174 we pass to the cumulative constraint the following collection of tasks:

\[
\begin{align*}
\text{points} & = (P_1, P_2, P_3, P_4, P_6, P_7, P_9) & \text{height} & = 1, \\
\text{points} & = (P_2, P_5) & \text{height} & = 1, \\
\text{points} & = (P_4, P_7, P_8) & \text{height} & = 1, \\
\text{points} & = (P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8) & \text{height} & = 0
\end{align*}
\]

**Algorithm**

A first natural way to handle the cumulative constraint is to accumulate the convex hull of the different tasks in a profile and to prune according to this profile. We give the main ideas for computing the compulsory part of a task and for pruning a task according to the profile of compulsory parts.

**Compulsory part of a task** Given a task \( T \) characterised as the convex hull of a set of distinct points \( P_1, P_2, \ldots, P_k \) the compulsory part of \( T \) corresponds to the, possibly empty, interval \([s_T, e_T]\) where:

- \( s_T \) is the largest value \( v \) such that, when all variables \( P_1, P_2, \ldots, P_k \) are greater than or equal to \( v \), all variables \( P_1, P_2, \ldots, P_k \) can still take distinct values.
- \( e_T \) is the smallest value \( v \) such that, when all variables \( P_1, P_2, \ldots, P_k \) are less than or equal to \( v \), all variables \( P_1, P_2, \ldots, P_k \) can still take distinct values.

**Pruning according to the profile of compulsory parts** Given two instants \( i \) and \( j \) \((i < j)\) and a task \( T \) characterised as the convex hull of a set of distinct points \( P_1, P_2, \ldots, P_k \), assume that \( T \) cannot overlap \( i \) and \( j \) since this would lead exceeding LIMIT, the second argument of the cumulative constraint. Furthermore assume that, when all variables \( P_1, P_2, \ldots, P_k \) are both greater than \( i \) and less than \( j \), all variables \( P_1, P_2, \ldots, P_k \) cannot take distinct values. Then all values of \([i + 1, j - 1]\) can be removed from variables \( P_1, P_2, \ldots, P_k \).

**See also**

- **common keyword**: cumulative (resource constraint).
- **used in graph description**: alldifferent, between_min_max, sum_ctr.
Keywords

characteristic of a constraint: convex.
constraint type: scheduling constraint, resource constraint, temporal constraint.
filtering: compulsory part.
problems: pattern sequencing.
** Derived Collection **

\[ \text{col}(\text{INSTANTS}\rightarrow\text{collection(instant}\rightarrow\text{dvar}), [\text{item(instant}\rightarrow\text{TASKS.points.var}])} \]

- **Arc input(s):** TASKS
- **Arc generator:** SELF\rightarrow\text{collection(tasks)}
- **Arc arity:** 1
- **Arc constraint(s):** alldifferent(tasks.points)
- **Graph property(ies):** NARC = |TASKS|

- **Arc input(s):** INSTANTS TASKS
- **Arc generator:** PRODUCT\rightarrow\text{collection(instants,tasks)}
- **Arc arity:** 2
- **Arc constraint(s):** between_min_max(instants.instant, tasks.points)
- **Graph class:** ACYCLIC, BIPARTITE, NO_LOOP
- **Sets:**
  - SUCC \rightarrow \text{col} \left[ \text{variables} \rightarrow \text{col} \left[ \text{VARIABLES}\rightarrow\text{collection(var}\rightarrow\text{dvar}), [\text{item(var}\rightarrow\text{TASKS.height}]) \right] \right]
- **Constraint(s) on sets:** sum_ctr(variables, \leq, LIMIT)

**Graph model**

The first graph constraint enforces for each task that the set of points defining its time interval are all distinct. The second graph constraint makes sure for each time point \( t \), that the cumulated heights of the tasks that overlap \( t \) does not exceed the limit of the resource.

Parts (A) and (B) of Figure 5.175 respectively show the initial and final graph associated with the second graph constraint of the **Example** slot. On the one hand, each source vertex of the final graph can be interpreted as a time point corresponding to a point used in the definitions of the different tasks. On the other hand, the successors of a source vertex correspond to those tasks that overlap a given time point. The cumulative convex constraint holds since, for each successor set \( \mathcal{S} \) of the final graph, the sum of the heights of the tasks in \( \mathcal{S} \) does not exceed the limit \( \text{LIMIT} = 3 \).
Figure 5.175: Initial and final graph of the cumulative_convex constraint
5.90 cumulative_product

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from cumulative.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>cumulative_product(TASKS, LIMIT)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>TASKS : collection ( (\text{origin} - \text{dvar}, \text{duration} - \text{dvar}, \text{end} - \text{dvar}, \text{height} - \text{dvar}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LIMIT : int</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>require_at_least(2, TASKS, [origin, duration, end])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>required(TASKS, height)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TASKS.duration ( \geq 0 )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TASKS.origin ( \leq ) TASKS.end</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TASKS.height ( \geq 1 )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LIMIT ( \geq 0 )</td>
<td></td>
</tr>
</tbody>
</table>

Purpose

Consider a set \( \mathcal{T} \) of tasks described by the TASKS collection. The cumulative_product constraint enforces that at each point in time, the product of the heights of the set of tasks that overlap that point, does not exceed a given limit. A task overlaps a point \( i \) if and only if (1) its origin is less than or equal to \( i \), and (2) its end is strictly greater than \( i \). It also imposes for each task of \( \mathcal{T} \) the constraint \( \text{origin} + \text{duration} = \text{end} \).

Example

\[
\left( \begin{array}{cccc}
\text{origin} - 1 & \text{duration} - 3 & \text{end} - 4 & \text{height} - 1, \\
\text{origin} - 2 & \text{duration} - 9 & \text{end} - 11 & \text{height} - 2, \\
\text{origin} - 3 & \text{duration} - 10 & \text{end} - 13 & \text{height} - 1, \\
\text{origin} - 6 & \text{duration} - 6 & \text{end} - 12 & \text{height} - 1, \\
\text{origin} - 7 & \text{duration} - 2 & \text{end} - 9 & \text{height} - 3 \\
\end{array} \right)
\]

Figure 5.176 shows the solution associated with the example. To each task of the cumulative_product constraint corresponds a set of rectangles coloured with the same colour: the sum of the lengths of the rectangles corresponds to the duration of the task, while the height of the rectangles (i.e., all the rectangles associated with a task have the same height) corresponds to the height of the task. The profile corresponding to the product of the heights of the tasks that overlap a given point is depicted by a thick red line. The cumulative_product constraint holds since at each point in time the product of the heights of the tasks that overlap that point is not strictly greater than the upper limit 6 enforced by the last argument of the cumulative_product constraint.
Typical

\[
\begin{align*}
|\text{TASKS}| &> 1 \\
\text{range}(\text{TASKS}.\text{origin}) &> 1 \\
\text{range}(\text{TASKS}.\text{duration}) &> 1 \\
\text{range}(\text{TASKS}.\text{end}) &> 1 \\
\text{range}(\text{TASKS}.\text{height}) &> 1 \\
\text{TASKS}.\text{duration} &> 0 \\
\text{LIMIT} &< \text{prod}(\text{TASKS}.\text{height})
\end{align*}
\]

Symmetries

- Items of TASKS are permutable.
- TASKS\text{.height} can be decreased to any value \( \geq 0 \).
- One and the same constant can be added to the \text{origin} and \text{end} attributes of all items of TASKS.
- LIMIT can be increased.

Arg. properties

Contractible wrt. TASKS.

Reformulation

The cumulative\text{.product} constraint can be expressed in term of a set of reified constraints and of \(|\text{TASKS}|\) constraints of the form \( h_1 \cdot h_2 \cdot \ldots \cdot h_{|\text{TASKS}|} \leq l\):

1. For each pair of tasks \( \text{TASKS}[i], \text{TASKS}[j] \) (\( i, j \in [1, |\text{TASKS}|] \)) of the TASKS collection we create a variable \( H_{ij} \) which is set to the height of task \( \text{TASKS}[j] \) if task \( \text{TASKS}[j] \) overlaps the origin attribute of task \( \text{TASKS}[i] \), and to 1 otherwise:
   - If \( i = j \):
     - \( H_{ij} = \text{TASKS}[i].\text{height} \).
   - If \( i \neq j \):
     - \( H_{ij} = \text{TASKS}[j].\text{height} \lor H_{ij} = 1 \).
     - \((\text{TASKS}[j].\text{origin} \leq \text{TASKS}[i].\text{origin} \land \text{TASKS}[j].\text{end} > \text{TASKS}[i].\text{origin}) \lor (H_{ij} = \text{TASKS}[j].\text{height})) \lor \)
     - \((\text{TASKS}[i].\text{origin} > \text{TASKS}[j].\text{origin} \lor \text{TASKS}[j].\text{end} \leq \text{TASKS}[i].\text{origin}) \land (H_{ij} = 1))\)

2. For each task \( \text{TASKS}[i] \) (\( i \in [1, |\text{TASKS}|] \)) we impose a constraint of the form \( H_{i1} \cdot H_{i2} \cdot \ldots \cdot H_{i|\text{TASKS}|} \leq \text{LIMIT} \).

See also

common keyword: cumulative\text{.resource constraint).

used in graph description: product_ctr.

Figure 5.176: Solution of the cumulative\text{.product} constraint
Keywords

characteristic of a constraint: product.

constraint type: scheduling constraint, resource constraint, temporal constraint.

filtering: compulsory part.

modelling: zero-duration task.
### Arc input(s)

- TASKS

### Arc(s)

- **Arc generator**: SELF \(\rightarrow\) collection(tasks)
- **Arc arity**: 1
- **Arc constraint(s)**:
  - tasks\_origin + tasks\_duration = tasks\_end

### Graph property(ies)

- \(\text{NARC} = |\text{TASKS}|\)

### Arc input(s)

- TASKS TASKS

### Arc(s)

- **Arc generator**: PRODUCT \(\rightarrow\) collection(tasks1, tasks2)
- **Arc arity**: 2
- **Arc constraint(s)**:
  - tasks\_1\_duration > 0
  - tasks\_2\_origin \(\leq\) tasks\_1\_origin
  - tasks\_1\_origin < tasks\_2\_end

### Graph class

- ACYCLIC
- BIPARTITE
- NO LOOP

### Sets

- SUCC \(\rightarrow\)
  - source,
  - variables \(-\) col(VARIABLES\(-\)collection(var\(-\)dvar),
    - [item(var - ITEMS\_height)])

### Constraint(s) on sets

- product\_ctr(variables, \(\leq\), LIMIT)

### Graph model

Parts (A) and (B) of Figure 5.177 respectively show the initial and final graph associated with the second graph constraint of the **Example** slot. On the one hand, each source vertex of the final graph can be interpreted as a time point. On the other hand the successors of a source vertex correspond to those tasks that overlap that time point. The cumulative product constraint holds since for each successor set \(S\) of the final graph the product of the heights of the tasks in \(S\) does not exceed the limit \(\text{LIMIT} = 6\).

### Signature

Since TASKS is the maximum number of vertices of the final graph of the first graph constraint we can rewrite \(\text{NARC} = |\text{TASKS}|\) to \(\text{NARC} \geq |\text{TASKS}|\). This leads to simplify \(\text{NARC} \to \text{NARC}\).
Figure 5.177: Initial and final graph of the cumulative_product constraint
### 5.91 cumulative_two_d

#### DESCRIPTION

Inspired by `cumulative` and `diffn`.

#### LINKS

Constraint: `cumulative_two_d(RECTANGLES, LIMIT)`

#### Arguments

- **RECTANGLES**: collection
  - `start1 - dvar`
  - `size1 - dvar`
  - `last1 - dvar`
  - `start2 - dvar`
  - `size2 - dvar`
  - `last2 - dvar`
  - `height - dvar`

- **LIMIT**: int

#### Restrictions

- `require_at_least(2, RECTANGLES, [start1, size1, last1])`
- `require_at_least(2, RECTANGLES, [start2, size2, last2])`
- `required(RECTANGLES, height)`
- `RECTANGLES.size1 ≥ 0`
- `RECTANGLES.size2 ≥ 0`
- `RECTANGLES.height ≥ 0`
- `LIMIT ≥ 0`

#### Purpose

Consider a set $R$ of rectangles described by the `RECTANGLES` collection. Enforces that at each point of the plane, the cumulated height of the set of rectangles that overlap that point, does not exceed a given limit.

#### Example

Part (A) of Figure 5.178 shows the 4 parallelepipeds of height 4, 2, 3 and 1 associated with the items of the `RECTANGLES` collection (parallelepipeds since each rectangle also has a height). Part (B) gives the corresponding cumulated 2-dimensional profile, where each number is the cumulated height of all the rectangles that contain the corresponding region. The `cumulative_two_d` constraint holds since the highest peak of the cumulated 2-dimensional profile does not exceed the upper limit 4 imposed by the last argument of the `cumulative_two_d` constraint.

#### Typical

- `|RECTANGLES| > 1`
- `RECTANGLES.size1 > 0`
- `RECTANGLES.size2 > 0`
- `RECTANGLES.height > 0`
- `LIMIT < sum(RECTANGLES.height)`
Symmetries
- Items of RECTANGLES are permutable.
- Attributes of RECTANGLES are permutable w.r.t. permutation \((\text{start1}, \text{start2}) (\text{size1}, \text{size2}) (\text{last1}, \text{last2}) (\text{height})\) (permutation applied to all items).
- RECTANGLES.height can be decreased to any value \(\geq 0\).
- One and the same constant can be added to the \text{start1} and \text{last1} attributes of all items of RECTANGLES.
- One and the same constant can be added to the \text{start2} and \text{last2} attributes of all items of RECTANGLES.
- LIMIT can be increased.

Arg. properties
Contractible wrt. RECTANGLES.

Usage
The cumulative\_two\_d constraint is a necessary condition for the \text{diffn} constraint in 3 dimensions (i.e., the placement of parallelepipeds in such a way that they do not pairwise overlap and that each parallelepiped has its sides parallel to the sides of the placement space).

Algorithm
A first natural way to handle this constraint would be to accumulate the compulsory part \([232]\) of the different rectangles in a quadtree \([346]\). To each leave of the quadtree we associate the cumulated height of the rectangles containing the corresponding region.

Systems
gc, geost in Choco.

See also
related: \text{diffn}(cumulative\_two\_d is a necessary condition for \text{diffn}: forget one dimension when the number of dimensions is equal to 3).
specialisation: bin\_packing(square of size 1 with a height replaced by task of duration 1), cumulative(rectangle with a height replaced by task with same height).

Keywords
characteristic of a constraint: derived collection.
constraint type: predefined constraint.
filtering: quadtree, compulsory part.

Figure 5.178: Two representations of a 2-dimensional cumulated profile
**geometry**: geometrical constraint.
5.92 cumulative_with_level_of_priority

**DESCRIPTION**

Constraint

```
cumulative_with_level_of_priority(TASKS,PRIORITIES)
```

**ARGS**

```
TASKS : collection
    (priority-int, origin-dvar, duration-dvar, end-dvar, height-dvar)

PRIORITIES : collection(id-int, capacity-int)
```

**RESTRICTIONS**

```
required(TASKS,[priority,height])
require_at_least(2,TASKS,[origin,duration,end])
TASKS.priority ≥ 1
TASKS.priority ≤ |PRIORITIES|
TASKS.duration ≥ 0
TASKS.origin ≤ TASKS.end
TASKS.height ≥ 0
required(PRIORITIES,[id,capacity])
PRIORITY.id ≥ 1
PRIORITY.id ≤ |PRIORITIES|
increasing_seq(PRIORITIES,id)
increasing_seq(PRIORITIES,capacity)
```

**PURPOSE**

Consider a set $T$ of tasks described by the TASKS collection where each task has a given priority chosen in the range $[1,|PRIORITIES|]$. Let $T_i$ denote the subset of tasks of $T$ that all have a priority less than or equal to $i$. For each set $T_i$, the cumulative_with_level_of_priority constraint enforces that at each point in time, the cumulated height of the set of tasks that overlap that point, does not exceed a given limit. A task overlaps a point $i$ if and only if (1) its origin is less than or equal to $i$, and (2) its end is strictly greater than $i$. Finally, it also imposes for each task of $T$ the constraint origin + duration = end.

**EXAMPLE**

```
( priority - 1 origin - 1 duration - 2 end - 3 height - 1, 
  priority - 1 origin - 2 duration - 3 end - 5 height - 1, 
  priority - 1 origin - 5 duration - 2 end - 7 height - 2, 
  priority - 2 origin - 3 duration - 2 end - 5 height - 2, 
  priority - 2 origin - 6 duration - 3 end - 9 height - 1
  (id - 1 capacity - 2, id - 2 capacity - 3)
```

Figure 5.179 shows the cumulated profile associated with both levels of priority. To each task of the cumulative_with_level_of_priority constraint corresponds a set of rectangles containing the same number (i.e., the position of the task within the TASKS collection): the sum of the lengths of the rectangles corresponds to the duration of the
task, while the height of the rectangles (i.e., all the rectangles associated with a task have the same height) corresponds to the resource consumption of the task. Tasks that have a priority of 1 are coloured in pink, while tasks that have a priority of 2 are coloured in blue. The cumulative_with_level_of_priority constraint holds since:

- At each point in time the cumulated resource consumption profile of the tasks of priority 1 does not exceed the upper capacity 2 enforced by the first item of the PRIORITIES collection.
- At each point in time the cumulated resource consumption profile of the tasks of priority 1 and 2 does not exceed the upper capacity 3 enforced by the second item of the PRIORITIES collection.

Figure 5.179: Resource consumption profile according to both levels of priority

Typical

<table>
<thead>
<tr>
<th></th>
<th>TASKS</th>
<th></th>
<th>PRIORITIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>[TASKS] &gt; 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>range(TASKS.priority) &gt; 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>range(TASKS.origin) &gt; 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>range(TASKS.duration) &gt; 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>range(TASKS.end) &gt; 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TASKS.duration &gt; 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TASKS.height &gt; 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PRIORITIES</td>
<td>[PRIORITIES] &gt; 0</td>
<td></td>
</tr>
<tr>
<td>PRIORITIES.capacity &lt; sum(TASKS.height)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[TASKS] &gt; [PRIORITIES]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Symmetries

- Items of TASKS are permutable.
- TASKS.priority can be increased to any value ≤ |PRIORITIES|.
- TASKS.height can be decreased to any value ≥ 0.
- One and the same constant can be added to the origin and end attributes of all items of TASKS.
- PRIORITIES.capacity can be increased.

Arg. properties

Contractible wrt. TASKS.

Usage

The cumulative_with_level_of_priority constraint was suggested by problems from the telecommunication area where one has to ensure different levels of quality of service.
For this purpose the capacity of a transmission link is splitted so that a given percentage is reserved to each level. In addition we have that, if the capacities allocated to levels 1, 2, ..., i is not completely used, then level i+1 can use the corresponding spare capacity.

**Remark**

The cumulative_with_level_of_priority constraint can be modelled by a conjunction of cumulative constraints. As shown by the next example, the consistency for all variables of the cumulative constraints does not implies consistency for the corresponding cumulative_with_level_of_priority constraint. The following cumulative_with_level_of_priority constraint

\[
\begin{pmatrix}
\text{priority} - 1 \text{ origin} - o_1 \text{ duration} - 2 \text{ height} - 2, \\
\text{priority} - 1 \text{ origin} - o_2 \text{ duration} - 2 \text{ height} - 1, \\
\text{priority} - 2 \text{ origin} - o_3 \text{ duration} - 1 \text{ height} - 3, \\
\text{id} - 1 \text{ capacity} - 2, \\
\text{id} - 2 \text{ capacity} - 3
\end{pmatrix}
\]

where the domains of \(o_1\), \(o_2\) and \(o_3\) are respectively equal to \{1, 2, 3\}, \{1, 2, 3\} and \{1, 2, 3, 4\} corresponds to the following conjunction of cumulative constraints

\[
\text{cumulative}
\begin{pmatrix}
\text{origin} - o_1 \text{ duration} - 2 \text{ height} - 2, \\
\text{origin} - o_2 \text{ duration} - 2 \text{ height} - 1, \\
\text{id} - 1 \text{ capacity} - 2, \\
\text{id} - 2 \text{ capacity} - 3
\end{pmatrix}, 2
\]

\[
\text{cumulative}
\begin{pmatrix}
\text{origin} - o_1 \text{ duration} - 2 \text{ height} - 2, \\
\text{origin} - o_2 \text{ duration} - 2 \text{ height} - 1, \\
\text{origin} - o_3 \text{ duration} - 1 \text{ height} - 3
\end{pmatrix}, 3
\]

Even if the cumulative constraint could achieve arc-consistency, the previous conjunction of cumulative constraints would not detect the fact that there is no solution.

**See also**

*common keyword: cumulative (resource constraint).*

*used in graph description: sum.ctr.*

**Keywords**

*characteristic of a constraint: derived collection.*

*constraint type: scheduling constraint, resource constraint, temporal constraint.*

*modelling: zero-duration task.*
Derived Collection

\[
\text{TIME_POINTS; collection(idp - int, duration - dvar, point - dvar),}
\]

\[
\text{item (idp - TASKS.priority, duration - TASKS.duration, point - TASKS.origin),}
\]

\[
\text{item (idp - TASKS.priority, duration - TASKS.duration, point - TASKS.end).}
\]

Arc input(s) TASKS
Arc generator \( \text{SELF} \rightarrow \text{collection(tasks)} \)
Arc arity 1
Arc constraint(s) \( \text{tasks.origin + tasks.duration = tasks.end} \)
Graph property(ies) \( \text{NARC} = |\text{TASKS}| \)

For all items of PRIORITIES:

Arc input(s) TIME_POINTS TASKS
Arc generator \( \text{PRODUCT} \rightarrow \text{collection(time_points, tasks)} \)
Arc arity 2
Arc constraint(s) \( \text{time_points.idp = PRIORITIES.id} \)
\( \text{time_points.idp} \geq \text{tasks.priority} \)
\( \text{time_points.duration} > 0 \)
\( \text{tasks.origin} \leq \text{time_points.point} \)
\( \text{time_points.point} < \text{tasks.end} \)
Graph class \( \text{ACYCLIC} \)
\( \text{BIPARTITE} \)
\( \text{NO_LOOP} \)
Sets \( \text{SUCC} \rightarrow \)
\[
\begin{bmatrix}
\text{source}, \\
\text{variables - col (VARIABLES; collection(var - dvar), [item(var - TASKS.height)])}
\end{bmatrix}
\]
Constraint(s) on sets \( \text{sum_ctr} (\text{variables, } \leq \text{PRIORITIES.capacity}) \)

Graph model

Within the context of the second graph constraint, part (A) of Figure 5.180 shows the initial graphs associated with priorities 1 and 2 of the Example slot. Part (B) of Figure 5.180 shows the corresponding final graphs associated with priorities 1 and 2. On the one hand, each source vertex of the final graph can be interpreted as a time point \( p \). On the other hand the successors of a source vertex correspond to those tasks that both overlap that time point \( p \) and have a priority less than or equal to a given level. The cumulative_with_level_of_priority constraint holds since for each successor set \( S \) of the final graph the sum of the height of the tasks in \( S \) is less than or equal to the capacity associated with a given level of priority.
Figure 5.180: Initial and final graph of the cumulative_with_level_of_priority constraint
Since \( \text{TASKS} \) is the maximum number of vertices of the final graph of the first graph constraint we can rewrite \( \text{NARC} = |\text{TASKS}| \) to \( \text{NARC} \geq |\text{TASKS}| \). This leads to simplify \( \text{NARC} \) to \( \text{NARC} \).
5.93 cumulatives

**DESCRIPTION**

- **Constraint**: cumulatives(TASKS, MACHINES, CTR)
- **Arguments**
  - TASKS: collection
  - MACHINES: collection
  - CTR: atom
- **Restrictions**
  - required(TASKS, [machine, height])
  - require_at_least(2, TASKS, [origin, duration, end])
  - in_attr(TASKS, machine, MACHINES, id)
  - TASKS.duration ≥ 0
  - TASKS.origin ≤ TASKS.end
  - |MACHINES| > 0
  - required(MACHINES, [id, capacity])
  - distinct(MACHINES, id)
  - CTR ∈ [≤, ≥]

**Purpose**

Consider a set \( T \) of tasks described by the TASKS collection. When \( CTR \) is equal to \( \leq \) (respectively \( ≥ \)), the cumulatives constraint enforces the following condition for each machine \( m \): At each point in time, where at least one task assigned on machine \( m \) is present, the cumulated height of the set of tasks that both overlap that point and are assigned to machine \( m \) should be less than or equal to (respectively greater than or equal to) the capacity associated with machine \( m \). A task overlaps a point \( i \) if and only if (1) its origin is less than or equal to \( i \), and (2) its end is strictly greater than \( i \). It also imposes for each task of \( T \) the constraint \( \text{origin} + \text{duration} = \text{end} \).

**Example**

Figure 5.181 shows with a thick line the cumulated profile on the two machines described by the MACHINES collection. Within this profile a task with a positive (respectively negative) height is represented by a pink (respectively blue) rectangle, where the length of the rectangle corresponds to the duration of the task. The cumulatives constraint...
holds since, both on machines 1 and 2, we have that at each point in time the cumulated resource consumption is greater than or equal to the limit 0 enforced by the last argument (i.e., the attribute capacity of the items of the MACHINES collection) of the cumulatives constraint (i.e., we have a limit of 0 both on machines 1 and 2).

![Resource consumption profile on the different machines](image)

**Figure 5.181: Resource consumption profile on the different machines**

**Typical**

<table>
<thead>
<tr>
<th>TASKS</th>
<th>&gt; 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>range(TASKS.machine)</td>
<td>&gt; 1</td>
</tr>
<tr>
<td>range(TASKS.origin)</td>
<td>&gt; 1</td>
</tr>
<tr>
<td>range(TASKS.duration)</td>
<td>&gt; 1</td>
</tr>
<tr>
<td>range(TASKS.end)</td>
<td>&gt; 1</td>
</tr>
<tr>
<td>range(TASKS.height)</td>
<td>&gt; 1</td>
</tr>
<tr>
<td>TASKS.duration</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>TASKS.height</td>
<td>≠ 0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>MACHINES</td>
<td>&gt; 1</td>
</tr>
<tr>
<td>MACHINES.capacity</td>
<td>&lt; sum(TASKS.height)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>TASKS</td>
<td>&gt;</td>
</tr>
</tbody>
</table>

**Symmetries**

- Items of TASKS are permutable.
- Items of MACHINES are permutable.
- All occurrences of two distinct values in TASKS.machine or MACHINES.id can be swapped; all occurrences of a value in TASKS.machine or MACHINES.id can be renamed to any unused value.

**Arg. properties**

Contractible wrt. TASKS when RELOP ∈ [≤] and minval(TASKS.height) ≥ 0.

**Usage**

As shown in the Example slot, the cumulatives constraint is useful for covering problems where different demand profiles have to be covered by a set of tasks. This is modelled in the following way:

- To each demand profile is associated a given machine m and a set of tasks for which all attributes (machine, origin, duration, end, height) are fixed; moreover the machine attribute is fixed to m and the height attribute is strictly negative. For each machine m the cumulated profile of all the previous tasks constitutes the demand profile to cover.
- To each task that can be used to cover the demand is associated a task for which the height attribute is a positive integer; the height attribute describes the amount of
demand that can be covered by the task at each instant during its execution (between its origin and its end) on the demand profile associated with the machine attribute.

- In order to express the fact that each demand profile should completely be covered, we set the capacity attribute of each machine to 0. We can also relax the constraint by setting the capacity attribute to a negative number that specifies the maximum allowed uncovered demand at each instant.

The demand profiles might also not be completely fixed in advance.

When all the heights of the tasks are non-negative, one other possible use of the cumulatives constraint is to enforce to reach a minimum level of resource consumption. This is imposed on those time points that are overlapped by at least one task.

By introducing a dummy task of height 0, of origin the minimum origin of all the tasks and of end the maximum end of all the tasks, this can also be imposed between the first and the last utilisation of the resource.

Finally the cumulatives constraint is also useful for scheduling problems where several cumulative machines are available and where you have to assign each task on a specific machine.

Algorithm

Three filtering algorithms for this constraint are described in [32].

Systems

cumulatives in Gecode, cumulatives in SICStus.

See also

assignment dimension removed: cumulative (negative heights not allowed).

common keyword:

calendar (scheduling constraint),

coloured_cumulatives (resource constraint).

generalisation: diffn (task with machine assignment and origin attributes replaced by orthotope).

used in graph description: sum_ctr.

Keywords

application area: workload covering.

characteristic of a constraint: derived collection.

complexity: sequencing with release times and deadlines.

constraint type: scheduling constraint, resource constraint, temporal constraint, timetabling constraint.

filtering: compulsory part, sweep.

modelling: assignment dimension, assignment to the same set of values, scheduling with machine choice, calendars and preemption, zero-duration task.

modelling exercises: assignment to the same set of values, scheduling with machine choice, calendars and preemption.

problems: producer-consumer, demand profile.
Derived Collection

\[
\text{TIME_POINTS} \rightarrow \text{collection(idm=int, duration=dvar, point=dvar)},
\]

\[
\text{item} \rightarrow \text{collection(idm=\text{TASKS}\.machine, duration=\text{TASKS}\.duration, point=\text{TASKS}\.origin)}
\]

\[
\text{item} \rightarrow \text{collection(idm=\text{TASKS}\.machine, duration=\text{TASKS}\.duration, point=\text{TASKS}\.end)}
\]

Arc input(s) TASKS
Arc generator SELF \rightarrow \text{collection(tasks)}
Arc arity 1
Arc constraint(s) tasks\.origin + tasks\.duration = tasks\.end
Graph property(ies) \text{NARC} = |\text{TASKS}|

For all items of MACHINES:

Arc input(s) \text{TIME_POINTS TASKS}
Arc generator \text{PRODUCT} \rightarrow \text{collection(time_points, tasks)}
Arc arity 2
Arc constraint(s)

- time_points\.idm = MACHINES.id
- time_points\.idm = tasks\.machine
- time_points\.duration > 0
- tasks\.origin \leq time_points\.point
- time_points\.point < tasks\.end

Graph class

- \text{ACYCLIC}
- \text{BIPARTITE}
- \text{NO_LOOP}

Sets

\text{SUCC} \rightarrow

\[
\text{source, variables} \rightarrow \text{col(VARIABLES collection(var=dvar)), [item(var=TASKS\.height)]}
\]

Constraint(s) on sets \text{sum_ctr(variables,CTR,MACHINES\.capacity)}

Graph model

Within the context of the second graph constraint, part (A) of Figure 5.182 shows the initial graphs associated with machines 1 and 2 of the Example slot. Part (B) of Figure 5.182 shows the corresponding final graphs associated with machines 1 and 2. On the one hand, each source vertex of the final graph can be interpreted as a time point \(\hat{p}\) on a specific machine \(m\). On the other hand the successors of a source vertex correspond to those tasks that both overlap that time point \(\hat{p}\) and are assigned to machine \(m\). Since they do not have any successors we have eliminated those vertices corresponding to the end of the last three tasks of the TASKS collection. The cumulatives constraint holds since for each successor set \(\mathcal{S}\) of the final graph the sum of the height of the tasks in \(\mathcal{S}\) is greater than or equal to the capacity of the machine corresponding to the time point associated with \(\mathcal{S}\).
Figure 5.182: Initial and final graph of the cumulatives constraint
Signature

Since \textit{NARC} is the maximum number of vertices of the final graph of the first graph constraint we can rewrite \(\text{NARC} = |\text{TASKS}|\) to \(\text{NARC} \geq |\text{TASKS}|\). This leads to simplify \textit{NARC} to \textit{NARC}. 
### 5.94 cutset

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[143]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>cutset(SIZE_CUTSET, NODES)</td>
<td></td>
</tr>
</tbody>
</table>
| Arguments | SIZE\_CUTSET : dvar  
NODES : collection(index - int, succ - sint, bool - dvar) | |
| Restrictions | SIZE\_CUTSET ≥ 0  
SIZE\_CUTSET ≤ |NODES|  
required(NODES, [index, succ, bool])  
NODES.index ≥ 1  
NODES.index ≤ |NODES|  
distinct(NODES, index)  
NODES.bool ≥ 0  
NODES.bool ≤ 1 | |
| Purpose | Consider a digraph $G$ with $n$ vertices described by the NODES collection. Enforces that the subset of kept vertices of cardinality $n - \text{SIZE\_CUTSET}$ and their corresponding arcs form a graph without circuit. | |
| Example | $\begin{pmatrix} 1,  
index - 1 & \text{succ} - \{2, 3, 4\} & \text{bool} - 1,  
\text{index} - 2 & \text{succ} - \{3\} & \text{bool} - 1,  
\text{index} - 3 & \text{succ} - \{4\} & \text{bool} - 1,  
\text{index} - 4 & \text{succ} - \{1\} & \text{bool} - 0 \end{pmatrix}$ | The cutset constraint holds since the vertices of the NODES collection for which the bool attribute is set to 1 correspond to a graph without circuit and since exactly one (SIZE\_CUTSET = 1) vertex has its bool attribute set to 0. |
| Typical | SIZE\_CUTSET > 0  
SIZE\_CUTSET ≤ |NODES|  
|NODES| > 1 | |
| Symmetry | Items of NODES are permutable. | |
| Usage | The article [143] introducing the cutset constraint mentions applications from various areas such that deadlock breaking or program verification. | |
| Remark | The undirected version of the cutset constraint corresponds to the minimum feedback vertex set problem. | |
| Algorithm | The filtering algorithm presented in [143] uses graph reduction techniques inspired from Levy and Low [242] as well as from Lloyd, Soffa and Wang [245]. | |
Keywords

- **Application area**: deadlock breaking, program verification.
- **Constraint type**: graph constraint.
- **Final graph structure**: circuit, directed acyclic graph, acyclic, no loop.
- **Problems**: minimum feedback vertex set.
Arc input(s) NODES
Arc generator \( CLIQUE \rightarrow \text{collection}(\text{nodes}_1, \text{nodes}_2) \)
Arc arity 2
Arc constraint(s)
- \( \text{in}_\text{set}(\text{nodes}_2.\text{index}, \text{nodes}_1.\text{succ}) \)
- \( \text{nodes}_1.\text{bool} = 1 \)
- \( \text{nodes}_2.\text{bool} = 1 \)
Graph property(ies)
- \( \text{MAX}_\text{NSCC} \leq 1 \)
- \( \text{NVERTEX} = |\text{NODES}| - \text{SIZE}_\text{CUTSET} \)
Graph class
- \( \text{ACYCLIC} \)
- \( \text{NO}_\text{LOOP} \)

Graph model
We use a set of integers for representing the successors of each vertex. Because of the arc constraint, all arcs such that the bool attribute of one extremity is equal to 0 are eliminated; Therefore all vertices for which the bool attribute is equal to 0 are also eliminated (since they will correspond to isolated vertices). The graph property \( \text{MAX}_\text{NSCC} \leq 1 \) enforces the size of the largest strongly connected component to not exceed 1; Therefore, the final graph can’t contain any circuit.

Part (A) of Figure 5.183 shows the initial graph from which we have chosen to start. It is derived from the set associated with each vertex. Each set describes the potential values of the succ attribute of a given vertex. Part (B) of Figure 5.183 gives the final graph associated with the Example slot. Since we use the \( \text{NVERTEX} \) graph property, the vertices of the final graph are stressed in bold. The cutset constraint holds since the final graph does not contain any circuit and since the number of removed vertices \( \text{SIZE}_\text{CUTSET} \) is equal to 1.

![Figure 5.183: Initial and final graph of the cutset set constraint](image-url)
5.95 cycle

### Description

**Origin**

[39]

**Constraint**

\[ \text{cycle} (\text{NCYCLE}, \text{NODES}) \]

**Arguments**

- **NCYCLE**: dvar
- **NODES**: collection(index=int, succ=dvar)

**Restrictions**

- **NCYCLE** \( \geq 1 \)
- **NCYCLE** \( \leq |\text{NODES}| \)
- **required** (\text{NODES}, [index, succ])
- \text{NODES}.index \( \geq 1 \)
- **required** (\text{NODES}, index)
- **distinct** (\text{NODES}, index)
- **NODES**.succ \( \geq 1 \)
- **NODES**.succ \( \leq |\text{NODES}| \)

### Purpose

Consider a digraph \( G \) described by the \text{NODES} collection. \text{NCYCLE} is equal to the number of circuits for covering \( G \) in such a way that each vertex of \( G \) belongs to one single circuit. \text{NCYCLE} can also be interpreted as the number of cycles of the permutation associated with the successor variables of the \text{NODES} collection.

### Example

\[
\begin{pmatrix}
\text{index} - 1 & \text{succ} - 2, \\
\text{index} - 2 & \text{succ} - 1, \\
\text{index} - 3 & \text{succ} - 5, \\
\text{index} - 4 & \text{succ} - 3, \\
\text{index} - 5 & \text{succ} - 4
\end{pmatrix}
\]

In this example we have the following 2 (\text{NCYCLE} = 2) cycles: \( 1 \rightarrow 2 \rightarrow 1 \) and \( 3 \rightarrow 5 \rightarrow 4 \rightarrow 3 \). Consequently, the cycle constraint holds.

### Typical

- \text{NCYCLE} < |\text{NODES}|
- |\text{NODES}| > 2

### Symmetry

Items of \text{NODES} are permutable.

### Arg. properties

**Functional dependency**: \text{NCYCLE} determined by \text{NODES}.

### Usage

The PhD thesis of Éric Bourreau [79] mentions the following applications of extensions of the cycle constraint:

- The balanced Euler knight problem where one tries to cover a rectangular chessboard of size \( N \times M \) by \( C \) knights that all have to visit between \( 2 \cdot \left\lfloor \left( N \cdot M \right) / C \right\rfloor / 2 \) and \( 2 \cdot \left\lceil \left( N \cdot M \right) / C \right\rceil / 2 \) distinct locations. For some values of \( N, M \) and \( C \) there
does not exist any solution to the previous problem. This is for instance the case when \( N = M = C = 6 \). Figure 5.184 depicts the graph associated with the \( 6 \times 6 \) chessboard as well as examples of balanced solutions with respectively 1, 2, 3, 4 and 5 knights.

- Some **pick-up delivery** problems where a fleet of vehicles has to transport a set of orders. Each order is characterised by its initial location, its final destination and its weight. In addition one also has to take into account the capacity of the different vehicles.

![Graph of potential moves of a 6 X 6 chessboard](image1)

![1 knight (36 moves)](image2)

![2 knights (18 and 18 moves)](image3)

![3 knights (12, 12 and 12 moves)](image4)

![4 knights (8, 8, 10 and 10 moves)](image5)

![5 knights (6, 6, 8 and 8 moves)](image6)

Figure 5.184: Graph of potential moves of the \( 6 \times 6 \) chessboard and corresponding balanced tours

**Remark**

In the original cycle constraint of CHIP the index attribute was not explicitly present. It was implicitly defined as the position of a variable in a list.

In an early version of the CHIP there was a constraint named circuit that, from a declarative point of view, was equivalent to cycle(1, NODES). In ALICE [238] the circuit constraint was also present.

Given a complete digraph of \( n \) vertices as well as an unrestricted number of circuits NCYCLE, the total number of solutions of the corresponding cycle constraint corresponds to the sequence A000142 of the On-Line Encyclopedia of Integer Sequences [370]. Given a complete digraph of \( n \) vertices as well as a fixed number of circuits NCYCLE between 1 and \( n \), the total number of solutions of the corresponding cycle constraint corresponds to the so called Stirling number of first kind.

**Algorithm**

Since all succ variables have to take distinct values one can reuse the algorithms associated with the alldifferent constraint. A second necessary condition is to have no more than
Reformulation

Let \( n \) and \( s_1, s_2, \ldots, s_n \) respectively denote the number of vertices (i.e., \(|\text{NODES}|\)) and the successor variables associated with vertices 1, 2, \ldots, \( n \). The cycle constraint can be reformulated as a conjunction of one \texttt{alldifferent} constraint, \( n \cdot (n - 1) \) \texttt{element} constraints, \( n \) \texttt{minimum} constraints, and one \texttt{nvalue} constraint.

- First, we state an \texttt{alldifferent}(\( s_1, s_2, \ldots, s_n \)) constraint for enforcing distinct values to be assigned to the successor variables.
- Second, the key idea is to extract for each vertex \( i \) (with \( i \in [1, n] \)) all the vertices that belong to the same cycle. This is done by stating a conjunction of \( n - 1 \) \texttt{element} constraints of the form:
  \[
  \text{element}(i, (s_1, s_2, \ldots, s_n), s_{i,1}).
  \]
  \[
  \text{element}(s_{i,1}, (s_1, s_2, \ldots, s_n), s_{i,2}).
  \]
  \[
  \ldots
  \]
  \[
  \text{element}(s_{i,n-2}, (s_1, s_2, \ldots, s_n), s_{i,n-1}).
  \]

Then, using a \texttt{minimum}(\( m_1, (i, s_{i,1}, s_{i,2}, \ldots, s_{i,n-1}) \)) constraint, we get a unique representative for the cycle containing vertex \( i \).
- Third, using a \texttt{nvalue}(\texttt{NCYCLE}, \( \langle m_1, m_2, \ldots, m_n \rangle \)) constraint, we get the number of distinct cycles.

See also

- \texttt{alldifferent} (permutation),
- \texttt{cycle_cluster} (graph constraint, one_suc),
- \texttt{cycle_card_on_path} (permutation, graph partitioning constraint),
- \texttt{cycle_or_accessibility} (graph constraint),
- \texttt{cycle_resource} (graph partitioning constraint),
- \texttt{derangement} (permutation),
- \texttt{graph_crossing} (graph constraint, graph partitioning constraint),
- \texttt{inverse} (permutation),
- \texttt{map} (graph partitioning constraint),
- \texttt{symmetric_alldifferent} (permutation),
- \texttt{tour} (graph constraint),
- \texttt{tree} (graph partitioning constraint).

\texttt{implies: alldifferent}.

\texttt{related: balance_cycle} (counting number of cycles versus controlling how balanced the cycles are).

\texttt{specialisation: circuit} (\texttt{NCYCLE} set to 1).

\texttt{used in reformulation: alldifferent, element, minimum, nvalue}.

Keywords

- characteristic of a constraint: core.
- combinatorial object: permutation.
- constraint arguments: business rules.
constraint type: graph constraint, graph partitioning constraint.
filtering: strong bridge, DFS-bottleneck.
final graph structure: circuit, connected component, strongly connected component, one_succ.
modelling: cycle, functional dependency.
problems: pick-up delivery.
puzzles: Euler knight.
Arc input(s)  NODES
Arc generator  $\text{CLIQUE} \rightarrow \text{collection}(\text{nodes}_1, \text{nodes}_2)$
Arc arity  2
Arc constraint(s)  $\text{nodes}_1.\text{succ} = \text{nodes}_2.\text{index}$
Graph property(ies)  
  • $\text{NTREE} = 0$
  • $\text{NCC} = \text{NCYCLE}$
Graph class  ONE\_SUCC

Graph model

From the restrictions and from the arc constraint, we deduce that we have a bijection from the successor variables to the values of interval $[1, |\text{NODES}|]$. With no explicit restrictions it would have been impossible to derive this property.

In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices. This is why the cycle constraint considers objects that have two attributes:

- One fixed attribute index that is the identifier of the vertex,
- One variable attribute succ that is the successor of the vertex.

The graph property $\text{NTREE} = 0$ is used in order to avoid having vertices that both do not belong to a circuit and have at least one successor located on a circuit. This concretely means that all vertices of the final graph should belong to a circuit.

Parts (A) and (B) of Figure 5.185 respectively show the initial and final graph associated with the Example slot. Since we use the NCC graph property, we show the two connected components of the final graph. The constraint holds since all the vertices belong to a circuit (i.e., $\text{NTREE} = 0$) and since $\text{NCYCLE} = \text{NCC} = 2$.

![Figure 5.185: Initial and final graph of the cycle constraint](image-url)
### 5.96 cycle_card_on_path

<table>
<thead>
<tr>
<th>Description</th>
<th>Links</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHIP</td>
<td>cycle_card_on_path(NCYCLE, NODES, ATLEAST, ATMOST, PATH_LEN, VALUES)</td>
<td></td>
</tr>
</tbody>
</table>

#### Arguments
- `NCYCLE` : dvar
- `NODES` : collection(index=int, succ=dvar, colour=dvar)
- `ATLEAST` : int
- `ATMOST` : int
- `PATH_LEN` : int
- `VALUES` : collection(val=int)

#### Restrictions
- `NCYCLE ≥ 1`
- `NCYCLE ≤ |NODES|`
- `required(NODES, [index, succ, colour])`
- `NODES.index ≥ 1`
- `NODES.index ≤ |NODES|`
- `distinct(NODES, index)`
- `NODES.succ ≥ 1`
- `NODES.succ ≤ |NODES|`
- `ATLEAST ≥ 0`
- `ATLEAST ≤ PATH_LEN`
- `ATMOST ≥ ATLEAST`
- `PATH_LEN ≥ 0`
- `|VALUES| ≥ 1`
- `required(VALUES, val)`
- `distinct(VALUES, val)`

#### Purpose
Consider a digraph $G$ described by the `NODES` collection. `NCYCLE` is the number of circuits for covering $G$ in such a way that each vertex belongs to one single circuit. In addition the following constraint must also hold: on each set of `PATH_LEN` consecutive distinct vertices of each final circuit, the number of vertices for which the attribute colour takes his value in the collection of values `VALUES` should be located within the range `[ATLEAST, ATMOST]`.

#### Example
\[
\begin{pmatrix}
  \text{index}\, -1 & \text{succ}\, -7 & \text{colour}\, -2, \\
  \text{index}\, -2 & \text{succ}\, -4 & \text{colour}\, -3, \\
  \text{index}\, -3 & \text{succ}\, -8 & \text{colour}\, -2, \\
  \text{index}\, -4 & \text{succ}\, -9 & \text{colour}\, -1, \\
  \text{index}\, -5 & \text{succ}\, -1 & \text{colour}\, -2, & 1, 2, 3, \\
  \text{index}\, -6 & \text{succ}\, -2 & \text{colour}\, -1, \\
  \text{index}\, -7 & \text{succ}\, -5 & \text{colour}\, -1, \\
  \text{index}\, -8 & \text{succ}\, -6 & \text{colour}\, -1, \\
  \text{index}\, -9 & \text{succ}\, -3 & \text{colour}\, -1 \\
\end{pmatrix}
\]
The constraint `cycle_card_on_path` holds since the vertices of the `NODES` collection correspond to a set of disjoint circuits and since, for each set of 3 (i.e., `PATH_LEN = 3`) consecutive vertices, colour 1 (i.e., the value provided by the `VALUES` collection) occurs at least once (i.e., `ATLEAST = 1`) and at most twice (i.e., `ATMOST = 2`).

### Typical

<table>
<thead>
<tr>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="typical.png" alt="Typical" /></td>
</tr>
</tbody>
</table>

### Symmetries

- Items of `NODES` are **permutable**.
- An occurrence of a value of `NODES.colour` that belongs to `VALUES.val` (resp. does not belong to `VALUES.val`) can be **replaced** by any other value in `VALUES.val` (resp. not in `VALUES.val`).
- `ATLEAST` can be **decreased** to any value \( \geq 0 \).
- `ATMOST` can be **increased**.
- Items of `VALUES` are **permutable**.

### Usage

Assume that the vertices of \( G \) are partitioned into the following two categories:

- Clients to visit.
- Depots where one can reload a vehicle.

Using the `cycle_card_on_path` constraint we can express a constraint like: after visiting three consecutive clients we should visit a depot. This is typically not possible with the `atmost` constraint since we do not know in advance the set of variables involved in the `atmost` constraint.

### Remark

This constraint is a special case of the `sequence` parameter of the `cycle` constraint of CHIP \([79] pages 121–128\).

### See also

- **common keyword**: `cycle (graph partitioning constraint)`.
- **used in graph description**: `among_low_up`.

### Keywords

- **characteristic of a constraint**: coloured.
- **combinatorial object**: sequence.
- **constraint type**: graph constraint, graph partitioning constraint, sliding sequence constraint.
- **final graph structure**: connected component, one_suc.
Arc input(s) NOD\(E\)S
Arc generator \(\text{CLIQUE} \rightarrow \text{collection}(\text{nodes1}, \text{nodes2})\)
Arc arity 2
Arc constraint(s) \(\text{nodes1}\).\text{succ} = \text{nodes2}.\text{index}\)
Graph property(ies) • \(\text{NTREE} = 0\)
• \(\text{NCC} = \text{NCYCLE}\)
Graph class \(\text{ONE\_SUC}\)
Sets \(\text{PATH\_LENGTH}(\text{PATH\_LEN}) \rightarrow \left[\text{variables} - \text{col} \left(\text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \text{item} (\text{var} - \text{NOD}\(E\)).\text{colour})\right)\right]\)
Constraint(s) on sets \(\text{among\_low\_up}(\text{ATLEAST}, \text{ATMOST}, \text{variables}, \text{VALUES})\)

Graph model

Parts (A) and (B) of Figure 5.186 respectively show the initial and final graph associated with the Example slot. Since we use the \(\text{NCC}\) graph property, we show the two connected components of the final graph. The constraint \(\text{cycle\_card\_on\_path}\) holds since all the vertices belong to a circuit (i.e., \(\text{NTREE} = 0\)) and since for each set of three consecutive vertices, colour 1 occurs at least once and at most twice (i.e., the \(\text{among\_low\_up}\) constraint holds).

\(\text{(A)}\)

\(\text{(B)}\)

Figure 5.186: Initial and final graph of the \(\text{cycle\_card\_on\_path}\) constraint
5.97 **cycle_or_accessibility**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inspired by [226].</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>cycle_or_accessibility(MAXDIST, NCYCLE, NODES)</td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>MAXDIST : int</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NCYCLE : dvar</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES : collection(index-int, succ-dvar, x-int, y-int)</td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>MAXDIST ≥ 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NCYCLE ≥ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NCYCLE ≤</td>
<td>NODES</td>
</tr>
<tr>
<td></td>
<td>required(NODES, [index, succ, x, y])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.index ≥ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.index ≤</td>
<td>NODES</td>
</tr>
<tr>
<td></td>
<td>distinct(NODES, index)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.succ ≥ 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.succ ≤</td>
<td>NODES</td>
</tr>
<tr>
<td></td>
<td>NODES.x ≥ 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.y ≥ 0</td>
<td></td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Consider a digraph G described by the NODES collection. Cover a subset of the vertices of G by a set of vertex-disjoint circuits in such a way that the following property holds: for each uncovered vertex v₁ of G there exists at least one covered vertex v₂ of G such that the Manhattan distance between v₁ and v₂ is less than or equal to MAXDIST.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Figure 5.187 represents the solution associated with the example. The covered vertices are coloured in blue, while the links starting from the uncovered vertices are dashed. The cycle_or_accessibility constraint holds since:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• In the solution we have NCYCLE = 2 disjoint circuits.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• All the 3 uncovered nodes are located at a distance that does not exceed MAXDIST = 3 from at least one covered node.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Typical</strong></td>
<td>MAXDIST &gt; 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NCYCLE &lt;</td>
<td>NODES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NODES</td>
</tr>
</tbody>
</table>

\[
\begin{bmatrix}
\text{index} - 1 & \text{succ} - 6 & x - 4 & y - 5, \\
\text{index} - 2 & \text{succ} - 0 & x - 9 & y - 1, \\
\text{index} - 3 & \text{succ} - 0 & x - 2 & y - 4, \\
\text{index} - 4 & \text{succ} - 1 & x - 2 & y - 6, \\
\text{index} - 5 & \text{succ} - 5 & x - 7 & y - 2, \\
\text{index} - 6 & \text{succ} - 4 & x - 4 & y - 7, \\
\text{index} - 7 & \text{succ} - 0 & x - 6 & y - 4
\end{bmatrix}
\]
Symmetries

- Items of NODES are permutable.
- Attributes of NODES are permutable w.r.t. permutation (index) (succ) (x, y) (permutation applied to all items).
- One and the same constant can be added to the x attribute of all items of NODES.
- One and the same constant can be added to the y attribute of all items of NODES.

Arg. properties

Functional dependency: NCYCLE determined by NODES.

Remark

This kind of facilities location problem is described in [226, pages 187–189] pages. In addition to our example they also mention the cost problem that is usually a trade-off between the vertices that are directly covered by circuits and the others.

See also

common keyword: cycle (graph constraint).
used in graph description: nvalues,except,0.

Keywords

constraint type: graph constraint.
final graph structure: strongly connected component.
geometry: geometrical constraint.
modelling: functional dependency.
problems: facilities location problem.

Figure 5.187: Final graph associated with the facilities location problem
Arc input(s) NODES
Arc generator $\text{CLIQUE} \mapsto \text{collection}(\text{nodes1}, \text{nodes2})$
Arc arity 2
Arc constraint(s) $\text{nodes1.succ} = \text{nodes2.index}$
Graph property(ies) • $\text{NTREE} = 0$
• $\text{NCC} = \text{NCYCLE}$

Arc input(s) NODES
Arc generator $\text{CLIQUE} \mapsto \text{collection}(\text{nodes1}, \text{nodes2})$
Arc arity 2
Arc constraint(s) \begin{align*}
\bigvee \left( & \bigwedge \left( \begin{array}{l}
\text{nodes1.succ} = \text{nodes2.index}, \\
\text{nodes1.succ} = 0, \\
\text{nodes2.succ} \neq 0, \\
\text{abs}(\text{nodes1.x} - \text{nodes2.x}) + \text{abs}(\text{nodes1.y} - \text{nodes2.y}) \leq \text{MAXDIST}
\end{array} \right) \right)
\end{align*}
Graph property(ies) $\text{NVERTEX} = |\text{NODES}|$
Sets $\text{PRED} \mapsto \begin{bmatrix}
\text{variables} - \text{col} \left( \text{VARIABLES} - \text{collection}([\text{var} - \text{dvar}], [\text{item}([\text{var} - \text{NODES}]]></bmatrix), \\
\text{destination}
\end{bmatrix}$
Constraint(s) on sets $\text{nvalues except 0}([\text{variables}, =, 1])$

Graph model

For each vertex $v$ we have introduced the following attributes:

- $\text{index}$: the label associated with $v$,
- $\text{succ}$: if $v$ is not covered by a circuit then 0; If $v$ is covered by a circuit then index of the successor of $v$.
- $\text{x}$: the $x$-coordinate of $v$,
- $\text{y}$: the $y$-coordinate of $v$.

The first graph constraint enforces all vertices, which have a non-zero successor, to form a set of $\text{NCYCLE}$ vertex-disjoint circuits.

The final graph associated with the second graph constraint contains two types of arcs:

- The arcs belonging to one circuit (i.e., $\text{nodes1.succ} = \text{nodes2.index}$),
- The arcs between one vertex $v_1$ that does not belong to any circuit (i.e., $\text{nodes1.succ} = 0$) and one vertex $v_2$ located on a circuit (i.e., $\text{nodes2.succ} \neq 0$) such that the Manhattan distance between $v_1$ and $v_2$ is less than or equal to $\text{MAXDIST}$.

In order to specify the fact that each vertex is involved in at least one arc we use the graph property $\text{NVERTEX} = |\text{NODES}|$. Finally the dynamic constraint $\text{nvalues except 0}([\text{variables}, =, 1])$ expresses the fact that, for each vertex $v$, there is exactly one predecessor of $v$ that belongs to a circuit.
Parts (A) and (B) of Figure 5.188 respectively show the initial and final graph associated with the second graph constraint of the Example slot.

![Diagram](A) ![Diagram](B)

Figure 5.188: Initial and final graph of the cycle_or_accessibility constraint

**Signature**

Since |NODES| is the maximum number of vertices of the final graph associated with the second graph constraint we can rewrite \( N\text{VERTEX} = |\text{NODES}| \) to \( N\text{VERTEX} \geq |\text{NODES}| \). This leads to simplify \( N\text{VERTEX} \) to \( N\text{VERTEX} \).
5.98 cycle_resource

Description

Consider a digraph $G$ defined as follows:

- To each item of the RESOURCE and TASK collections corresponds one vertex of $G$. A vertex that was generated from an item of the RESOURCE (respectively TASK) collection is called a resource vertex (respectively task vertex).
- There is an arc from a resource vertex $r$ to a task vertex $t$ if $t \in \text{RESOURCE}[r].\text{first_task}$.
- There is an arc from a task vertex $t$ to a resource vertex $r$ if $r \in \text{TASK}[t].\text{next_task}$.
- There is an arc from a task vertex $t_1$ to a task vertex $t_2$ if $t_2 \in \text{TASK}[t_1].\text{next_task}$.
- There is no arc between two resource vertices.

Enforce to cover $G$ in such a way that each vertex belongs to one single circuit. Each circuit is made up from one single resource vertex and zero, one or more task vertices. For each resource-vertex a domain variable indicates how many task-vertices belong to the corresponding circuit. For each task a domain variable provides the identifier of the resource that can effectively handle that task.
The cycle_resource constraint holds since the graph corresponding to the vertices described by its arguments consists of the following 3 disjoint circuits:

- The first circuit involves the resource vertex 1 as well as the task vertices 5, 4 and 7.
- The second circuit is limited to the resource vertex 2.
- Finally the third circuit is made up from the remaining vertices, namely the resource vertex 3 and the task vertices 8 and 6.

**Typical**

<table>
<thead>
<tr>
<th>RESOURCE</th>
<th>&gt; 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>TASK</td>
<td>&gt; 1</td>
</tr>
<tr>
<td>TASK</td>
<td>&gt;</td>
</tr>
</tbody>
</table>

**Symmetries**

- Items of RESOURCE are permutable.
- Items of TASK are permutable.
- All occurrences of two distinct values in RESOURCE.id or TASK.resource can be swapped.

**Usage**

This constraint is useful for some vehicles routing problem where the number of locations to visit depends of the vehicle type that is effectively used. The resource attribute allows expressing various constraints such as:

- The compatibility or incompatibility between tasks and vehicles,
- The fact that certain tasks should be performed by the same vehicle,
- The preassignment of certain tasks to a given vehicle.

**Remark**

This constraint could be expressed with the cycle constraint of CHIP by using the following optional parameters:

- The resource node parameter [79, page 97].
- The circuit weight parameter [79, page 101].
- The name parameter [79, page 104].

**See also**

common keyword: cycle (graph partitioning constraint).

**Keywords**

characteristic of a constraint: derived collection.

constraint type: graph constraint, resource constraint, graph partitioning constraint.

final graph structure: connected component, strongly connected component.
Derived Collection

\[
\text{col} := \text{RESOURCE\_TASK}\rightarrow\text{collection}(\text{index}\rightarrow\text{int}, \text{succ}\rightarrow\text{dvar}, \text{name}\rightarrow\text{dvar}),
\]

\[
\begin{align*}
\text{item} & : \text{succ}\rightarrow\text{RESOURCE\_first\_task} \\
\text{name} & : \text{index}\rightarrow\text{RESOURCE\_id}
\end{align*}
\]

\[
\begin{align*}
\text{item} & : \text{succ}\rightarrow\text{TASK\_id} \\
\text{name} & : \text{index}\rightarrow\text{TASK\_resource}
\end{align*}
\]

Arc input(s)  RESOURCE\_TASK
Arc generator  \( CLIQUE \mapsto \text{collection}(\text{resource\_task1}, \text{resource\_task2}) \)
Arc arity  2
Arc constraint(s)  
\[
\begin{align*}
\text{resource\_task1\_succ} &= \text{resource\_task2\_index} \\
\text{resource\_task1\_name} &= \text{resource\_task2\_name}
\end{align*}
\]
Graph property(ies)  
\[
\begin{align*}
\text{NTREE} &= 0 \\
\text{NCC} &= |\text{RESOURCE}| \\
\text{NVERTEX} &= |\text{RESOURCE}| + |\text{TASK}|
\end{align*}
\]
Graph class  ONE\_SUCC

For all items of RESOURCE:

Arc input(s)  RESOURCE\_TASK
Arc generator  \( CLIQUE \mapsto \text{collection}(\text{resource\_task1}, \text{resource\_task2}) \)
Arc arity  2
Arc constraint(s)  
\[
\begin{align*}
\text{resource\_task1\_succ} &= \text{resource\_task2\_index} \\
\text{resource\_task1\_name} &= \text{resource\_task2\_name} \\
\text{resource\_task1\_name} &= \text{RESOURCE\_id}
\end{align*}
\]
Graph property(ies)  \text{NVERTEX} = \text{RESOURCE\_nb\_task} + 1

Graph model

The graph model of the cycle\_resource constraint illustrates the following points:

- How to differentiate the constraint on the length of a circuit according to a resource that is assigned to a circuit? This is achieved by introducing a collection of resources and by asking a different graph property for each item of that collection.
- How to introduce the concept of name that corresponds to the resource that handles a given task? This is done by adding to the arc constraint associated with the cycle constraint the condition that the name variables of two consecutive vertices should be equal.

Part (A) of Figure 5.189 shows the initial graphs (of the second graph constraint) associated with resources 1, 2 and 3 of the Example slot. Part (B) of Figure 5.189 shows the corresponding final graphs (of the second graph constraint) associated with resources 1, 2 and 3. Since we use the NVERTEX graph property, the vertices of the final graphs are stressed in bold. To each resource corresponds a circuit of respectively 3, 0 and 2 task-vertices.
Signature

Since the initial graph of the first graph constraint contains $|\text{RESOURCE}| + |\text{TASK}|$ vertices, the corresponding final graph cannot have more than $|\text{RESOURCE}| + |\text{TASK}|$ vertices. Therefore we can rewrite the graph property $N_{\text{VERTEX}} = |\text{RESOURCE}| + |\text{TASK}|$ to $N_{\text{VERTEX}} \geq |\text{RESOURCE}| + |\text{TASK}|$ and simplify $N_{\text{VERTEX}}$ to $N_{\text{VERTEX}}$. 
Figure 5.189: Initial and final graph of the cycle_resource constraint
5.99 cyclic_change

**Description**
Derived from change.

**Constraint**
cyclic_change(NCHANGE, CYCLE_LENGTH, VARIABLES, CTR)

**Arguments**
- **NCHANGE**: dvar
- **CYCLE_LENGTH**: int
- **VARIABLES**: collection(var−dvar)
- **CTR**: atom

**Restrictions**
- \( NCHANGE \geq 0 \)
- \( NCHANGE < |VARIABLES| \)
- \( CYCLE_LENGTH > 0 \)
- \( required(VARIABLES, var) \)
- \( VARIABLES.var \geq 0 \)
- \( VARIABLES.var < CYCLE_LENGTH \)
- \( CTR \in [\neq, <, \geq, >, \leq] \)

**Purpose**
\( NCHANGE \) is the number of times that constraint \( ((X + 1) \mod CYCLE_LENGTH) \neq Y \) holds; \( X \) and \( Y \) correspond to consecutive variables of the collection \( VARIABLES \).

**Example**
\( (2, 4, (3, 0, 2, 3, 1), \neq) \)

Since \( CTR \) is set to \( \neq \) and since \( CYCLE_LENGTH \) is set to 4, a change between two consecutive items \( X \) and \( Y \) of the \( VARIABLES \) collection corresponds to the fact that the condition \( ((X + 1) \mod 4) \neq Y \) holds. Consequently, the cyclic_change constraint holds since we have the two following changes (i.e., \( NCHANGE = 2 \)) within \( (3, 0, 2, 3, 1) \):

- A first change between the consecutive values 0 and 2,
- A second change between the consecutive values 3 and 1.

However, the sequence 3 0 does not correspond to a change since \( (3 + 1) \mod 4 \) is equal to 0.

**Typical**
- \( NCHANGE > 0 \)
- \( |VARIABLES| > 1 \)
- \( range(VARIABLES.var) > 1 \)
- \( CTR \in [\neq] \)

**Symmetry**
Items of \( VARIABLES \) can be shifted.

**Arg. properties**
Functional dependency: \( NCHANGE \) determined by \( CYCLE_LENGTH \), \( VARIABLES \) and \( CTR \).
Usage
This constraint may be used for personnel cyclic timetabling problems where each person has to work according to cycles. In this context each variable of the VARIABLES collection corresponds to the type of work a person performs on a specific day. Because of some perturbation (e.g., illness, unavailability, variation of the workload) it is in practice not reasonable to ask for perfect cyclic solutions. One alternative is to use the cyclic change constraint and to ask for solutions where one tries to minimise the number of cycle breaks (i.e., the variable NCHANGE).

See also
- common keyword: change, cyclic_change_joker (number of changes).
- implies: cyclic_change_joker.

Keywords
- characteristic of a constraint: cyclic, automaton, automaton with counters.
- constraint arguments: pure functional dependency.
- constraint network structure: sliding cyclic(1) constraint network(2).
- constraint type: timetabling constraint.
- final graph structure: acyclic, bipartite, no loop.
- modelling: number of changes, functional dependency.
### Arc input(s)

**VARIABLES**

### Arc generator

\[ \text{PATH} \rightarrow \text{collection}(\text{variables1}, \text{variables2}) \]

### Arc arity

2

### Arc constraint(s)

\[(\text{variables1}.\text{var} + 1) \mod \text{CYCLE}\_\text{LENGTH} \leq \text{CTR} \leq \text{variables2}.\text{var} \]

### Graph property(ies)

- **NARC** = \text{NCHANGE}

### Graph class

- ACYCLIC
- BIPARTITE
- NO_LOOP

### Graph model

Parts (A) and (B) of Figure 5.190 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

![Graph Model](image)

**Figure 5.190:** Initial and final graph of the cyclic\_change constraint
Automaton

Figure 5.191 depicts the automaton associated with the cyclic_change constraint. To each pair of consecutive variables \((\text{VAR}_i, \text{VAR}_{i+1})\) of the collection VARIABLES corresponds a 0-1 signature variable \(S_i\). The following signature constraint links \(\text{VAR}_i\), \(\text{VAR}_{i+1}\) and \(S_i\):

\[
((\text{VAR}_i + 1) \mod \text{CYCLE\_LENGTH}) \text{CTR} \text{VAR}_{i+1} \Leftrightarrow S_i.
\]

Figure 5.191: Automaton of the cyclic_change constraint

Figure 5.192: Hypergraph of the reformulation corresponding to the automaton of the cyclic_change constraint
**5.100 cyclic_change_joker**

**Origin**
Derived from cyclic_change.

**Constraint**
cyclic_change_joker(NCHANGE, CYCLE_LENGTH, VARIABLES, CTR)

**Arguments**
- NCHANGE : dvar
- CYCLE_LENGTH : int
- VARIABLES : collection(var–dvar)
- CTR : atom

**Restrictions**
- NCHANGE ≥ 0
- NCHANGE < |VARIABLES|
- CYCLE_LENGTH > 0
- required(VARIABLES, var)
- VARIABLES.var ≥ 0
- CTR ∈ [≠, <, ≥, >, ≤]

**Purpose**
NCHANGE is the number of times that the following constraint holds:

\[(X + 1) \bmod \text{CYCLE\_LENGTH} \neq Y \land X < \text{CYCLE\_LENGTH} \land Y < \text{CYCLE\_LENGTH}\]

X and Y correspond to consecutive variables of the collection VARIABLES.

**Example**

\[
\left\{\begin{array}{c}
\text{var} - 3, \\
\text{var} - 0, \\
\text{var} - 2, \\
\text{var} - 4, \\
\text{var} - 4, \\
\text{var} - 4, \\
\text{var} - 4, \\
\text{var} - 3, \\
\text{var} - 1, \\
\text{var} - 4
\end{array}\right\}, \neq
\]

Since CTR is set to \(\neq\) and since CYCLE_LENGTH is set to 4, a change between two consecutive items \(X\) and \(Y\) of the VARIABLES collection corresponds to the fact that the condition \(((X + 1) \bmod 4) \neq Y \land X < 4 \land Y < 4\) holds. Consequently, the cyclic_change_joker constraint holds since we have the two following changes (i.e., NCHANGE = 2) within \(\{3, 0, 2, 4, 4, 3, 1, 4\}\):

- A first change between 0 and 2,
- A second change between 3 and 1.

But when the joker value 4 is involved, there is no change. This is why no change is counted between values 2 and 4, between 4 and 4 and between 1 and 4.
Typical

NCHANGE > 0
CYCLE_LENGTH > 1
|VARIABLES| > 1
range(VARIABLES.var) > 1
maxval(VARIABLES.var) ≥ CYCLE_LENGTH
CTR ∈ \[\neq\]

Symmetry

Items of VARIABLES can be shifted.

Arg. properties

Functional dependency: NCHANGE determined by CYCLE_LENGTH, VARIABLES and CTR.

Usage

The cyclic_change_joker constraint can be used in the same context as the cyclic_change constraint with the additional feature: in our example codes 0 to 3 correspond to different type of activities (i.e., working the morning, the afternoon or the night) and code 4 represents a holiday. We want to express the fact that we do not count any change for two consecutive days \( d_1, d_2 \) such that \( d_1 \) or \( d_2 \) is a holiday.

See also

common keyword: change, cyclic_change(number of changes).

implied by: cyclic_change.

Keywords

characteristic of a constraint: cyclic, joker value, automaton, automaton with counters.
constraint arguments: pure functional dependency.
constraint network structure: sliding cyclic(1) constraint network(2).
constraint type: timetabling constraint.
final graph structure: acyclic, bipartite, no loop.
modelling: number of changes, functional dependency.
<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>$PATH \rightarrow \text{collection}(\text{variables1, variables2})$</td>
</tr>
<tr>
<td>Arc arity</td>
<td>2</td>
</tr>
</tbody>
</table>
| Arc constraint(s) | $\bullet (\text{variables1.var} + 1) \mod \text{CYCLE\_LENGTH} \leq \text{variables2.var}$
$\bullet \text{variables1.var} < \text{CYCLE\_LENGTH}$
$\bullet \text{variables2.var} < \text{CYCLE\_LENGTH}$ |
| Graph property(ies) | NARC = NCHANGE |
| Graph class | • ACYCLIC
• BIPARTITE
• NO LOOP |

**Graph model**

The *joker values* are those values that are greater than or equal to CYCLE\_LENGTH. We do not count any change for those arc constraints involving at least one variable taking a joker value.

Parts (A) and (B) of Figure 5.193 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

![Graph Diagram](image)

Figure 5.193: Initial and final graph of the cyclic\_change\_joker constraint
Automaton

Figure 5.194 depicts the automaton associated with the cyclic_change_joker constraint. To each pair of consecutive variables \((\text{VAR}_i, \text{VAR}_{i+1})\) of the collection VARIABLES corresponds a 0-1 signature variable \(S_i\). The following signature constraint links \(\text{VAR}_i, \text{VAR}_{i+1}\) and \(S_i\):

\[
((\text{VAR}_i \mod \text{CYCLE_LENGTH}) \text{CTR} \text{VAR}_{i+1} \\
(\text{VAR}_i < \text{CYCLE_LENGTH}) \land (\text{VAR}_{i+1} < \text{CYCLE_LENGTH})) \Leftrightarrow S_i.
\]

Figure 5.194: Automaton of the cyclic_change_joker constraint

\[
\begin{align*}
\text{VAR}_1, & \text{VAR}_2, \text{VAR}_3, \ldots, \text{VAR}_{n-1}, \text{VAR}_n \quad \text{S}_1, \text{S}_2, \ldots, \text{S}_{n-1}, \text{S}_n \quad \{C=0\} & \\
\text{Q}_0 = s, & \text{Q}_1, \text{Q}_2, \ldots, \text{Q}_{n-1} = s, \text{Q}_n & \\
\text{C}_0 = 0, & \text{C}_1, \text{C}_2, \ldots, \text{C}_{n-1} = \text{NCHANGE} \quad \text{NCHANGE} = C + 1
\end{align*}
\]

Figure 5.195: Hypergraph of the reformulation corresponding to the automaton of the cyclic_change_joker constraint
### 5.101 dag

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[131]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>( \text{dag}(\text{NODES}) )</td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>( \text{NODES} : \text{collection}(\text{index} - \text{int}, \text{succ} - \text{svar}) )</td>
<td></td>
</tr>
</tbody>
</table>
| Restrictions| \( \text{required}(\text{NODES}, [\text{index}, \text{succ}]) \)  
\( \text{NODES}.\text{index} \geq 1 \)  
\( \text{NODES}.\text{index} \leq |\text{NODES}| \)  
\( \text{distinct}(\text{NODES}, \text{index}) \)  
\( \text{NODES}.\text{succ} \geq 1 \)  
\( \text{NODES}.\text{succ} \leq |\text{NODES}| \) |       |
| Purpose     | Consider a digraph \( G \) described by the \( \text{NODES} \) collection. Select a subset of arcs of \( G \) so that the corresponding graph does not contain any circuit. |       |

The \( \text{dag} \) constraint holds since the \( \text{NODES} \) collection depicts a graph without circuit.

<table>
<thead>
<tr>
<th>Example</th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| \( \begin{align*} 
\text{index} - 1 & \quad \text{succ} - \{2, 4\}, \\
\text{index} - 2 & \quad \text{succ} - \{3, 4\}, \\
\text{index} - 3 & \quad \text{succ} - 0, \\
\text{index} - 4 & \quad \text{succ} - 0, \\
\text{index} - 5 & \quad \text{succ} - \{6\}, \\
\text{index} - 6 & \quad \text{succ} - 0 
\end{align*} \) |       |       |

Typical \( |\text{NODES}| > 2 \)

Symmetry: Items of \( \text{NODES} \) are permutable.

Algorithm: A filtering algorithm for the \( \text{dag} \) constraint is given in [131, page 90]. It removes potential arcs that would create a circuit of mandatory arcs.

See also: \( \text{used in graph description: in_set} \).

Keywords:
- \( \text{constraint arguments: constraint involving set variables} \).
- \( \text{constraint type: graph constraint} \).
Arc input(s) NODES
Arc generator $SELF \rightarrow \text{collection}(\text{nodes})$
Arc arity 1
Arc constraint(s) $\text{in\_set}(\text{nodes}.\text{key}, \text{nodes}.\text{succ})$
Graph property(ies) $NARC = 0$

Arc input(s) NODES
Arc generator $\text{CLIQUE} \rightarrow \text{collection}(\text{nodes1}, \text{nodes2})$
Arc arity 2
Arc constraint(s) $\text{in\_set}(\text{nodes2}.\text{index}, \text{nodes1}.\text{succ})$
Graph property(ies) $\text{MAX\_NSCC} \leq 1$

Graph model

The first graph constraint removes the loop of each vertex. The second graph constraint forbids the creation of circuits involving more than one vertex.

Part (A) of Figure 5.196 shows the initial graph associated with the second graph constraint of the Example slot. This initial graph from which we start is derived from the set associated with each vertex. Each set describes the potential values of the $\text{succ}$ attribute of a given vertex. Part (B) of Figure 5.196 gives the final graph associated with the Example slot.

![Graph Model Diagram](image)

Figure 5.196: Initial and final graph of the dag set constraint
## 5.102 decreasing

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>decreasing(VARIABLES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>VARIABLES : collection(var–dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restriction</td>
<td>required(VARIABLES, var)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>The variables of the collection VARIABLES are decreasing.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>$((8, 4, 1, 1))$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Typical</td>
<td>$</td>
<td>\text{VARIABLES}</td>
<td>&gt; 1$</td>
</tr>
<tr>
<td>Symmetry</td>
<td>One and the same constant can be added to the var attribute of all items of VARIABLES.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arg. properties</td>
<td>Contractible wrt. VARIABLES.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Systems</td>
<td>$\text{increasingNValue}$ in Choco, rel in Gecode, decreasing in MiniZinc.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>See also</td>
<td>$\text{common keyword: strictly.increasing}$ (order constraint).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keywords</td>
<td>$\text{characteristic of a constraint:}$ automaton, automaton without counters, reified automaton constraint.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{constraint network structure:}$ sliding cyclic(1) constraint network(1).</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{constraint type:}$ decomposition, order constraint.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{filtering:}$ arc-consistency.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{final graph structure:}$ acyclic, bipartite, no loop.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Arc input(s) | VARIABLES
---|---
Arc generator | $PATH \rightarrow collection(variables1, variables2)$
Arc arity | 2
Arc constraint(s) | $variables1.var \geq variables2.var$
Graph property(ies) | $NARC = |VARIABLES| - 1$
Graph class | • ACYCLIC
• BIPARTITE
• NO LOOP

Graph model | Parts (A) and (B) of Figure 5.197 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

![Graph Model](image_url)

Figure 5.197: Initial and final graph of the decreasing constraint
Automaton

Figure 5.198 depicts the automaton associated with the decreasing constraint. To each pair of consecutive variables \((\text{VAR}_i, \text{VAR}_{i+1})\) of the collection \(\text{VARIABLES}\) corresponds a 0-1 signature variable \(S_i\). The following signature constraint links \(\text{VAR}_i, \text{VAR}_{i+1}\) and \(S_i\): \(\text{VAR}_i \geq \text{VAR}_{i+1} \iff S_i\).

![Automaton Diagram](image)

**Figure 5.198: Automaton of the decreasing constraint**

Figure 5.199: Hypergraph of the reformulation corresponding to the automaton of the decreasing constraint
5.103  **deepest_valley**

**DESCRIPTION**

<table>
<thead>
<tr>
<th>Origin</th>
<th>Derived from valley.</th>
</tr>
</thead>
</table>

**LINKS**

**Constraint**

deepest_valley(DEPTH, VARIABLES)

**Arguments**

| DEPTH   | :  dvar            |
| VARIABLES | : collection(var−dvar) |

**Restrictions**

| DEPTH ≥ 0 |
| VARIABLES, var ≥ 0 |
| required(VARIABLES, var) |

**Purpose**

A variable $V_k$ ($1 < k < m$) of the sequence of variables $\text{VARIABLES} = V_1, \ldots, V_m$ is a valley if and only if there exists an $i$ ($1 < i \leq k$) such that $V_{i-1} > V_i$ and $V_i = V_{i+1} = \ldots = V_k$ and $V_k < V_{k+1}$. DEPTH is the minimum value of the valley variables. If no such variable exists DEPTH is equal to the default value MAXINT.

**Example**

\[
\begin{pmatrix}
\text{var − 5}, \\
\text{var − 3}, \\
\text{var − 4}, \\
2, \\
\text{var − 8}, \\
\text{var − 8}, \\
\text{var − 2}, \\
\text{var − 7}, \\
\text{var − 1}
\end{pmatrix}
\]

The deepest_valley constraint holds since 2 is the deepest valley of the sequence 5 3 4 8 8 2 7 1.

Figure 5.200: The sequence and its deepest valley
Typical

\[
\text{DEPTH} \leq \text{maxval}(\text{VARIABLES.var}) \\
|\text{VARIABLES}| > 1 \\
\text{range}(\text{VARIABLES.var}) > 1
\]

Symmetry

Items of \text{VARIABLES} can be reversed.

See also

common keyword: highest peak, valley (sequence).

Keywords

characteristic of a constraint: maxint, automaton, automaton with counters.

combinatorial object: sequence.

constraint network structure: sliding cyclic(1) constraint network(2).
Figure 5.201 depicts the automaton associated with the \textit{deepest_valley} constraint. To each pair of consecutive variables \((\text{VAR}_i, \text{VAR}_{i+1})\) of the collection \textsc{variables} corresponds a signature variable \(S_i\). The following signature constraint links \(\text{VAR}_i\), \(\text{VAR}_{i+1}\) and \(S_i\):

\[\text{VAR}_i < \text{VAR}_{i+1} \iff S_i = 0 \land \text{VAR}_i = \text{VAR}_{i+1} \iff S_i = 1 \land \text{VAR}_i > \text{VAR}_{i+1} \iff S_i = 2.\]

\[\{C = \max \text{int}\}\]

\[\text{VAR} > \text{VAR}_{i+1}; \quad \text{VAR} = \text{VAR}_{i+1}; \quad \text{VAR} < \text{VAR}_{i+1}\]

\[\{C = \min(C, \text{VAR})\}\]

\[\text{DEPTH} = C\]

\[u = s\]

\[\text{C} = \max \text{int}\]

\[\text{C} = \text{DEPTH}\]

\[\text{Q}_0 = s\]

\[\text{C} = \max \text{int}\]

\[\text{Q}_1 = s\]

\[\text{C} = \text{DEPTH}\]

\[\text{Q}_2 = s\]

\[\text{C} = \text{DEPTH}\]

\[\text{Q}_3 = s\]

\[\text{C} = \text{DEPTH}\]

\[\text{Q}_{n-1} = s\]

\[\text{C} = \text{DEPTH}\]

\[\text{Q}_n = s\]

\[\text{C} = \text{DEPTH}\]

Figure 5.202: Hypergraph of the reformulation corresponding to the automaton of the \textit{deepest_valley} constraint
5.104 derangement

**DESCRIPTION**

Origin

Derived from cycle.

Constraint
derangement(NODES)

**LINKS**

**GRAPH**

**Argument**

\[
\text{NODES} : \text{collection}(\text{index-int, succ-dvar})
\]

**Restrictions**

\[
\begin{align*}
|\text{NODES}| & > 1 \\
\text{required(\text{NODES}, \text{index, succ})} \\
\text{\text{NODES}.index} & \geq 1 \\
\text{\text{NODES}.index} & \leq |\text{NODES}| \\
\text{distinct(\text{NODES, index})} \\
\text{\text{NODES}.succ} & \geq 1 \\
\text{\text{NODES}.succ} & \leq |\text{NODES}|
\end{align*}
\]

**Purpose**

Enforce to have a permutation with no cycle of length one. The permutation is depicted by the succ attribute of the NODES collection.

**Example**

\[
\begin{bmatrix}
\text{index}−1 & \text{succ}−2, \\
\text{index}−2 & \text{succ}−1, \\
\text{index}−3 & \text{succ}−5, \\
\text{index}−4 & \text{succ}−3, \\
\text{index}−5 & \text{succ}−4
\end{bmatrix}
\]

In the permutation of the example we have the following 2 cycles: 1 → 2 → 1 and 3 → 5 → 4 → 3. Since these cycles have both a length strictly greater than one the corresponding derangement constraint holds.

**Typical**

\[
|\text{NODES}| > 2
\]

**Symmetries**

- Items of NODES are permutable.
- Attributes of NODES are permutable w.r.t. permutation (index, succ) (permutation applied to all items).

**Remark**

A special case of the cycle [39] constraint.

**See also**

- common keyword: alldifferent, cycle (permutation).
- implies (items to collection): lex_alldifferent.

**Keywords**

- characteristic of a constraint: sort based reformulation.
- combinatorial object: permutation.
- constraint type: graph constraint.
- filtering: arc-consistency.
- final graph structure: one_succ.
Arc input(s) NODES
Arc generator $CLIQUE \rightarrow collection(nnodes_1, nnodes_2)$
Arc arity 2
Arc constraint(s)
• $nnodes_1.succ = nnodes_2.index$
• $nnodes_1.succ \neq nnodes_1.index$
Graph property(ies) NTREE = 0
Graph class ONE_SUCC

Graph model
Parts (A) and (B) of Figure 5.203 respectively show the initial and final graph associated with the Example slot. The derangement constraint holds since the final graph does not contain any vertex that does not belong to a circuit (i.e., NTREE = 0).

In order to express the binary constraint that links two vertices of the NODES collection one has to make explicit the index value of the vertices. This is why the derangement constraint considers objects that have two attributes:

- One fixed attribute index that is the identifier of the vertex,
- One variable attribute succ that is the successor of the vertex.

Forbidding cycles of length one is achieved by the second condition of the arc constraint.

Signature Since 0 is the smallest possible value of NTREE we can rewrite the graph property $NTREE = 0$ to $NTREE \leq 0$. This leads to simplify NTREE to NTREE.
### 5.105 differ_from_at_least_k_pos

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>differ_from_at_least_k_pos(K, VECTOR1, VECTOR2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>VECTOR : collection(var−dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>K : int</td>
<td>VECTOR1 : VECTOR</td>
<td>VECTOR2 : VECTOR</td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>Enforce two vectors VECTOR1 and VECTOR2 to differ from at least K positions.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Example     | \[
|             | \begin{pmatrix} 2, (2,5,2,0), \\
|             | (3,6,2,1) \end{pmatrix} |
|             | The differ_from_at_least_k_pos constraint holds since the first and second vectors differ from 3 positions, which is greater than or equal to K = 2. |
| Typical     | K > 0 | | |
| Symmetries  | • Arguments are permutable w.r.t. permutation (K) (VECTOR1, VECTOR2). |
|             | • K can be decreased to any value ≥ 0. |
|             | • Items of VECTOR1 and VECTOR2 are permutable (same permutation used). |
| Arg. properties | Extensible wrt. VARIABLES1 and VARIABLES2 (add items at same position). |
| Remark      | Used in the Arc constraint(s) slot of the all_differ_from_at_least_k_pos constraint. |
| Used in     | all_differ_from_at_least_k_pos. |
| See also    | system of constraints: all_differ_from_at_least_k_pos. |
| Keywords    | characteristic of a constraint: vector, automaton, automaton with counters. |
|             | constraint network structure: alpha-acyclic constraint network(2). |
|             | constraint type: value constraint. |
Arc input(s) : VECTOR1 VECTOR2
Arc generator : \textit{PRODUCT(=)} \mapsto \textit{collection(vector1, vector2)}
Arc arity : 2
Arc constraint(s) : \textit{vector1.var} \neq \textit{vector2.var}
Graph property(ies) : \textbf{NARC} \geq \textit{k}

Graph model

Parts (A) and (B) of Figure 5.204 respectively show the initial and final graph associated with the \textbf{Example} slot. Since we use the \textbf{NARC} graph property, the arcs of the final graph are stressed in bold.

![Graph diagram](image)

Figure 5.204: Initial and final graph of the \textit{differ from at least k pos} constraint
Figure 5.205 depicts the automaton associated with the `differ_from_at_least_k_pos` constraint. Let $VAR_1_i$ and $VAR_2_i$ be the $i^{th}$ variables of the $VECTOR_1$ and $VECTOR_2$ collections. To each pair of variables $(VAR_1_i, VAR_2_i)$ corresponds a signature variable $S_i$. The following signature constraint links $VAR_1_i$, $VAR_2_i$ and $S_i$: $VAR_1_i = VAR_2_i \iff S_i$.

Figure 5.205: Automaton of the `differ_from_at_least_k_pos` constraint

Figure 5.206: Hypergraph of the reformulation corresponding to the automaton of the `differ_from_at_least_k_pos` constraint
5.106  diffn

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[39]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>diffn(ORTHOTOPE)</td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>disjoint, disjoint1, disjoint2, diff2.</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>ORTHOTOPE : collection(ori-dvar,siz-dvar,end-dvar)</td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>ORTHOTOPE : collection(orth - ORTHOTOPE)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td>ORTHOTOPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>require_at_least(2, ORTHOTOPE, [ori, siz, end])</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ORTHOTOPE.siz ≥ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ORTHOTOPE.ori ≤ ORTHOTOPE.end</td>
</tr>
<tr>
<td></td>
<td></td>
<td>required(ORTHOTOPE, orth)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>same_size(ORTHOTOPE, orth)</td>
</tr>
</tbody>
</table>

Purpose

Generalised multi-dimensional non-overlapping constraint: Holds if, for each pair of orthotopes \(O_1, O_2\), \(O_1\) and \(O_2\) do not overlap. Two orthotopes do not overlap if there exists at least one dimension where their projections do not overlap.

Example

\[
\left(\begin{array}{ccc}
\text{orth} = & \langle \text{ori} - 2, \text{siz} - 2, \text{end} - 4 \rangle \\
\text{orth} = & \langle \text{ori} - 1, \text{siz} - 3, \text{end} - 4 \rangle \\
\text{orth} = & \langle \text{ori} - 4, \text{siz} - 4, \text{end} - 8 \rangle \\
\text{orth} = & \langle \text{ori} - 3, \text{siz} - 3, \text{end} - 6 \rangle \\
\text{orth} = & \langle \text{ori} - 9, \text{siz} - 2, \text{end} - 11 \rangle \\
\text{orth} = & \langle \text{ori} - 4, \text{siz} - 3, \text{end} - 7 \rangle
\end{array}\right)
\]

Figure 5.207 represents the respective position of the three rectangles of the example. The coordinates of the leftmost lowest corner of each rectangle are stressed in bold. The diffn constraint holds since the three rectangles do not overlap.
Typical

|ORTHOTOPE| > 1
|ORTHOTOPE.siz| > 0
|ORTHOTOPE| > 1

Symmetries

- Items of ORTHOTOPEs are permutable.
- Items of ORTHOTOPEs.orth are permutable (same permutation used).
- ORTHOTOPEs.orth.siz can be decreased to any value ≥ 0.
- One and the same constant can be added to the ori and end attributes of all items of ORTHOTOPEs.orth.

Arg. properties

Contractible wrt. ORTHOTOPEs.

Usage

The \texttt{diffn} constraint occurs in placement and scheduling problems. It was for instance used for scheduling problems where one has to both assign each non-preemptive task to a resource and fix its origin so that two tasks, which are assigned to the same resource, do not overlap. When the resource is a set of persons to which non-preemptive tasks have to be assigned this corresponds to so called \textit{timetabling problems}. A second practical application from the area of the design of memory-dominated embedded systems \cite{378} can be found in \cite{379}. Together with arithmetic and \texttt{cumulative} constraints, the \texttt{diffn} constraint was used in \cite{377} for packing more complex shapes such as angles. Figure 5.208 illustrates the angle packing problem on an instance involving 10 angles taken from \cite{377}.

![Figure 5.208: A solution for the angle packing problem of items A1 = [2, 4, 3, 1], A2 = [2, 2, 1, 3], A3 = [1, 3, 3, 2], A4 = [2, 1, 4, 3], A5 = [1, 7, 2, 2], A6 = [1, 2, 5, 5], A7 = [6, 2, 2, 3], A8 = [4, 2, 2, 1], A9 = [3, 1, 1, 4], A10 = [3, 2, 1, 1].](image)

One other packing problem attributed to S. Golomb is to find the smallest square that can contain the set of consecutive squares from $1 \times 1$ up to $n \times n$ so that these squares do not overlap each other (see the smallest rectangle area problem).

Remark

When we have segments (respectively rectangles) the \texttt{diffn} constraint is referenced under the name \texttt{disjoint1} (respectively \texttt{disjoint2}) in \texttt{SICStus Prolog} \cite{94}. When we have rectangles the \texttt{diffn} constraint is also called \texttt{diff2} in \texttt{JaCoP}. In \texttt{MiniZinc} (http://www.g12.cs.mu.oz.au/minizinc/) the \texttt{diffn} constraint considers only rectangles.
It was shown in [381, page 137] that, finding out whether a non-overlapping constraint between a set of rectangles has a solution or not is NP-hard. This was achieved by reduction from sequencing with release times and deadlines.

In the two-dimensional case, when rectangles heights are all equal to one and when rectangles starts in the first dimension are all fixed, the **diffn** constraint can be rewritten as a **k_alldifferent** constraint corresponding to a system of **alldifferent** constraints derived from the maximum cliques of the corresponding interval graph.

Checking whether a **diffn** constraint for which all variables are fixed is satisfied or not is related to the Klee’s measure problem: given a collection of axis-aligned multi-dimensional boxes, how quickly can one compute the volume of their union. Then the **diffn** constraint holds if the volume of the union is equal to the sum of the volumes of the different boxes.

A first possible method for filtering is to use **constructive disjunction**. The idea is to try out each alternative of a disjunction (e.g., given two orthotopes $o_1$ and $o_2$ that should not overlap, we successively assume for each dimension that $o_1$ finishes before $o_2$, and that $o_2$ finishes before $o_1$) and to remove values that were pruned in all alternatives. For the two-dimensional case of **diffn** a second possible solution used in [341] is to represent explicitly the two-dimensional domain of the origin of each rectangle by a quadtree [346] and to accumulate all forbidden regions within this data structure. As for conventional domain variables, a failure occurs when a two-dimensional domain get empty. A third possible filtering algorithm based on **sweep** is described in [31].

The thesis of J. Nelissen [272] considers the case where all rectangles have the same size and can be rotated from 90 degrees (i.e., the pallet loading problem.). For the $n$-dimensional case of **diffn** a filtering algorithm handling the fact that two objects do not overlap is given in [42].

![Figure 5.209: A hard instance from [272, page 165]: A solution for packing 99 rectangles of size $5 \times 9$ into a rectangle of size $86 \times 52$](image)

Extensions of the non-overlapping constraint to polygons and to more complex shapes are respectively described in [42] and in [336]. Specialised propagation algorithms for the **squared squares** problem [80] (based on the fact that no waste is permitted) are given in [167] and in [166].
The *cumulative* constraint can be used as a *necessary condition* for the *diffn* constraint. Figure 5.210 illustrates this point for the two-dimensional case. A first (respectively second) *cumulative* constraint is obtained by forgetting the $y$-coordinate (respectively the $x$-coordinate) of the origin of each rectangle occurring in a *diffn* constraint. Parts (B) and (C) respectively depict the cumulated profiles associated with the projection of the rectangles depicted by part (A) on the $x$ and $y$ axes.

The *cumulative* constraint is a necessary *but not sufficient condition* for the two-dimensional case of the *diffn* constraint. Figure 5.211 illustrates this point on an example taken from [73] where one has to place the 8 rectangles R1, R2, R3, R4, R5, R6, R7, R8 of respective size $5 \times 2$, $8 \times 2$, $6 \times 1$, $5 \times 1$, $2 \times 1$, $3 \times 1$, $2 \times 2$ and $1 \times 2$ in a big rectangle of size $12 \times 4$. As shown by Figure 5.211 there is a cumulative solution where R8 is splitted in two parts but M. Hujter proves in [203] that there is no solution where no rectangle is split.

---

**Figure 5.210:** Looking from the perspective of the *cumulative* constraint in a two-dimensional rectangles placement problem

**Figure 5.211:** Illustrating the necessary but not sufficient placement condition

In the context of $n$ parallelepipeds that have to be packed [172, 243] within a box of sizes $X \times Y \times Z$ one can proceed as follows for stating three *cumulative* constraints. The $i^{th}$
(\(i \in [1,n]\)) parallelepiped is described by the following attributes:

- \(o_x, o_y, o_z (i \in [1,n])\) the coordinates of its origin on the \(x, y\) and \(z\)-axes.
- \(s_x, s_y, s_z (i \in [1,n])\) its sizes on the \(x, y\) and \(z\)-axes.
- \(p_x, p_y, p_z (i \in [1,n])\) the surfaces of its projections on the planes \(yz, xz,\) and \(xy\) respectively equal to \(sy, sz, sx, sz,\) and \(sx, sy,\)
- \(v_i\) its volume (equal to \(sx \cdot sy \cdot sz\)).

For the placement of \(n\) parallelepipeds we get the following necessary conditions that respectively correspond to three cumulative constraints on the planes \(yz, xz,\) and \(xy:\)

\[
\forall i \in [1,X] : \sum_{j \mid o_x \leq i \leq o_x + s_x - 1} p_x_j \leq YZ
\]
\[
\forall i \in [1,Y] : \sum_{j \mid o_y \leq i \leq o_y + s_y - 1} p_y_j \leq XZ
\]
\[
\forall i \in [1,Z] : \sum_{j \mid o_z \leq i \leq o_z + s_z - 1} p_z_j \leq XY
\]

**Reformulation**

Based on the fact that two orthotopes do not overlap if there exists at least one dimension where their projections do not overlap one can reformulate the \texttt{diffn(ORTHOTOPES)} constraint as a disjunction of inequalities between the origin and the end attributes. In addition one has to link the origin, the size and the end attributes of each orthotope in each dimension.

If we consider the example described in the **Example** slot we get the following reformulation:

- \(4 = 2 + 2\) (link between the origin, size and end in dimension 1 of the first orthotope),
- \(4 = 1 + 3\) (link between the origin, size and end in dimension 2 of the first orthotope),
- \(8 = 4 + 4\) (link between the origin, size and end in dimension 1 of the second orthotope),
- \(6 = 3 + 3\) (link between the origin, size and end in dimension 2 of the second orthotope),
- \(11 = 9 + 2\) (link between the origin, size and end in dimension 1 of the third orthotope),
- \(7 = 4+3\) (link between the origin, size and end in dimension 2 of the third orthotope),
- \(4 \leq 4 \vee 8 \leq 2 \vee 4 \leq 3 \vee 6 \leq 1\) (non-overlapping between the first and second orthotopes),
- \(4 \leq 9 \vee 11 \leq 2 \vee 4 \leq 4 \vee 7 \leq 1\) (non-overlapping between the first and third orthotopes),
- \(8 \leq 9 \vee 11 \leq 4 \vee 6 \leq 4 \vee 7 \leq 3\) (non-overlapping between the second and third orthotopes).

diffn_column, diffn_include, place_in_pyramid.

common keyword: calendar (multi-site employee scheduling with calendar constraints, scheduling with machine choice, calendars and preemption),
diffn_column, diffn_include (geometrical constraint, orthotope),
non_overlap_sboxes (geometrical constraint, non-overlapping),
visible (geometrical constraint).

implied by: orths_are_connected.

implies: cumulative (implies one cumulative constraint for each dimension).

related: cumulative_two_d (cumulative_two_d is a necessary condition for diffn: forget one dimension when the number of dimensions is equal to 3),
lex_chain_less, lex_chain_less_eq (lexicographic ordering on the origins of tasks, rectangles, ...),
two_orth_column, two_orth_include.

specialisation: all_min_dist (orthotope replaced by line segment, of same length),
all_different (orthotope replaced by variable),
cumulatives (orthotope replaced by task with machine assignment and origin attributes),
disjunctive (orthotope replaced by task of height 1),
lex_all_different (when rectangles heights are all equal to 1 and rectangles starts in the first dimension are all fixed),
lex_lex chain_less (orthotope replaced by vector).

used in graph description: orth_link_ori_siz_end, two_orth_do_not_overlap.

application area: floor planning problem.

characteristic of a constraint: core.

combinatorial object: pentomino.

complexity: sequencing with release times and deadlines.

constraint arguments: business rules.

constraint type: decomposition, timetabling constraint, relaxation.

filtering: Klee measure problem, sweep, quadtree, compulsory part, constructive disjunction, SAT.

geometry: geometrical constraint, orthotope, polygon, non-overlapping.

heuristics: heuristics for two-dimensional rectangle placement problems.

modelling: disjunction, assignment dimension, assignment to the same set of values, assigning and scheduling tasks that run in parallel, relaxation dimension, sequence dependent set-up, multi-site employee scheduling with calendar constraints, scheduling with machine choice, calendars and preemption.

modelling exercises: assignment to the same set of values, assigning and scheduling tasks that run in parallel, relaxation dimension, sequence dependent set-up, multi-site employee scheduling with calendar constraints, scheduling with machine choice, calendars and preemption.

problems: strip packing, two-dimensional orthogonal packing, pallet loading.

puzzles: squared squares, packing almost squares, Partridge, pentomino, Shikaku, smallest square for packing consecutive dominoes,
smallest square for packing rectangles with distinct sizes, smallest rectangle area, Conway packing problem.
Arc input(s) | ORTHOTOPES  
---|---
Arc generator | SELF\(\rightarrow\)collection(orthotopes)  
Arc arity | 1  
Arc constraint(s) | orth\_link\_ori\_siz\_end(orthotopes.orth)  
Graph property(ies) | NARC = |ORTHOTOPES|

Arc input(s) | ORTHOTOPES  
---|---
Arc generator | CLIQUE(\(\neq\))\(\rightarrow\)collection(orthotopes1, orthotopes2)  
Arc arity | 2  
Arc constraint(s) | two\_orth\_do\_not\_overlap(orthotopes1.orth, orthotopes2.orth)  
Graph property(ies) | NARC = |ORTHOTOPES| + |ORTHOTOPES| − |ORTHOTOPES|

Graph model

The diffn constraint is expressed by using two graph constraints:

- The first graph constraint enforces for each dimension and for each orthotope the link between the corresponding ori, siz and end attributes.
- The second graph constraint imposes each pair of distinct orthotopes to not overlap.

Parts (A) and (B) of Figure 5.212 respectively show the initial and final graph associated with the second graph constraint of the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

![Graph Model Diagram]

Figure 5.212: Initial and final graph of the diffn constraint

Signature

Since |ORTHOTOPES| is the maximum number of vertices of the final graph of the first graph constraint we can rewrite NARC = |ORTHOTOPES| to NARC \(\geq\) |ORTHOTOPES|. This leads to simplify NARC to NARC.
Since we use the $\text{CLIQUE}(\neq)$ arc generator on the ORTHOTOPES collection, $|\text{ORTHOTOPES}| \cdot |\text{ORTHOTOPES}| - |\text{ORTHOTOPES}|$ is the maximum number of vertices of the final graph of the second graph constraint. Therefore we can rewrite $\text{NARC} = |\text{ORTHOTOPES}| \cdot |\text{ORTHOTOPES}| - |\text{ORTHOTOPES}|$ to $\text{NARC} \geq |\text{ORTHOTOPES}| \cdot |\text{ORTHOTOPES}| - |\text{ORTHOTOPES}|$. Again, this leads to simplify $\text{NARC}$ to $\text{NARC}$. 
5.107 **diffn_column**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>CHIP: option guillotine cut (column) of <strong>diffn</strong>.</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>diffn_column(ORTHOTOPES, DIM)</td>
<td></td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>ORTHOTOPE : collection(ori-dvar, siz-dvar, end-dvar)</td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>ORTHOTOPES : collection(orth - ORTHOTOPE)</td>
<td>DIM : int</td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| ORTHOTOPE | > 0
| require_at_least(2, ORTHOTOPE, [ori, siz, end])
| ORTHOTOPE.siz ≥ 0
| ORTHOTOPE.ori ≤ ORTHOTOPE.end
| required(ORTHOTOPES, orth)
| same_size(ORTHOTOPES, orth)
| DIM > 0
| DIM ≤ |ORTHOTOPE|
| diffn(ORTHOTOPES) |

Extension of the generalised multi-dimensional non-overlapping **diffn** constraint. Holds if, for each pair of orthotopes \((O_1, O_2)\) the following conditions hold:

- \(O_1\) and \(O_2\) do not overlap. Two orthotopes do not overlap if there exists at least one dimension where their projections do not overlap.
- Let \(P_1\) and \(P_2\) respectively denote the projections of \(O_1\) and \(O_2\) in dimension DIM. If \(P_1\) and \(P_2\) overlap then the size of their intersection is equal to the size of \(O_1\) in dimension DIM, as well as to the size of \(O_2\) in dimension DIM.

### Example

$$\begin{align*}
\text{orth} = & \begin{cases}
\text{ori} - 1 & \text{siz} - 3 & \text{end} - 4, \\
\text{ori} - 3 & \text{siz} - 2 & \text{end} - 5
\end{cases}, \\
\text{orth} = & \begin{cases}
\text{ori} - 9 & \text{siz} - 1 & \text{end} - 10, \\
\text{ori} - 4 & \text{siz} - 3 & \text{end} - 7
\end{cases}, \\
\text{orth} = & \begin{cases}
\text{ori} - 4 & \text{siz} - 2 & \text{end} - 6, \\
\text{ori} - 3 & \text{siz} - 4 & \text{end} - 7
\end{cases}, \\
\text{orth} = & \begin{cases}
\text{ori} - 1 & \text{siz} - 3 & \text{end} - 4, \\
\text{ori} - 6 & \text{siz} - 1 & \text{end} - 7
\end{cases}, \\
\text{orth} = & \begin{cases}
\text{ori} - 6 & \text{siz} - 2 & \text{end} - 8, \\
\text{ori} - 1 & \text{siz} - 4 & \text{end} - 5
\end{cases}, \\
\text{orth} = & \begin{cases}
\text{ori} - 10 & \text{siz} - 1 & \text{end} - 11, \\
\text{ori} - 1 & \text{siz} - 1 & \text{end} - 2
\end{cases}, \\
\text{orth} = & \begin{cases}
\text{ori} - 9 & \text{siz} - 1 & \text{end} - 10, \\
\text{ori} - 1 & \text{siz} - 1 & \text{end} - 2
\end{cases}, \\
\text{orth} = & \begin{cases}
\text{ori} - 6 & \text{siz} - 2 & \text{end} - 8, \\
\text{ori} - 6 & \text{siz} - 1 & \text{end} - 7
\end{cases}
\end{align*}$$
Figure 5.213 represents the respective position of the eight rectangles of the example. The coordinates of the leftmost lowest corner of each rectangle are stressed in bold. The $\text{diffn.column}$ constraint holds since (1) the eight rectangles do not overlap and since (2) when their projection in dimension $\text{DIM} = 1$ overlap the size of their intersection is equal to the size of the corresponding rectangles in dimension $\text{DIM} = 1$.

![Diagram of eight non-overlapping rectangles](image)

Figure 5.213: Eight non-overlapping rectangles such that, for each pair of rectangles $R_i, R_j$ ($1 \leq i < j \leq 12$), if the projections in dimension 1 of rectangles $R_i$ and $R_j$ intersect then the size of their intersection is equal to the size of $R_i$ in dimension 1 and to the size of $R_j$ in dimension 1.

**Typical**

\[
\begin{align*}
\text{ORTHOTOPE} & > 1 \\
\text{ORTHOTOPE.siz} & > 0 \\
\text{ORTHOTOPE.siz} & > 1
\end{align*}
\]

**Symmetries**

- Items of \text{ORTHOTOPE}s are \text{permutable}.
- One and the same constant can be \text{added} to the \text{ori} and \text{end} attributes of all items of \text{ORTHOTOPE}s.orth.

**Arg. properties**

\text{Contractible} wrt. \text{ORTHOTOPE}s.

**See also**

- common keyword: $\text{diffn(geometrical constraint,orthotope)}$.
- $\text{diffn.include(geometrical constraint,orthotope,positioning constraint)}$.
- \text{implies: diffn.include}.
- \text{used in graph description: two.orth.column}.

**Keywords**

- constraint type: \text{decomposition}.
- geometry: geometrical constraint, positioning constraint, orthotope, guillotine cut.
Arc input(s)  \text{ORTHOTOPES} \\
Arc generator  \text{CLIQUE}(\leq) \mapsto \text{collection}(\text{orthotopes}_1, \text{orthotopes}_2) \\
Arc arity  2 \\
Arc constraint(s)  \text{two}\_\text{orth}\_\text{column}(\text{orthotopes}_1.\text{orth}, \text{orthotopes}_2.\text{orth}, \text{DIM}) \\
Graph property(ies)  \text{NARC} = |\text{ORTHOTOPES}| \ast (|\text{ORTHOTOPES}| - 1)/2 \\

Graph model  
Since showing all items produces too big graphs, parts (A) and (B) of Figure 5.214 respectively show the initial and final graph associated with the first three items of the \textbf{Example} slot. Since we use the \textbf{NARC} graph property, the arcs of the final graph are stressed in bold.

![Graphs](image)

(A)  
(B)  

Figure 5.214: Initial and final graph of the \texttt{diffn}\_\texttt{column} constraint
5.108  **diffn_include**

**DESCRIPTION**

**Origin**
CHIP: option guillotine cut (include) of *diffn*.

**Constraint**
`diffn_include(ORTHOTOPES, DIM)`

**Type**
ORTHOPOSE : `collection(ori-dvar, siz-dvar, end-dvar)`

**Arguments**
ORTHOTOPES : `collection(orth – ORTHOTOPE)`
DIM : `int`

**Restrictions**

| ORTHOTOPE | > 0
|---|---
| `require_at_least(2, ORTHOTOPE, [ori, siz, end])` |
| ORTHOTOPE.siz ≥ 0 |
| ORTHOTOPE.ori ≤ ORTHOTOPE.end |
| `required(ORTHOTOPES, orth)` |
| `same_size(ORTHOTOPES, orth)` |
| DIM > 0 |
| DIM ≤ |ORTHOTOPE| |
| `diffn(ORTHOTOPES)` |

**Purpose**

Extension of the generalised multi-dimensional non-overlapping `diffn` constraint. Holds if, for each pair of orthotopes \((O_1, O_2)\) the following conditions hold:

- \(O_1\) and \(O_2\) do not overlap. Two orthotopes do not overlap if there exists at least one dimension where their projections do not overlap.

- Let \(P_1\) and \(P_2\) respectively denote the projections of \(O_1\) and \(O_2\) in dimension \(DIM\). If \(P_1\) and \(P_2\) overlap then, either \(P_1\) is included in \(P_2\), either \(P_2\) is included in \(P_1\).
Example

Figure 5.215 represents the respective position of the twelve rectangles of the example. The coordinates of the leftmost lowest corner of each rectangle are stressed in bold. The diffn. include constraint holds since (1) the twelve rectangles do not overlap and since (2) when their projection in dimension DIM = 1 overlap one of the projections is included within the other one.

Figure 5.215: Twelve non-overlapping rectangles such that, for each pair of rectangles $R_i$, $R_j$ (1 ≤ $i < j$ ≤ 12), if the projections in dimension 1 of rectangles $R_i$ and $R_j$ intersect then one of the projections is included within the other projection.
### Typical

|ORTHOTOPE| > 1  
|ORTHOTOPE.siz| > 0  
|ORTHOTOPES| > 1

### Symmetries

- Items of ORTHOTOPES are permutable.
- One and the same constant can be added to the ori and end attributes of all items of ORTHOTOPES.orth.

### Arg. properties

Contractible wrt. ORTHOTOPES.

### See also

**common keyword:**

- `diffn(geometrical constraint, orthotope),
- `diffn_column(geometrical constraint, orthotope, positioning constraint)`.

**implied by:** `diffn_column`.

**used in graph description:** `two_orth_column`.

### Keywords

- **constraint type:** decomposition.
- **geometry:** geometrical constraint, positioning constraint, orthotope.
Arc input(s)  ORTHOTOPES
Arc generator  $\text{CLIQUE}(\langle \rangle) \rightarrow \text{collection}(\text{orthotopes1}, \text{orthotopes2})$
Arc arity  2
Arc constraint(s)  $\text{two\_orth\_include}(\text{orthotopes1.orth}, \text{orthotopes2.orth}, \text{DIM})$
Graph property(ies)  $\text{NARC} = |\text{ORTHOTOPES}| \ast (|\text{ORTHOTOPES}| - 1)/2$

Graph model  Since showing all items produces too big graphs, parts (A) and (B) of Figure 5.216 respectively show the initial and final graph associated with the first three items of the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

![Graph Model](image)

Figure 5.216: Initial and final graph of the diffn_incluse constraint
5.109  discrepancy

**Description**  
Constraint: discrepancy(VARIABLES, K)

**Arguments**  
- VARIABLES: collection(var - dvar, bad - sint)
- K: int

**Restrictions**  
- required(VARIABLES, var)
- required(VARIABLES, bad)
- K ≥ 0
- K ≤ |VARIABLES|

**Purpose**  
K is the number of variables of the collection VARIABLES that take their value in their respective sets of bad values.

**Example**  
\[
\begin{pmatrix}
\text{var} - 4 & \text{bad} = \{1, 4, 6\}, \\
\text{var} - 5 & \text{bad} = \{0, 1\}, \\
\text{var} - 5 & \text{bad} = \{1, 6, 9\}, \\
\text{var} - 4 & \text{bad} = \{1, 4\}, \\
\text{var} - 1 & \text{bad} = \emptyset
\end{pmatrix}
\]

The discrepancy constraint holds since exactly K = 2 variables (i.e., the first and fourth variables) of the VARIABLES collection take their value within their respective sets of bad values.

**Typical**  
- |VARIABLES| > 1
- K < |VARIABLES|

**Symmetries**  
- Items of VARIABLES are permutable.
- All occurrences of two distinct values in VARIABLES.var or VARIABLES.bad can be swapped; all occurrences of a value in VARIABLES.var or VARIABLES.bad can be renamed to any unused value.

**Arg. properties**  
- Functional dependency: K determined by VARIABLES.
- Aggregate: VARIABLES(union), K(+).

**Remark**  
Limited discrepancy search was first introduced by M. L. Ginsberg and W. D. Harvey as a search technique in [178]. Later on, discrepancy based filtering was presented in the PhD thesis of F. Focacci [157, pages 171–172]. Finally the discrepancy constraint was explicitly defined in the PhD thesis of W.-J. van Hoeve [398, page 104].

**See also**  
- **common keyword**: among (counting constraint).
- used in graph description: in_set.
Keywords

- **constraint arguments**: pure functional dependency.
- **constraint type**: value constraint, counting constraint.
- **filtering**: arc-consistency.
- **heuristics**: heuristics, limited discrepancy search.
- **modelling**: functional dependency.
<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>SELF↦collection(variables)</td>
</tr>
<tr>
<td>Arc arity</td>
<td>1</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>in_set(variables.var, variables.bad)</td>
</tr>
<tr>
<td>Graph property(ies)</td>
<td>NARC=k</td>
</tr>
</tbody>
</table>

**Graph model**

The arc constraint corresponds to the constraint \(\text{in\_set}(\text{variables.var}, \text{variables.bad})\) defined in this catalogue. We employ the \(\text{SELF}\) arc generator in order to produce an initial graph with a single loop on each vertex.

Parts (A) and (B) of Figure 5.217 respectively show the initial and final graph associated with the Example slot. Since we use the \(\text{NARC}\) graph property, the loops of the final graph are stressed in bold.

![Graph](image)

(A) (B)

Figure 5.217: Initial and final graph of the discrepancy constraint
5.110 disj

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[267]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>disj(TASKS)</td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>TASKS : collection(start−dvar, duration−dvar, before−svar, position−dvar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(TASKS,[start, duration, before, position]) TASKS.duration ≥ 1 TASKS.position ≥ 0 TASKS.position &lt;</td>
<td></td>
</tr>
</tbody>
</table>

**Purpose**

All the tasks of the collection TASKS should not overlap. For a given task \( t \) the attributes before and position respectively correspond to the set of tasks starting before task \( t \) (assuming that the first task is labelled by 1) and to the position of task \( t \) (assuming that the first task has position 0).

**Example**

\[
\begin{align*}
\text{start} & -1 & \text{duration} & -3 & \text{before} & = \emptyset & \text{position} & -0, \\
\text{start} & -9 & \text{duration} & -1 & \text{before} & = \{1, 3, 4\} & \text{position} & -3, \\
\text{start} & -7 & \text{duration} & -2 & \text{before} & = \{1, 4\} & \text{position} & -2, \\
\text{start} & -4 & \text{duration} & -1 & \text{before} & = \{1\} & \text{position} & -1
\end{align*}
\]

Figure 5.218 shows the tasks of the example. Since these tasks do not overlap the disj constraint holds.

Figure 5.218: Tasks

**Typical**

\(|\text{TASKS}| > 1\)

**Symmetries**

- One and the same constant can be added to the start attribute of all items of TASKS.
- TASKS.duration can be decreased to any value \( \geq 1 \).

**Usage**

The disj constraint was originally applied [267] to solve the open-shop problem.
Remark
This constraint is similar to the disjunctive constraint. In addition to the start and the duration attributes of a task \( t \), the \( \text{disj} \) constraint introduces a set variable \( \text{before} \) that represents the set of tasks that end before the start of task \( t \) as well as a domain variable \( \text{position} \) that gives the absolute order of task \( t \) in the resource. Since it assumes that the first task has position 0 we have that, for a given task \( t \), the number of elements of its \( \text{before} \) attribute is equal to the value of its \( \text{position} \) attribute.

Algorithm
The main idea of the algorithm is to apply in a systematic way shaving on the \( \text{position} \) attribute of a task. It is implemented in Gecode [353].

See also
common keyword: disjunctive (scheduling constraint).
used in graph description: in_set.

Keywords
complexity: sequencing with release times and deadlines.
constraint arguments: constraint involving set variables.
constraint type: scheduling constraint, resource constraint, decomposition.
Arc input(s)  TASKS
Arc generator  \( CLIQUE(\neq) \mapsto \text{collection}(\text{tasks}1,\text{tasks}2) \)
Arc arity  2
Arc constraint(s)  
- \( \bigvee \left( \begin{array}{c} \text{tasks}1.\text{start} + \text{tasks}1.\text{duration} \leq \text{tasks}2.\text{start}, \\ \text{tasks}2.\text{start} + \text{tasks}2.\text{duration} \leq \text{tasks}1.\text{start} \end{array} \right) \)
- \( \text{tasks}1.\text{start} + \text{tasks}1.\text{duration} \leq \text{tasks}2.\text{start} \iff \text{in}\_\text{set}(\text{tasks}1.\text{key}, \text{tasks}2.\text{before}) \)
- \( \text{tasks}1.\text{start} + \text{tasks}1.\text{duration} \leq \text{tasks}2.\text{start} \iff \text{tasks}1.\text{position} < \text{tasks}2.\text{position} \)
Graph property(ies)  \( \text{NARC} = |\text{TASKS}| \times |\text{TASKS}| - |\text{TASKS}| \)

Graph model  We generate a clique with a non-overlapping constraint between each pair of distinct tasks and state that the number of arcs of the final graph should be equal to the number of arcs of the initial graph. For two tasks \( t_1 \) and \( t_2 \), the three conditions of the arc constraint respectively correspond to:

- The fact that \( t_1 \) ends before the start of \( t_2 \) or that \( t_2 \) ends before the start of \( t_1 \).
- The equivalence between the fact that \( t_1 \) ends before the start of \( t_2 \) and the fact that the identifier of task \( t_1 \) belongs to the before attribute of task \( t_2 \).
- The equivalence between the fact that \( t_1 \) ends before the start of \( t_2 \) and the fact that the position attribute of task \( t_1 \) is strictly less than the position attribute of task \( t_2 \).

Parts (A) and (B) of Figure 5.219 respectively show the initial and final graph associated with the Example slot. The \( \text{disj} \) constraint holds since all the arcs of the initial graph belong to the final graph: all the non-overlapping constraints holds.

![Figure 5.219: Initial and final graph of the \( \text{disj} \) constraint](image)

(A)  (B)
5.111 disjoint

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from alldifferent.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>disjoint(VARIABLES1, VARIABLES2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VARIABLES1 : collection(var−dvar)</td>
<td>VARIABLES2 : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(VARIABLES1, var)</td>
<td>required(VARIABLES2, var)</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>Each variable of the collection VARIABLES1 should take a value that is distinct from all the values assigned to the variables of the collection VARIABLES2.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example

\[
\begin{pmatrix}
(1, 9, 1, 5), \\
\text{var − 2,} \\
\text{var − 7,} \\
\langle \text{var − 7,} \\
\text{var − 0,} \\
\text{var − 6,} \\
\text{var − 8}
\end{pmatrix}
\]

In this example, values 1, 5, 9 are used by the variables of VARIABLES1 and values 0, 2, 6, 7, 8 by the variables of VARIABLES2. Since there is no intersection between the two previous sets of values the disjoint constraint holds.

Typical

\[
\begin{align*}
|\text{VARIABLES1}| & > 1 \\
|\text{VARIABLES2}| & > 1
\end{align*}
\]

Symmetries

- Arguments are permutable w.r.t. permutation (VARIABLES1, VARIABLES2).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value of VARIABLES1.var can be replaced by any value of VARIABLES1.var.
- An occurrence of a value of VARIABLES2.var can be replaced by any value of VARIABLES2.var.
- All occurrences of two distinct values in VARIABLES1.var or VARIABLES2.var can be swapped; all occurrences of a value in VARIABLES1.var or VARIABLES2.var can be renamed to any unused value.

Arg. properties

- Contractible wrt. VARIABLES1.
- Contractible wrt. VARIABLES2.
Remark

Despite the fact that this is not an uncommon constraint, it can not be modelled in a compact way neither with a disequality constraint (i.e., two given variables have to take distinct values) nor with the alldifferent constraint. The disjoint constraint can bee seen as a special case of the common(NCOMMON1, NCOMMON2, VARIABLES1, VARIABLES2) constraint where NCOMMON1 and NCOMMON2 are both set to 0.

MiniZinc (http://www.g12.cs.mu.oz.au/minizinc/) has a disjoint constraint between two set variables rather than between two collections of variables.

Algorithm

Let us note:

• $n_1$ the minimum number of distinct values taken by the variables of the collection VARIABLES1.
• $n_2$ the minimum number of distinct values taken by the variables of the collection VARIABLES2.
• $n_{12}$ the maximum number of distinct values taken by the union of the variables of VARIABLES1 and VARIABLES2.

One invariant to maintain for the disjoint constraint is $n_1 + n_2 \leq n_{12}$. A lower bound of $n_1$ and $n_2$ can be obtained by using the algorithms provided in [26, 38]. An exact upper bound of $n_{12}$ can be computed by using a bipartite matching algorithm.

Used in

k_disjoint.

See also

generalisation: disjoint_tasks (variable replaced by task).

implies: alldifferent_on_intersection, lex_differernt.

system of constraints: k_disjoint.

Keywords

characteristic of a constraint: disequality, automaton, automaton with array of counters.

constraint type: value constraint.

filtering: bipartite matching.

modelling: empty intersection.
Arc input(s)  VARIABLES1 VARIABLES2
Arc generator  \( \text{PRODUCT} \rightarrow \text{collection}(\text{variables1}, \text{variables2}) \)
Arc arity  2
Arc constraint(s)  \( \text{variables1}.\text{var} = \text{variables2}.\text{var} \)
Graph property(ies)  \( \text{NARC} = 0 \)

Graph model  \( \text{PRODUCT} \) is used in order to generate the arcs of the graph between all variables of VARIABLES1 and all variables of VARIABLES2. Since we use the graph property \( \text{NARC} = 0 \) the final graph will be empty. Figure 5.220 shows the initial graph associated with the Example slot. Since we use the \( \text{NARC} = 0 \) graph property the final graph is empty.

Figure 5.220: Initial graph of the disjoint constraint (the final graph is empty)

Signature  Since 0 is the smallest number of arcs of the final graph we can rewrite \( \text{NARC} = 0 \) to \( \text{NARC} \leq 0 \). This leads to simplify \( \text{NARC} \) to \( \text{NARC} \).
Automaton

Figure 5.221 depicts the automaton associated with the disjoint constraint. To each variable $\text{VAR}_1_i$ of the collection $\text{VARIABLES}_1$ corresponds a signature variable $S_i$ that is equal to 0. To each variable $\text{VAR}_2_i$ of the collection $\text{VARIABLES}_2$ corresponds a signature variable $S_{i+}\vert\text{VARIABLES}_1\vert$ that is equal to 1.

$\{C[\_]=0,D[\_]=0\}$

$0,$

$\{C[\text{VAR}_1_{i}]=C[\text{VAR}_1_{i}]+1\}$

$1,$

$\{D[\text{VAR}_2_{i}]=D[\text{VAR}_2_{i}]+1\}$

$t:$

$\text{arith\_or}(C,D,\lt,1)$

$1,$

$\{D[\text{VAR}_2_{i}]=D[\text{VAR}_2_{i}]+1\}$

Figure 5.221: Automaton of the disjoint constraint
## 5.112 disjoint_sboxes

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>LOGIC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Geometry, derived from [318]</td>
<td></td>
</tr>
</tbody>
</table>

**Constraint**

\[
\text{disjoint}_\text{sboxes}(K, \text{DIMS}, \text{OBJECTS}, \text{SBOXES})
\]

**Synonym**

\[
\text{disjoint}.
\]

**Types**

\[
\begin{align*}
\text{VARIABLES} & : \text{\textit{collection}}(v-\text{dvar}) \\
\text{INTEGERS} & : \text{\textit{collection}}(v-\text{int}) \\
\text{POSITIVES} & : \text{\textit{collection}}(v-\text{int})
\end{align*}
\]

**Arguments**

\[
\begin{align*}
K & : \text{\textit{int}} \\
\text{DIMS} & : \text{\textit{sint}} \\
\text{OBJECTS} & : \text{\textit{collection}}(\text{oid-int}, \text{sid-int}, x - \text{VARIABLES}) \\
\text{SBOXES} & : \text{\textit{collection}}(\text{sid-int}, t - \text{INTEGERS}, l - \text{POSITIVES})
\end{align*}
\]

**Restrictions**

\[
\begin{align*}
|\text{VARIABLES}| & \geq 1 \\
|\text{INTEGERS}| & \geq 1 \\
|\text{POSITIVES}| & \geq 1 \\
\text{required}(\text{VARIABLES}, v) & \\
|\text{VARIABLES}| & = K \\
\text{required}(\text{INTEGERS}, v) & \\
|\text{INTEGERS}| & = K \\
\text{required}(\text{POSITIVES}, v) & \\
|\text{POSITIVES}| & = K \\
\text{POSITIVES}.v & > 0 \\
K & > 0 \\
\text{DIMS} & \geq 0 \\
\text{DIMS} & < K \\
\text{increasing_seq}(\text{OBJECTS}, [\text{oid}]) & \\
\text{required}(\text{OBJECTS}, [\text{oid}, \text{sid}, x]) & \\
\text{OBJECTS}.\text{oid} & \geq 1 \\
\text{OBJECTS}.\text{oid} \leq |\text{OBJECTS}| & \\
\text{OBJECTS}.\text{sid} & \geq 1 \\
\text{OBJECTS}.\text{sid} \leq |\text{SBOXES}| & \\
|\text{SBOXES}| & \geq 1 \\
\text{required}(\text{SBOXES}, [\text{sid}, t, l]) & \\
\text{SBOXES}.\text{sid} & \geq 1 \\
\text{SBOXES}.\text{sid} \leq |\text{SBOXES}| & \\
\text{do_not_overlap}(\text{SBOXES}) &
\end{align*}
\]
Holds if, for each pair of objects \((O_i, O_j), \ i \neq j\), \(O_i\) and \(O_j\) are disjoint with respect to a set of dimensions depicted by \(\text{DIMS}\). \(O_i\) and \(O_j\) are objects that take a shape among a set of shapes. Each \textit{shape} is defined as a finite set of shifted boxes, where each shifted box is described by a box in a \(K\)-dimensional space at a given offset (from the origin of the shape) with given sizes. More precisely, a \textit{shifted box} is an entity defined by its shape \(\text{id}\), shift offset \(t\), and sizes \(1\). Then, a shape is defined as the union of shifted boxes sharing the same \text{id}. An \textit{object} is an entity defined by its unique object identifier \(\text{oid}\), \text{id} \(\text{sid}\) and origin \(x\).

Two objects \(O_i\) and object \(O_j\) are \textit{disjoint} with respect to a set of dimensions depicted by \(\text{DIMS}\) if and only if for all shifted box \(s_i\) associated with \(O_i\) and for all shifted box \(s_j\) associated with \(O_j\) there exists at least one dimension \(d \in \text{DIMS}\) such that (1) the origin \(s_i\) in dimension \(d\) is strictly greater than the end of \(s_j\) in dimension \(d\), or (2) the origin \(s_j\) in dimension \(d\) is strictly greater than the end of \(s_i\) in dimension \(d\).

Figure 5.222 shows the objects of the example. Since these objects are pairwise disjoint the \textit{disjoint_sboxes} constraint holds.

\[
\begin{align*}
2, \{0,1\}, \\
\begin{cases}
\text{oid} = 1, \ 	ext{sid} = 1, \ x - \{1,1\}, \\
\text{oid} = 2, \ 	ext{sid} = 2, \ x - \{4,1\}, \\
\text{oid} = 3, \ 	ext{sid} = 4, \ x - \{2,4\}
\end{cases}
\end{align*}
\]

**Purpose**

\text{OBJECTS} > 1

**Symmetries**

- Items of \text{OBJECTS} are \textit{permutable}.
- Items of \text{SBOXES} are \textit{permutable}.
- \text{SBOXES}1.v can be \textit{decreased} to any value \(\geq 1\).

**Arg. properties**

\text{Suffix-contractible} wrt. \text{OBJECTS}.

**Remark**

One of the eight relations of the \textit{Region Connection Calculus} [318]. Unlike the \textit{non_overlap_sboxes} constraint, which just prevents objects from overlapping, the \textit{disjoint_sboxes} constraint in addition enforces that borders and corners of objects are not directly in contact.

*common keyword*: \text{contains_sboxes}, \text{coveredby_sboxes}, \text{covers_sboxes}, \text{equal_sboxes}, \text{inside_sboxes}, \text{meet_sboxes(rcc8)}, \text{non_overlap_sboxes}(\text{geometrical constraint,logic}), \text{overlap_sboxes}(rcc8).
Keywords

- **constraint type**: logic.
- **geometry**: geometrical constraint, rcc8.
Figure 5.222: The three mutually disjoint objects of the example
• origin(O1, S1, D) \triangleq 01.x(D) + S1.t(D)
• end(O1, S1, D) \triangleq 01.x(D) + S1.t(D) + S1.1(D)
• disjoint_sboxes(Dims, O1, S1, O2, S2) \triangleq \\
\exists D \in Dims \\
\quad \bigvee \\
\quad \begin{pmatrix} \\
\quad \text{origin}(O1, S1, D) > \\
\quad \text{end}(O2, S2, D) \\
\quad \text{origin}(O2, S2, D) > \\
\quad \text{end}(O1, S1, D) \\
\end{pmatrix} \\
• disjoint_objects(Dims, O1, O2) \triangleq \\
\forall S1 \in \text{sboxes}(O1.sid) \\
\forall S2 \in \text{sboxes}(O2.sid) \\
\text{disjoint_sboxes}(Dims, O1, S1, O2, S2) \\
• all_disjoint(Ddims, OIDS) \triangleq \\
\forall O1 \in \text{objects}(OIDS) \\
\forall O2 \in \text{objects}(OIDS) \\
\quad (O1.\text{oid} < \Rightarrow \\
\quad O2.\text{oid}) \\
\quad \text{disjoint_objects}(Ddims, O1, O2) \\
• all_disjoint(DIMENSIONS, OIDS)
## 5.113 disjoint_tasks

### DESCRIPTION

Derived from `disjoint`.

### LINKS

- disjoint_tasks(TASKS1, TASKS2)

### GRAPH

- TASKS1: collection(origin=dvar, duration=dvar, end=dvar)
- TASKS2: collection(origin=dvar, duration=dvar, end=dvar)

### Restrictions

- `require_at_least(2, TASKS1, [origin, duration, end])`
- `TASKS1.duration ≥ 0`
- `TASKS1.origin ≤ TASKS1.end`
- `require_at_least(2, TASKS2, [origin, duration, end])`
- `TASKS2.duration ≥ 0`
- `TASKS2.origin ≤ TASKS2.end`

### Purpose

Each task of the collection TASKS1 should not overlap any task of the collection TASKS2. Two tasks overlap if they have an intersection that is strictly greater than zero.

### Example

```
\begin{pmatrix}
\langle \text{origin}−6, \text{duration}−5, \text{end}−11, \\
\text{origin}−8, \text{duration}−2, \text{end}−10, \\
\text{origin}−2, \text{duration}−2, \text{end}−4, \\
\text{origin}−3, \text{duration}−3, \text{end}−6, \\
\text{origin}−12, \text{duration}−1, \text{end}−13
\end{pmatrix}
```

Figure 5.223 displays the two groups of tasks (i.e., the tasks of TASKS1 and the tasks of TASKS2). Since no task of the first group overlaps any task of the second group, the disjoint_tasks constraint holds.

### Typical

- `|TASKS1| > 1`
- `TASKS1.duration > 0`
- `|TASKS2| > 1`
- `TASKS2.duration > 0`
Symmetries

- Arguments are permutable w.r.t. permutation (TASKS1, TASKS2).
- Items of TASKS1 are permutable.
- Items of TASKS2 are permutable.
- One and the same constant can be added to the origin and end attributes of all items of TASKS1 and TASKS2.

Arg. properties

- Contractible wrt. TASKS1.
- Contractible wrt. TASKS2.

Remark

Despite the fact that this is not an uncommon constraint, it cannot be modelled in a compact way with one single cumulative constraint. But it can be expressed by using the coloured_cumulative constraint: We assign a first colour to the tasks of TASKS1 as well as a second distinct colour to the tasks of TASKS2. Finally we set up a limit of 1 for the maximum number of distinct colours allowed at each time point.

Reformulation

The disjoint_tasks constraint can be expressed in term of |TASKS1| · |TASKS2| reified constraints. For each task TASKS1[i] (i ∈ [1, |TASKS1|]) and for each task TASKS2[j] (j ∈ [1, |TASKS2|]) we generate a reified constraint of the form TASKS1[i].end ≤ TASKS2[j].origin ∨ TASKS2[j].end ≤ TASKS1[i].origin. In addition we also state for each task an arithmetic constraint that states that the end of a task is equal to the sum of its origin and its duration.

Systems

disjoint in Choco.

See also

generalisation: coloured_cumulative(tasks colours and limit on maximum number of colours in parallel are explicitly given).
specialisation: disjoint(task replaced by variable).

Keywords

constraint type: scheduling constraint, temporal constraint.
geometry: non-overlapping.
**Graph model**

PRODUCT is used in order to generate the arcs of the graph between all the tasks of the collection TASKS1 and all tasks of the collection TASKS2. The first two graph constraints respectively enforce for each task of TASKS1 and TASKS2 the fact that the end of a task is equal to the sum of its origin and its duration. The arc constraint of the third graph constraint depicts the fact that two tasks overlap. Therefore, since we use the graph property $\text{NARC} = 0$ the final graph associated with the third graph constraint will be empty and no task of TASKS1 will overlap any task of TASKS2. Figure 5.224 shows the initial graph of the third graph constraint associated with the Example slot. Because of the graph property $\text{NARC} = 0$ the corresponding final graph is empty.

**Signature**

Since TASKS1 is the maximum number of arcs of the final graph associated with the first graph constraint we can rewrite $\text{NARC} = |\text{TASKS1}|$. This leads to simplify $\text{NARC}$ to $\text{NARC}$.

We can apply a similar remark for the second graph constraint.

Finally, since 0 is the smallest number of arcs of the final graph we can rewrite $\text{NARC} = 0$ to $\text{NARC} \leq 0$. This leads to simplify $\text{NARC}$ to $\text{NARC}$. 
Figure 5.224: Initial graph of the disjoint tasks constraint (the final graph is empty)
### 5.114 disjunctive

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[86]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>disjunctive(TASKS)</td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td>one_machine.</td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>TASKS : collection(origin−dvar,duration−dvar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(TASKS,[origin,duration])&lt;br&gt;TASKS.duration ≥ 0</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>All the tasks of the collection TASKS that have a duration strictly greater than 0 should not overlap.</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>Figure 5.225 shows the tasks with non-zero duration of the example. Since these tasks do not overlap the disjunctive constraint holds.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.225: Tasks with non-zero duration

<table>
<thead>
<tr>
<th>Typical</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symmetries</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arg. properties</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Remark</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Some systems like Ilog CP Optimizer also imposes that zero duration tasks do not overlap non-zero duration tasks.
A soft version of this constraint, under the hypothesis that all durations are fixed, was presented by P. Baptiste et al. in [17]. In this context the goal was to perform as many tasks as possible within their respective due-dates.

When all tasks have the same (fixed) duration the disjunctive constraint can be reformulated as an \texttt{all\_min\_dist} constraint for which a filtering algorithm achieving bound-consistency is available [10].

Within the context of linear programming [198, page 386] provides several relaxations of the disjunctive constraint.

Some solvers use in a pre-processing phase, while stating precedence and cumulative constraints, \textit{an algorithm for automatically extracting large cliques} [83] from a set of tasks that should not pairwise overlap (i.e., two tasks \( t_i \) and \( t_j \) can not overlap either, because \( t_i \) ends before the start of \( t_j \), either because the sum of resource consumption of \( t_i \) and \( t_j \) exceeds the capacity of a cumulative resource that both tasks use) in order to state disjunctive constraints.

\textbf{Algorithm}

We have four main families of methods for handling the disjunctive constraint:

- **Methods based on the compulsory part** [232] of the tasks (also called time-tabling methods). These methods determine the time slots which for sure are occupied by a given task, and propagate back this information to the attributes of each task (i.e., the origin and the duration). Because of their simplicity, these methods have been originally used for handling the disjunctive constraint. Even if they propagate less than the other methods they can in practice handle a large number of tasks. To our best knowledge no efficient incremental algorithm devoted to this problem was published up to now (i.e., September 2006).

- **Methods based on constructive disjunction.** The idea is to try out each alternative of a disjunction (e.g., given two tasks \( t_1 \) and \( t_2 \) that should not overlap, we successively assume that \( t_1 \) finishes before \( t_2 \), and that \( t_2 \) finishes before \( t_1 \)) and to remove values that were pruned in both alternatives.

- **Methods based on edge-finding.** Given a set of tasks \( T \), edge-finding determines that some task must, can, or cannot execute first or last in \( T \). Efficient edge-finding algorithms for handling the disjunctive constraint were originally described in [87, 88] and more recently in [407, 284].

- **Methods that, for any task \( t \), consider the maximal number of tasks that can end up before the start of task \( t \) as well as the maximal number of tasks that can start after the end of task \( t \) [416].**

All these methods are usually used for adjusting the minimum and maximum values of the variables of the disjunctive constraint. However some systems use these methods for pruning the full domain of the variables. Finally, \textit{Jackson priority rule} [207] provides a necessary condition [88] for the disjunctive constraint. Given a set of tasks \( T \), it consists to progressively schedule all tasks of \( T \) in the following way:

- **It assigns to the first possible time point (i.e., the earliest start of all tasks of \( T \)) the available task with minimal latest end.** In this context, available means a task for which the earliest start is less than or equal to the considered time point.

- **It continues by considering the next time point until all the tasks are completely scheduled.**
Systems

- disjunctive in **Choco**, unary in **Gecode**.

See also

- common keyword: calendar, disj, disjunctive_or_same_end, disjunctive_or_same_start (scheduling constraint).
- generalisation: cumulative (task heights and resource limit are not necessarily all equal to 1), diffn (task of heigth 1 replaced by orthotope).
- implied by: precedence.
- implies: disjunctive_or_same_end, disjunctive_or_same_start.
- specialisation: all_min_dist (line segment replaced by line segment, of same length), alldifferent (task replaced by variable).

Keywords

- characteristic of a constraint: core, sort based reformulation.
- complexity: sequencing with release times and deadlines.
- constraint type: scheduling constraint, resource constraint, decomposition.
- filtering: compulsory part, constructive disjunction, Phi-tree.
- modelling: disjunction, sequence dependent set-up, zero-duration task.
- modelling exercises: sequence dependent set-up.
- problems: maximum clique.
Arc input(s)  

Arc generator  

Arc arity  

Arc constraint(s)  

Graph property(ies)  

Graph model  

We generate a *clique* with a non-overlapping constraint between each pair of distinct tasks and state that the number of arcs of the final graph should be equal to the number of arcs of the initial graph.

Parts (A) and (B) of Figure 5.226 respectively show the initial and final graph associated with the Example slot. The disjunctive constraint holds since all the arcs of the initial graph belong to the final graph: all the non-overlapping constraints holds.
### 5.115 **disjunctive_or_same_end**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Scheduling.</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>disjunctive_or_same_end(TASKS)</td>
<td></td>
</tr>
<tr>
<td><strong>Synonyms</strong></td>
<td>same_end_or_disjunctive, same_end_or_non_overlap.</td>
<td>non_overlap_or_same_end.</td>
</tr>
<tr>
<td><strong>Argument</strong></td>
<td>TASKS : $\text{collection}(\text{origin-dvar}, \text{duration-dvar})$</td>
<td></td>
</tr>
</tbody>
</table>
| **Restrictions** | $\text{required}((\text{TASKS}, [\text{origin, duration}]))$
$\text{TASKS.duratio}n \geq 0$ |       |

**Purpose**

All pairs of tasks of the collection TASKS that have a duration strictly greater than 0 should either not overlap either have the same end, i.e. $\forall i \in [1, |\text{TASKS}|], \forall j \in [i + 1, |\text{TASKS}|] :$ 

\[
\text{TASKS}[i].\text{duration} = 0 \lor \text{TASKS}[j].\text{duration} = 0 \lor \text{TASKS}[i].\text{origin} + \text{TASKS}[i].\text{duration} \leq \text{TASKS}[j].\text{origin} \lor \text{TASKS}[j].\text{origin} + \text{TASKS}[j].\text{duration} \leq \text{TASKS}[i].\text{origin} \lor \text{TASKS}[i].\text{origin} + \text{TASKS}[i].\text{duration} = \text{TASKS}[j].\text{origin} + \text{TASKS}[j].\text{duration}.
\]

**Example**

\[
\begin{pmatrix}
\text{origin - 4} & \text{duration - 3}, \\
\text{origin - 7} & \text{duration - 2}, \\
\text{origin - 5} & \text{duration - 2}
\end{pmatrix}
\]

Since the ends of the first and third tasks coincide, and since the second task does neither overlap the first task nor the third task, the **disjunctive_or_same_end** constraint holds.

**Typical**

$|\text{TASKS}| > 1$
$\text{TASKS.duratio}n \geq 1$

**Symmetries**

- Items of TASKS are permutable.
- TASKS.duration can be decreased to any value $\geq 0$.
- One and the same constant can be added to the origin attribute of all items of TASKS.

**Arg. properties**

Contractible wrt. TASKS.

**See also**

common keyword: **disjunctive, disjunctive_or_same_start** *(scheduling constraint)*.

implied by: **disjunctive**.

**Keywords**

constraint type: scheduling constraint, resource constraint, decomposition.

modelling: disjunction, zero-duration task.
Arc input(s)  
**TASKS**

Arc generator  
*CLIQUEx*→*collection*(tasks1, tasks2)

Arc arity  
2

Arc constraint(s)  
\[
\left\{ \begin{array}{l}
tasks1\text{.duration} = 0, \\
tasks2\text{.duration} = 0, \\
tasks1\text{.origin} + tasks1\text{.duration} \leq tasks2\text{.origin}, \\
tasks2\text{.origin} + tasks2\text{.duration} \leq tasks1\text{.origin}, \\
tasks1\text{.origin} + tasks1\text{.duration} = tasks2\text{.origin} + tasks2\text{.duration} \\
\end{array} \right. 
\]

Graph property(ies)  

\[NARC = |\text{TASKS}| \ast (|\text{TASKS}| - 1) / 2\]

Graph model  
We generate a *clique* with a non-overlapping constraint or a same end constraint between each pair of distinct tasks and state that the number of arcs of the final graph should be equal to the number of arcs of the initial graph.

Parts (A) and (B) of Figure 5.227 respectively show the initial and final graph associated with the **Example** slot. The *disjunctive_or_same_end* constraint holds since all the arcs of the initial graph belong to the final graph.

![Diagram of initial and final graph](image)

Figure 5.227: Initial and final graph of the *disjunctive_or_same_end* constraint
### 5.116 disjunctive_or_same_start

<table>
<thead>
<tr>
<th>Origin</th>
<th>Scheduling.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint</td>
<td>disjunctive_or_same_start(TASKS)</td>
</tr>
<tr>
<td>Synonyms</td>
<td>same_start_or_disjunctive, non_overlap_or_same_start, same_start_or_non_overlap</td>
</tr>
<tr>
<td>Argument</td>
<td>TASKS : collection(origin−dvar, duration−dvar)</td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(TASKS,[origin,duration])</td>
</tr>
<tr>
<td>Purpose</td>
<td>TASKS.duration ≥ 0</td>
</tr>
<tr>
<td></td>
<td>All pairs of tasks of the collection TASKS that have a duration strictly greater than 0 should either not overlap or have the same start, i.e. ∀i ∈ [1,</td>
</tr>
</tbody>
</table>
| Example | \[
\begin{pmatrix}
\text{origin} - 4 & \text{duration} - 3, \\
\text{origin} - 7 & \text{duration} - 2, \\
\text{origin} - 4 & \text{duration} - 1
\end{pmatrix}
\] |
| Purpose | Since the starts of the first and third tasks coincide, and since the second task does neither overlap the first task nor the third task, the disjunctive_or_same_start constraint holds. |
| Typical | | |
| | | |
| | | |
| | | |
| | | |
| Symmetries | • Items of TASKS are permutable. |
| | • TASKS.duration can be decreased to any value ≥ 0. |
| | • One and the same constant can be added to the origin attribute of all items of TASKS. |
| Arg. properties | Contractible wrt. TASKS. |
| See also | common keyword: disjunctive, disjunctive_or_same_end (scheduling constraint). implied by: disjunctive. |
| Keywords | constraint type: scheduling constraint, resource constraint, decomposition. modelling: disjunction, zero-duration task. |
Arc input(s): TASKS
Arc generator: $CLIQUE(<) \rightarrow \text{collection}(\text{tasks1, tasks2})$
Arc arity: 2
Arc constraint(s):

\[
\begin{align*}
\text{tasks1.duration} &= 0, \\
\text{tasks2.duration} &= 0, \\
\text{tasks1.origin} + \text{tasks1.duration} &\leq \text{tasks2.origin}, \\
\text{tasks2.origin} + \text{tasks2.duration} &\leq \text{tasks1.origin}, \\
\text{tasks1.origin} &= \text{tasks2.origin}
\end{align*}
\]

Graph property(ies): \[\text{NARC} = |\text{TASKS}| \ast (|\text{TASKS}| - 1)/2\]

Graph model:
We generate a *clique* with a non-overlapping constraint or a same start constraint between each pair of distinct tasks and state that the number of arcs of the final graph should be equal to the number of arcs of the initial graph.

Parts (A) and (B) of Figure 5.228 respectively show the initial and final graph associated with the Example slot. The disjunctive_or_same_start constraint holds since all the arcs of the initial graph belong to the final graph.

![Figure 5.228: Initial and final graph of the disjunctive_or_same_start constraint](image)
### 5.117 distance

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Arithmetic constraint.</td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>distance($X, Y, Z$)</td>
</tr>
</tbody>
</table>
| **Arguments** | $X$ : dvar  
$Y$ : dvar  
$Z$ : dvar |
| **Restriction** | $Z \geq 0$ |
| **Purpose** | Enforce the fact that $Z$ is equal to $|X - Y|$. |
| **Example** | $(5, 7, 2)$  
The distance constraint holds since $2 = |5 - 7|$. |
| **Typical** | $Z > 0$ |
| **Symmetry** | Arguments are permutable w.r.t. permutation $(X, Y) (Z)$. |
| **Arg. properties** | Functional dependency: $Z$ determined by $X$ and $Y$. |
| **Systems** | distanceEQ in Choco, distance in JaCoP, distance2 in JaCoP. |
| **See also** | implies: $leq$ cst.  
related: all_min_dist (fixed minimum distance between all pairs of variables of a collection of variables), smooth. |
| **Keywords** | constraint arguments: ternary constraint, pure functional dependency.  
constraint type: arithmetic constraint, predefined constraint.  
modelling: functional dependency. |
5.118 distance_between

**Origin**
N. Beldiceanu

**Constraint**
distance_between(DIST, VARIABLES1, VARIABLES2, CTR)

**Synonym**
distance.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIST</td>
<td>dvar</td>
</tr>
<tr>
<td>VARIABLES1</td>
<td>collection(var−dvar)</td>
</tr>
<tr>
<td>VARIABLES2</td>
<td>collection(var−dvar)</td>
</tr>
<tr>
<td>CTR</td>
<td>atom</td>
</tr>
</tbody>
</table>

**Restrictions**

- DIST ≥ 0
- DIST ≤ |VARIABLES1| * |VARIABLES2| − |VARIABLES1|
- required(VARIABLES1, var)
- required(VARIABLES2, var)
- |VARIABLES1| = |VARIABLES2|
- CTR ∈ [=, ≠, <, ≥, >, ≤]

**Purpose**
Let $U_i$ and $V_i$ be respectively the $i^{th}$ and $j^{th}$ variables ($i \neq j$) of the collection VARIABLES1. In a similar way, let $X_i$ and $Y_i$ be respectively the $i^{th}$ and $j^{th}$ variables ($i \neq j$) of the collection VARIABLES2. DIST is equal to the number of times one of the following mutually incompatible conditions are true:

- $U_i$ CTR $V_i$ holds and $X_i$ CTR $Y_i$ does not hold,
- $X_i$ CTR $Y_i$ holds and $U_i$ CTR $V_i$ does not hold.

**Example**

$$\left(2, \langle3, 4, 6, 2, 4\rangle, \langle2, 6, 9, 3, 6\rangle, <\right)$$

The distance_between constraint holds since the following DIST = 2 conditions are verified:


**Typical**

<table>
<thead>
<tr>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIST &gt; 0</td>
</tr>
<tr>
<td>DIST &lt;</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CTR ∈ [=, ≠]</td>
</tr>
</tbody>
</table>
Symmetries

- Arguments are permutable w.r.t. permutation (DIST) (VARIABLES1, VARIABLES2) (CTR).
- Items of VARIABLES1 and VARIABLES2 are permutable (same permutation used).
- One and the same constant can be added to the var attribute of all items of VARIABLES1.
- One and the same constant can be added to the var attribute of all items of VARIABLES2.

Arg. properties

Functional dependency: DIST determined by VARIABLES1, VARIABLES2 and CTR.

Usage

Measure the distance between two sequences in term of the number of constraint changes. This should be put in contrast to the number of value changes that is sometimes superficial.

See also

common keyword: distance_change (proximity constraint).

Keywords

constraint arguments: pure functional dependency.
constraint type: proximity constraint.
modelling: functional dependency.
<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES1/ VARIABLES2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>\textit{CLIQUE(\neq)} \rightarrow \text{collection}(\text{variables1}, \text{variables2})</td>
</tr>
<tr>
<td>Arc arity</td>
<td>2</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>\text{variables1}.\text{var} CTR \text{variables2}.\text{var}</td>
</tr>
<tr>
<td>Graph property(ies)</td>
<td>\text{DISTANCE} = \text{DIST}</td>
</tr>
</tbody>
</table>

**Graph model**

Within the Arc input(s) slot, the character / indicates that we generate two distinct graphs. The graph property \text{DISTANCE} measures the distance between two digraphs $G_1$ and $G_2$. This distance is defined as the sum of the following quantities:

- The number of arcs of $G_1$ that do not belong to $G_2$,
- The number of arcs of $G_2$ that do not belong to $G_1$.

Part (A) of Figure 5.229 gives the final graph associated with the sequence var-3,var-4,var-6,var-2,var-4 (i.e., the second argument of the constraint of the Example slot), while part (B) shows the final graph corresponding to var-2,var-6,var-9,var-3,var-6 (i.e., the third argument of the constraint of the Example slot). The two arc constraints that differ from one graph to the other are marked by a dotted line. The distance between constraint holds since between sequence var-3,var-4,var-6,var-2,var-4 and sequence var-2,var-6,var-9,var-3,var-6 there are \text{DIST} = 2 changes that respectively correspond to:

- Within the final graph associated with sequence var-3,var-4,var-6,var-2,var-4 the arc 4 \rightarrow 1 (i.e., values 2 \rightarrow 3) does not occur in the final graph associated with var-2,var-6,var-9,var-3,var-6,
- Within the final graph associated with sequence var-2,var-6,var-9,var-3,var-6 the arc 1 \rightarrow 4 (i.e., values 2 \rightarrow 3) does not occur in the final graph associated with var-3,var-4,var-6,var-2,var-4.

![Final graphs of the distance between constraint](image)

Figure 5.229: Final graphs of the distance between constraint
5.119  distance_change

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from change.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>distance_change(DIST, VARIABLES1, VARIABLES2, CTR)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td>distance.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>DIST: dvar, VARIABLES1: collection(var-dvar), VARIABLES2: collection(var-dvar), CTR: atom</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Restrictions| DIST $\geq 0$
DIST $< |VARIABLES1|
required(VARIABLES1, var)
required(VARIABLES2, var)
|VARIABLES1| = |VARIABLES2|
CTR $\in [=, \neq, <, \geq, >, \leq]$ |
| Purpose     | DIST is equal to the number of times one of the following two conditions is true ($1 \leq i < n$):
|             | • VARIABLES1[i].var CTR VARIABLES1[i + 1].var holds and VARIABLES2[i].var CTR VARIABLES2[i + 1].var does not hold,
|             | • VARIABLES2[i].var CTR VARIABLES2[i + 1].var holds and VARIABLES1[i].var CTR VARIABLES1[i + 1].var does not hold. |
| Example     | (1, (3, 3, 1, 2, 2),
{4, 4, 3, 3, 3}, \neq) |
|             | The distance_change constraint holds since the following condition (DIST = 1)
is verified: 
|             | \{ VARIABLES1[3].var = 1 \neq VARIABLES1[4].var = 2 \land
VARIABLES2[3].var = 3 = VARIABLES1[4].var = 3 \}. |
| Typical     | DIST $> 0$
|             | |VARIABLES1| $> 1$
|             | CTR $\in [=, \neq]$ |
| Symmetries  | • Arguments are permutable w.r.t. permutation (DIST)
(VARIABLES1, VARIABLES2) (CTR).
|             | • One and the same constant can be added to the var attribute of all items of VARIABLES1.
|             | • One and the same constant can be added to the var attribute of all items of VARIABLES2.
Functional dependency: DIST determined by VARIABLES1, VARIABLES2 and CTR.

Usage
Measure the distance between two sequences according to the change constraint.

Remark
We measure that distance with respect to a given constraint and not according to the fact that the variables are assigned distinct values.

See also
- common keyword: distance_between (proximity constraint).
- root concept: change.

Keywords
- characteristic of a constraint: automaton, automaton with counters.
- constraint arguments: pure functional dependency.
- constraint network structure: sliding cyclic(2) constraint network(2).
- constraint type: proximity constraint.
**Arc input(s)**
VARIABLES1/ VARIABLES2

**Arc generator**
PATH↦collection(variables1,variables2)

**Arc arity**
2

**Arc constraint(s)**
variables1.var CTR variables2.var

**Graph property(ies)**
DISTANCE= DIST

**Graph model**
Within the Arc input(s) slot, the character / indicates that we generate two distinct graphs. The graph property DISTANCE measures the distance between two digraphs $G_1$ and $G_2$. This distance is defined as the sum of the following quantities:

- The number of arcs of $G_1$ that do not belong to $G_2$,
- The number of arcs of $G_2$ that do not belong to $G_1$.

Part (A) of Figure 5.230 gives the final graph associated with the sequence var-3,var-3,var-1,var-2 (i.e., the second argument of the constraint of the Example slot), while part (B) shows the final graph corresponding to var-4,var-4,var-3,var-3 (i.e., the third argument of the constraint of the Example slot). Since arc 3 → 4 belongs to the first final graph but not to the second one, the distance between the two final graphs is equal to 1.

Figure 5.230: Final graphs of the distance change constraint
Automaton

Figure 5.231 depicts the automaton associated with the distance_change constraint. Let \((VAR_1, \text{VAR}_1_{i+1})\) and \((VAR_2, \text{VAR}_2_{i+1})\) respectively be the \(i\)'th pairs of consecutive variables of the collections \(\text{VARIABLES}_1\) and \(\text{VARIABLES}_2\). To each quadruple \((VAR_1, \text{VAR}_1_{i+1}, \text{VAR}_2, \text{VAR}_2_{i+1})\) corresponds a 0-1 signature variable \(S_i\). The following signature constraint links these variables:

\[
((VAR_1 = \text{VAR}_1_{i+1}) \land (VAR_2 \neq \text{VAR}_2_{i+1})) \lor \\
((VAR_1 \neq \text{VAR}_1_{i+1}) \land (VAR_2 = \text{VAR}_2_{i+1})) \iff S_i.
\]

\(s_0: \{C = 0\}
\)
\((\text{VAR}_1 \text{ CT} \text{R} \text{VAR}_1_{i+1} \text{ and } \text{VAR}_2 \text{ not CT} \text{R} \text{VAR}_2_{i+1}) \text{ or } \text{and } \text{(VAR}_1 \text{ not CT} \text{R} \text{VAR}_1_{i+1} \text{ and } \text{VAR}_2 \text{ CT} \text{R} \text{VAR}_2_{i+1})\) and

\(s_1: \{C = 1\}
\)

\(Q_0 = s \quad Q_1 \quad Q_2 \quad Q_{n-1} = s \quad Q_n = C = \text{DIST} \quad \text{DIST}\)

Figure 5.232: Hypergraph of the reformulation corresponding to the automaton of the distance_change constraint.
## 5.120 divisible

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Arithmetic.</td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>divisible(Q, D)</td>
</tr>
<tr>
<td><strong>Synonym</strong></td>
<td>div.</td>
</tr>
</tbody>
</table>
| **Arguments** | \( Q : \text{dvar} \)  
\( D : \text{dvar} \) |
| **Restrictions** | \( Q \geq 0 \)  
\( D > 0 \) |
| **Purpose** | Enforce the fact that the first variable \( Q \) is divisible by the second variable \( D \). |
| **Example** | \((12, 4)\) |
| The divisible constraint holds since 12 is divisible by 4. |
| **Typical** | \( Q > 1 \)  
\( D < Q \) |
| **See also** | implies: divisible_or_same_sign. |
| **Keywords** | constraint arguments: binary constraint.  
constraint type: predefined constraint, arithmetic constraint.  
filtering: arc-consistency. |
### 5.121 divisible_or

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Arithmetic.</td>
</tr>
<tr>
<td>Constraint</td>
<td>divisible_or(C, D)</td>
</tr>
<tr>
<td>Synonym</td>
<td>div_or.</td>
</tr>
</tbody>
</table>
| Arguments   | C : dvar  
D : dvar |
| Restrictions| C > 0  
D > 0 |

**Purpose**

Enforce the fact that the first variable C is divisible by the second variable D, or that D is divisible by C.

**Example**

(4, 12)

The divisible_or constraint holds since 12 is divisible by 4.

**See also**

implied by: divisible.

**Keywords**

constraint arguments: binary constraint.  
constraint type: predefined constraint, arithmetic constraint.
### 5.122  \texttt{dom\_reachability}

**Description**

**Links**

**Origin** [310]

**Constraint**

\[
\text{dom\_reachability}(\text{SOURCE}, \text{FLOW\_GRAPH}, \text{DOMINATOR\_GRAPH}, \text{TRANSITIVE\_CLOSURE\_GRAPH})
\]

**Arguments**

- **SOURCE**: \texttt{int}
- **FLOW\_GRAPH**: \texttt{collection(index-int, succ-svar)}
- **DOMINATOR\_GRAPH**: \texttt{collection(index-int, succ-sint)}
- **TRANSITIVE\_CLOSURE\_GRAPH**: \texttt{collection(index-int, succ-svar)}

**Restrictions**

\[
\begin{align*}
\text{SOURCE} & \geq 1 \\
\text{SOURCE} & \leq |\text{FLOW\_GRAPH}| \\
\text{required}(\text{FLOW\_GRAPH}[[\text{index}, \text{succ}]) \\
\text{FLOW\_GRAPH}.\text{index} & \geq 1 \\
\text{FLOW\_GRAPH}.\text{index} & \leq |\text{FLOW\_GRAPH}| \\
\text{FLOW\_GRAPH}.\text{succ} & \geq 1 \\
\text{FLOW\_GRAPH}.\text{succ} & \leq |\text{FLOW\_GRAPH}| \\
\text{distinct}(\text{FLOW\_GRAPH}, \text{index}) \\
\text{required}(\text{DOMINATOR\_GRAPH}[[\text{index}, \text{succ}]) \\
|\text{DOMINATOR\_GRAPH}| & = |\text{FLOW\_GRAPH}| \\
\text{DOMINATOR\_GRAPH}.\text{index} & \geq 1 \\
\text{DOMINATOR\_GRAPH}.\text{index} & \leq |\text{DOMINATOR\_GRAPH}| \\
\text{DOMINATOR\_GRAPH}.\text{succ} & \geq 1 \\
\text{DOMINATOR\_GRAPH}.\text{succ} & \leq |\text{DOMINATOR\_GRAPH}| \\
\text{distinct}(\text{DOMINATOR\_GRAPH}, \text{index}) \\
\text{required}(\text{TRANSITIVE\_CLOSURE\_GRAPH}[[\text{index}, \text{succ}]) \\
|\text{TRANSITIVE\_CLOSURE\_GRAPH}| & = |\text{FLOW\_GRAPH}| \\
\text{TRANSITIVE\_CLOSURE\_GRAPH}.\text{index} & \geq 1 \\
\text{TRANSITIVE\_CLOSURE\_GRAPH}.\text{index} & \leq |\text{TRANSITIVE\_CLOSURE\_GRAPH}| \\
\text{TRANSITIVE\_CLOSURE\_GRAPH}.\text{succ} & \geq 1 \\
\text{TRANSITIVE\_CLOSURE\_GRAPH}.\text{succ} & \leq |\text{TRANSITIVE\_CLOSURE\_GRAPH}| \\
\text{distinct}(\text{TRANSITIVE\_CLOSURE\_GRAPH}, \text{index})
\end{align*}
\]
Let \texttt{FLOW\_GRAPH}, \texttt{DOMINATOR\_GRAPH} and \texttt{TRANSITIVE\_CLOSURE\_GRAPH} be three directed graphs respectively called the \textit{flow graph}, the \textit{dominance graph} and the \textit{transitive closure graph} which all have the same vertices. In addition let \texttt{SOURCE} denote a vertex of the flow graph called the \textit{source node} (not necessarily a vertex with no incoming arcs). The \texttt{dom\_reachability} constraint holds if and only if the flow graph (and its source node) verifies:

- The dominance relation expressed by the dominance graph (i.e., if there is an arc \((i, j)\) in the dominance graph then, within the flow graph, all the paths from the source node to \(j\) contain \(i\); note that when there is no path from the source node to \(j\) then any node dominates \(j\)).
- The transitive relation expressed by the transitive closure graph (i.e., if there is an arc \((i, j)\) in the transitive closure graph then there is also a path from \(i\) to \(j\) in the flow graph).

\begin{verbatim}
index - 1 succ = {2},
1, index - 2 succ = {3, 4},
index - 3 succ = {},
index - 4 succ = {}.
index - 1 succ = {2, 3, 4},
index - 2 succ = {3, 4},
index - 3 succ = {},
index - 4 succ = {}.
index - 1 succ = {1, 2, 3, 4},
index - 2 succ = {2, 3, 4},
index - 3 succ = {3},
index - 4 succ = {4}.
\end{verbatim}

Example

The flow graph, the dominance graph and the transitive closure graph corresponding to the second, third and fourth arguments of the \texttt{dom\_reachability} constraint are respectively depicted by parts (A), (B) and (C) of Figure 5.233. The \texttt{dom\_reachability} holds since the following conditions hold.

- The dominance relation expressed by the dominance graph is verified:
  - Since \((1, 2)\) belongs to the dominance graph all the paths from 1 to 2 in the flow graph pass through 1.
  - Since \((1, 3)\) belongs to the dominance graph all the paths from 1 to 3 in the flow graph pass through 1.
  - Since \((1, 4)\) belongs to the dominance graph all the paths from 1 to 4 in the flow graph pass through 1.
  - Since \((2, 3)\) belongs to the dominance graph all the paths from 1 to 3 in the flow graph pass through 2.
  - Since \((2, 4)\) belongs to the dominance graph all the paths from 1 to 4 in the flow graph pass through 2.

- The graph depicted by the fourth argument of the \texttt{dom\_reachability} constraint (i.e., \texttt{TRANSITIVE\_CLOSURE\_GRAPH}) is the transitive closure of the graph depicted by the second argument (i.e., \texttt{FLOW\_GRAPH}).
Typical $|\text{FLOW\_GRAPH}| > 2$

Symmetries
- Items of $\text{FLOW\_GRAPH}$ are permutable.
- Items of $\text{DOMINATOR\_GRAPH}$ are permutable.
- Items of $\text{TRANSITIVE\_CLOSURE\_GRAPH}$ are permutable.

Usage
The $\text{dom\_reachability}$ constraint was introduced in order to solve reachability problems (e.g., disjoint paths, simple path with mandatory nodes).

Remark
Within the name $\text{dom\_reachability}$, dom stands for domination. In the context of path problems $\text{SOURCE}$ refers to the start of the path we want to build.

Algorithm
It was shown in [308] that, finding out whether a $\text{dom\_reachability}$ constraint has a solution or not is NP-hard. This was achieved by reduction to disjoint paths problem [169].

The first implementation [309] of the $\text{dom\_reachability}$ constraint was done in Mozart [114]. Later on, a second implementation [308] was done in Gecode [353]. Both implementations consist of the following two parts:
- Algorithms [342] for maintaining the lower bound of the transitive closure graph.
- Algorithms for maintaining the upper bound of the transitive closure graph, while respecting the dominance constraints [177].

See also
common keyword: path, path_from_to ($\text{path}$).

Keywords
combinatorial object: path.
constraint arguments: constraint involving set variables.
constraint type: predefined constraint, graph constraint.

Figure 5.233: (A) Flow graph, (B) dominance graph and (C) transitive closure graph of the Example slot (taken from [308, page 40])
5.123 domain

**DESCRIPTION**

Domain definition.

**LINKS**

- **Constraint**: `domain(VARIABLES, LOW, UP)`
- **Synonym**: `dom`
- **Arguments**:
  - `VARIABLES`: `collection(var−dvar)`
  - `LOW`: `int`
  - `UP`: `int`
- **Restrictions**:
  - `required(VARIABLES, var)`
  - `LOW ≤ UP`
- **Purpose**:
  - Enforce all the variables of the collection `VARIABLES` to take a value within the interval `[LOW, UP]`.
- **Example**:
  - `((2, 8, 2), 1, 9)`

The domain constraint holds since all the values 2, 8 and 2 of its first argument are greater than or equal to its second argument `LOW = 1` and less than or equal to its third argument `UP = 9`.

**Typical**:

- `|VARIABLES| > 1`
- `LOW < UP`

**Symmetries**:

- Items of `VARIABLES` are **permutable**.
- An occurrence of a value of `VARIABLES.var` can be **replaced** by any other value in `[LOW, UP]`.
- `LOW` can be **decreased**.
- `UP` can be **increased**.
- One and the same constant can be **added** to the `var` attribute of all items of `VARIABLES` as well as to `LOW` and `UP`.

**Arg. properties**:

- **Contractible** wrt. `VARIABLES`.

**Remark**:

The domain constraint is called `dom` in **Gecode** (http://www.gecode.org/).

**Reformulation**:

The domain `((var − V_1, var − V_2, ..., var − V_{|VARIABLES|}), LOW, UP)` constraint can be expressed in term of the conjunction:

- `V_1 ≥ LOW ∧ V_1 ≤ UP`.
- `V_2 ≥ LOW ∧ V_2 ≤ UP`.
- `...`
- `V_{|VARIABLES|} ≥ LOW ∧ V_{|VARIABLES|} ≤ UP`. 
Systems

member in Choco, dom in Gecode, domain in SICStus.

See also

common keyword: in, in_interval (domain definition).
uses in its reformulation: tree_range.

Keywords

constraint type: predefined constraint, value constraint.
modelling: interval, domain definition.
## 5.124 domain_constraint

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>[319]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>domain_constraint(VAR, VALUES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Synonym</strong></td>
<td>domain.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>VAR : dvar</td>
<td>VALUES : collection(var01−dvar, value−int)</td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>required(VALUES, [var01, value])</td>
<td>VALUES.var01 ≥ 0</td>
<td>VALUES.var01 ≤ 1</td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>Make the link between a domain variable VAR and those 0-1 variables that are associated with each potential value of VAR: The 0-1 variable associated with the value that is taken by variable VAR is equal to 1, while the remaining 0-1 variables are all equal to 0.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| **Example** | \[
\begin{pmatrix}
5, \\
\text{var01} - 0, & \text{value} - 9, \\
\text{var01} - 1, & \text{value} - 5, \\
\text{var01} - 0, & \text{value} - 2, \\
\text{var01} - 0, & \text{value} - 7
\end{pmatrix}
\] |

The domain_constraint holds since VAR = 5 is set to the value corresponding to the 0-1 variable set to 1, while the other 0-1 variables are all set to 0.

| **Typical** | \(|VALUES| > 1\) |
| **Symmetry** | Items of VALUES are permutible. |
| **Usage** | This constraint is used in order to make the link between a formulation using finite domain constraints and a formulation exploiting 0-1 variables. |
| **Reformulation** | The domain_constraint(VAR, \(\langle\text{var01} - B_1 \text{ value} - v_1, \text{var01} - B_2 \text{ value} - v_2, \ldots \rangle\)) constraint can be expressed in term of the following reified constraint (VAR = \(v_1 \land B_1 = 1\)) \lor (VAR = \(v_2 \land B_2 = 1\)) \lor \ldots \lor (VAR = v_{|VALUES|} \land B_{|VALUES|} = 1). |
| **Systems** | domainChanneling in Choco, channel in Gecode, in in SICStus, in_set in SICStus. |
See also

common keyword: link_set_to_booleans (channelling constraint).
related: roots.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint, derived collection.
constraint network structure: centered cyclic(1) constraint network(1).
constraint type: decomposition.
filtering: linear programming, arc-consistency.
modelling: channelling constraint, domain channel, Boolean channel.
**Derived Collection**

\[
col \left( \text{VALUE} \rightarrow \text{collection}(\text{var01} - \text{int}, \text{value} - \text{dvar}), \right) \\
[\text{item}(\text{var01} - 1, \text{value} - \text{VAR})]
\]

**Arc input(s)**

VALUE VALUES

**Arc generator**

\[\text{PRODUCT} \mapsto \text{collection}(\text{value}, \text{values})\]

**Arc arity**

2

**Arc constraint(s)**

\[\text{value}.\text{value} = \text{values}.\text{value} \iff \text{values}.\text{var01} = 1\]

**Graph property(ies)**

\[\text{NARC} = \vert \text{VALUES} \vert\]

**Graph model**

The domain constraint constraint is modelled with the following bipartite graph:

- The first class of vertices corresponds to one single vertex containing the domain variable.
- The second class of vertices contains one vertex for each item of the collection VALUES.

\[\text{PRODUCT}\] is used in order to generate the arcs of the graph. In our context it takes a collection with one single item \((\text{var01} - 1 \text{ value} - \text{VAR})\) and the collection VALUES.

The arc constraint between the variable \(\text{VAR}\) and one potential value \(v\) expresses the following:

- If the 0-1 variable associated with \(v\) is equal to 1, \(\text{VAR}\) is equal to \(v\).
- Otherwise, if the 0-1 variable associated with \(v\) is equal to 0, \(\text{VAR}\) is not equal to \(v\).

Since all arc constraints should hold the final graph contains exactly \(|\text{VALUES}|\) arcs.

Parts (A) and (B) of Figure 5.234 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

![Initial and final graph of the domain constraint constraint](image_url)
Since the number of arcs of the initial graph is equal to \texttt{VALUES} the maximum number of arcs of the final graph is also equal to \texttt{VALUES}. Therefore we can rewrite the graph property $\texttt{NARC} = |\texttt{VALUES}|$ to $\texttt{NARC} \geq |\texttt{VALUES}|$. This leads to simplify $\texttt{NARC}$ to $\texttt{NARC}$. 
Automaton

Figure 5.235 depicts the automaton associated with the `domain_constraint` constraint. Let `VAR0_1` and `VALUE_i` respectively be the `var01` and the `value` attributes of the `i`th item of the `VALUES` collection. To each triple `(VAR, VAR0_1, VALUE_i)` corresponds a 0-1 signature variable `S_i` as well as the following signature constraint: `(VAR = VALUE_i) ⇔ VAR0_1 ⇔ S_i`.

![Figure 5.235: Automaton of the `domain_constraint` constraint](image)

Figure 5.235: Automaton of the `domain_constraint` constraint

Figure 5.236: Hypergraph of the reformulation corresponding to the automaton of the `domain_constraint` constraint

![Figure 5.236: Hypergraph of the reformulation](image)
### 5.125 elem

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Derived from element.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>$\text{elem}(\text{ITEM}, \text{TABLE})$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Usual name</strong></td>
<td>element</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Synonyms</strong></td>
<td>$\text{nth}$, $\text{array}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>$\text{ITEM} : \text{collection}(\text{index} - \text{dvar}, \text{value} - \text{dvar})$&lt;br&gt;$\text{TABLE} : \text{collection}(\text{index} - \text{int}, \text{value} - \text{dvar})$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| **Restrictions** | $\text{required}(\text{ITEM}, [\text{index}, \text{value}])$<br>$\text{ITEM.index} \geq 1$<br>$\text{ITEM.index} \leq |\text{TABLE}|$
|             | $|\text{ITEM}| = 1$
|             | $|\text{TABLE}| > 0$
|             | $\text{required}(\text{TABLE}, [\text{index}, \text{value}])$
|             | $\text{TABLE.index} \geq 1$
|             | $\text{TABLE.index} \leq |\text{TABLE}|$
|             | $\text{distinct}(\text{TABLE}, \text{index})$ |       |           |
| **Purpose** | $\text{ITEM}$ is equal to one of the entries of the table $\text{TABLE}$. |       |           |
| **Example** | $\langle \text{index} - 3, \text{value} - 2 \rangle,$
|             | $\langle \text{index} - 1, \text{value} - 6 \rangle,$
|             | $\langle \text{index} - 2, \text{value} - 9 \rangle,$
|             | $\langle \text{index} - 3, \text{value} - 2 \rangle,$
|             | $\langle \text{index} - 4, \text{value} - 9 \rangle$ |       |           |
| **Typical** | $|\text{TABLE}| > 1$
|             | $\text{range}(\text{TABLE.value}) > 1$ |       |           |
| **Symmetries** | $\bullet$ Items of $\text{TABLE}$ are permutable.<br>$\bullet$ All occurrences of two distinct values in $\text{ITEM.value}$ or $\text{TABLE.value}$ can be swapped; all occurrences of a value in $\text{ITEM.value}$ or $\text{TABLE.value}$ can be renamed to any unused value. |       |           |
| **Arg. properties** | Functional dependency: $\text{ITEM.value}$ determined by $\text{ITEM.index}$ and $\text{TABLE}$. |       |           |
Usage

Makes the link between the discrete decision variable INDEX and the variable VALUE according to a given table of values TABLE. We now give five typical uses of the elem constraint.

1. In some problems we may have to represent a function $y = f(x)$ (with $x \in [1, m]$).

In this context we generate the following elem constraint where INDEX is a domain variable taking is values in $\{1, 2, \ldots, m\}$:

$$\text{elem} \begin{pmatrix}
\langle \text{index} - x \text{, value} - y \rangle , \\
\langle \text{index} - 1 \text{, value} - f(1) \rangle , \\
\langle \text{index} - 2 \text{, value} - f(2) \rangle , \\
\vdots \\
\langle \text{index} - m \text{, value} - f(m) \rangle
\end{pmatrix}$$

Figure 5.237: $y = x^3 \ (1 \leq x \leq 3)$

As an example, consider the problem of finding the smallest integer that can be decomposed in two different ways in the sum of two cubes \[187\]. The elem constraint can be used for representing the function $y = x^3$ (Figure 5.237). The unique solution $1729 = 12^3 + 1^3 = 10^3 + 9^3$ can be obtained by the following set of constraints:
3. In some problems we need to express a disjunction of the form \( \text{VAR} \land \text{VAR} \land \cdots \land \text{VAR} \).

4. In some scheduling problems the duration of a task depends on the machine where the task will be assigned in final schedule. In this case we generate for each task an \( \text{elem} \) constraint of the following form:

\[
\begin{align*}
\text{elem}(\langle \text{index} - \text{VAR} - \text{index} - \text{VAR} \rangle), \\
\text{elem}(\langle \text{index} - \text{VAR} - \text{index} - \text{VAR} \rangle), \\
\text{elem}(\langle \text{index} - \text{VAR} - \text{index} - \text{VAR} \rangle), \\
\text{elem}(\langle \text{index} - \text{VAR} - \text{index} - \text{VAR} \rangle), \\
y_1 + y_2 = y_3 + y_4 \\
x_1 < x_2 \\
x_3 < x_4 \\
x_1 < x_3
\end{align*}
\]

The last three inequalities constraints in the conjunction are used for breaking symmetries. The constraints \( x_1 < x_2 \) and \( x_3 < x_4 \) respectively order the pairs of variables \( (x_1, x_2) \) and \( (x_3, x_4) \) from which the sums \( x_1^2 + x_2^2 \) and \( x_3^2 + x_4^2 \) are generated. Finally the inequality \( x_1 < x_3 \) enforces a lexicographic ordering between the two pairs of variables \( (x_1, x_2) \) and \( (x_3, x_4) \).

2. In some optimisation problems a classical use of the \( \text{elem} \) constraint consists expressing the link between a discrete choice and its corresponding cost. For each discrete choice we create an \( \text{elem} \) constraint of the form:

\[
\begin{align*}
\text{elem}(\langle \text{index} - \text{Choice} \rangle) \\
\text{elem}(\langle \text{index} - \text{Choice} \rangle) \\
\text{elem}(\langle \text{index} - \text{Choice} \rangle) \\
\text{elem}(\langle \text{index} - \text{Choice} \rangle) \\
\text{index} - 1 \text{ value} - \text{Cost}_1, \\
\text{index} - 2 \text{ value} - \text{Cost}_2, \\
\text{index} - m \text{ value} - \text{Cost}_m
\end{align*}
\]

where:

- \text{Choice} is a domain variable that indicates which alternative will be finally selected,
- \text{Cost} is a domain variable that corresponds to the cost of the decision associated with the value of the \text{Choice} variable,
- \text{Cost}_1, \text{Cost}_2, \ldots, \text{Cost}_m are the respective costs associated with the alternatives \( 1, 2, \ldots, m \).

3. In some problems we need to express a disjunction of the form \( \text{VAR} = \text{VAR}_1 \lor \text{VAR} = \text{VAR}_2 \lor \cdots \lor \text{VAR} = \text{VAR}_n \). This can be directly reformulated as the following \( \text{elem} \) constraint, where \text{INDEX} is a domain variable taking its value in the finite set \( \{1, 2, \ldots, n\} \) and where the \text{TABLE} argument corresponds to the domain variables \( \text{VAR}_1, \text{VAR}_2, \ldots, \text{VAR}_n \):

\[
\begin{align*}
\text{elem}(\langle \text{index} - \text{INDEX} \rangle) \\
\text{elem}(\langle \text{index} - \text{INDEX} \rangle) \\
\text{elem}(\langle \text{index} - \text{INDEX} \rangle) \\
\text{index} - 1 \text{ value} - \text{VAR}_1, \\
\text{index} - 2 \text{ value} - \text{VAR}_2, \\
\text{index} - n \text{ value} - \text{VAR}_n
\end{align*}
\]
where:

- **Machine** is a domain variable that indicates the resource to which the task will be assigned,
- **Duration** is a domain variable that corresponds to the duration of the task,
- **Dur$_1$, Dur$_2$, ..., Dur$_m$** are the respective duration of the task according to the hypothesis that it runs on machine 1, 2 or $m$.

Figure 5.238: A task for which the duration depends on the machine to which it is assigned (e.g., if Machine = 1 then Duration = Dur$_1$ = 4, if Machine = 2 then Duration = Dur$_2$ = 6, if Machine = 3 then Duration = Dur$_3$ = 4)

**Figure 5.238** illustrates this particular use of the elem constraint for modelling that a task has a duration of 4, 6 and 4 when we respectively assign it on machines 1, 2 and 3.

5. In some vehicle routing problems we typically use the elem constraint to express the distance between location $i$ and the next location visited by a vehicle. For this purpose we generate for each location $i$ an elem constraint of the form:

\[
\text{elem} \left( \langle \text{index} - \text{Next}_i, \text{value} - \text{distance}_i \rangle, \langle \text{index} - 1, \text{value} - \text{Dist}_{i1} \rangle, \langle \text{index} - 2, \text{value} - \text{Dist}_{i2} \rangle, \ldots, \langle \text{index} - m, \text{value} - \text{Dist}_{im} \rangle \right)
\]

where:

- **Next$_i$** is a domain variable that gives the index of the location the vehicle will visit just after location $i$,
- **distance$_i$** is a domain variable that corresponds to the distance between location $i$ and the location the vehicle will visit just after,
- **Dist$_{i1}, Dist_{i2}, ..., Dist_{im}$** are the respective distances between location $i$ and locations 1, 2, ..., $m$. 
An other example where the table argument corresponds to domain variables is described in the keyword entry assignment to the same set of values.

**Remark**

Originally, the parameters of the elem constraint had the form elem(INDEX, TABLE, VALUE), where INDEX and VALUE were two domain variables and TABLE was a list of non-negative integers.

Within some systems (e.g., Gecode), the index of the first entry of the table TABLE corresponds to 0 rather than to 1.

When the first entry of the table TABLE corresponds to a value \( p \) that is different from 1 we can still use the elem constraint. We use the reformulation \( I = J - p + 1 \land elem(\langle index - I, value - V \rangle, \text{TABLE}) \), where \( I \) and \( J \) are domain variables respectively ranging from 1 to |TABLE| and from \( p \) to \( p + |\text{TABLE}|-1 \).

**Systems**

nth in Choco, element in Gecode, element in JaCoP, element in SICStus.

**See also**

common keyword: elem_from_to, element_matrix, element_product, element_sparse (array constraint), elements_sparse, stage_element (data constraint).

implied by: element.

implies: element (single item replaced by two variables), element_greater_eq, element_less_eq.

system of constraints: elements.

uses in its reformulation: elements_all_different.

**Keywords**

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint arguments: pure functional dependency.

constraint network structure: centered cyclic(2) constraint network(1).

constraint type: data constraint.

filtering: arc-consistency.

heuristics: labelling by increasing cost, regret based heuristics.

modelling: array constraint, table, functional dependency, variable indexing, variable subscript, disjunction, assignment to the same set of values, sequence dependent set-up.

modelling exercises: assignment to the same set of values, sequence dependent set-up, zebra puzzle.

puzzles: zebra puzzle.
Arc input(s)  
ITEM TABLE

Arc generator  
PRODUCT $\rightarrow$ collection(item, table)

Arc arity  
2

Arc constraint(s)  
• item.index = table.index
• item.value = table.value

Graph property(ies)  
NARC = 1

Graph model  
We regroup the INDEX and VALUE parameters of the original element constraint element(INDEX, TABLE, VALUE) into the parameter ITEM. We also make explicit the different indices of the table TABLE.

Parts (A) and (B) of Figure 5.239 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the unique arc of the final graph is stressed in bold.

![Figure 5.239: Initial and final graph of the elem constraint](image)

Signature  
Since all the index attributes of TABLE are distinct and because of the first condition of the arc constraint the final graph cannot have more than one arc. Therefore we can rewrite NARC = 1 to NARC $\geq$ 1 and simplify NARC to NARC.
Automaton

Figure 5.240 depicts the automaton associated with the elem constraint. Let INDEX and VALUE respectively be the index and the value attributes of the unique item of the ITEM collection. Let INDEX, and VALUE, respectively be the index and the value attributes of item i of the TABLE collection. To each quadruple (INDEX, VALUE, INDEX, VALUE) corresponds a 0-1 signature variable Si as well as the following signature constraint: ((INDEX = INDEX) ∧ (VALUE = VALUE)) ⇔ Si.

Figure 5.240: Automaton of the elem constraint

Figure 5.241: Hypergraph of the reformulation corresponding to the automaton of the elem constraint
### 5.126 elem_from_to

**Description**
- **Origin**: Derived from `elem`.
- **Constraint**: `elem_from_to(ITEM,TABLE)`
- **Synonym**: `element_from_to`
- **Arguments**:
  - `ITEM`: `collection`
    - `from` - `dvar`
    - `cst` - `int`
  - `TABLE`: `collection(index-int,value-dvar)`
- **Restrictions**:
  - `required(ITEM,[from.cst_from,to.cst_to,value])`
  - `ITEM.from ≥ 1`
  - `ITEM.from ≤ |TABLE|`
  - `ITEM.to ≥ 1`
  - `ITEM.to ≤ |TABLE|`
  - `ITEM.from ≤ ITEM.to`
  - `|ITEM| = 1`
  - `required(TABLE,[index,value])`
  - `TABLE.index ≥ 1`
  - `TABLE.index ≤ |TABLE|`
  - `increasing_seq(TABLE,[index])`

**Purpose**
Let `FROM`, `CST_FROM`, `TO`, `CST_TO`, `VALUE` respectively denote the attributes `ITEM[1].from`, `ITEM[1].cst_from`, `ITEM[1].to`, `ITEM[1].cst_to`, `ITEM[1].value` of the unique item of the `ITEM` collection.
Beside imposing the fact that `FROM ≤ TO` and that both `FROM` and `TO` are assigned a value in `[1, |TABLE|]`, the `elem_from_to` constraint enforces the following condition:
All entries of the `TABLE` collection from position `max(1, FROM + CST_FROM)` to position `min(|TABLE|, TO + CST_TO)` are equal to `VALUE`. When `max(1, FROM + CST_FROM)` is strictly greater than `min(|TABLE|, TO + CST_TO)` the constraint holds no matter what value is assigned to `VALUE`.

**Example**

```
(⟨from − 1,cst_from − 1⟩ to − 4,cst_to − 1⟩ value − 2),
⟨index − 1⟩ value − 6,
⟨index − 2⟩ value − 2,
⟨index − 3⟩ value − 2,
⟨index − 4⟩ value − 9,
⟨index − 5⟩ value − 9
```

The `elem_from_to` constraint holds since all entries between position `max(1, FROM + CST_FROM) = max(1,1 + 1) = 2` and position `min(|TABLE|, TO + CST_TO) = min(5, 4 − 1) = 3` are equal to 2.
Typical

20091115

955

ITEM.
cst
from
≥
0
ITEM.
cst
from
≤
1
ITEM.
cst
to
≥
−
1
ITEM.
cst
to
≤
1
|TABLE| > 1

range(TABLE.value) > 1

Symmetry

All occurrences of two distinct values in ITEM.value or TABLE.value can be swapped; all occurrences of a value in ITEM.value or TABLE.value can be renamed to any unused value.

Usage

Given an array $t[1..n]$ of integers (i.e., an array of integers for which the entries are defined between 1 and $n$), the elem_from_to constraint is for instance useful for encoding expressions of the form $\exists i \in [1,n], \forall j \in [i+1,n] \mid t[i] = 0$. Note that, when the interval $[i+1,n]$ is empty, the condition $\forall j \in [i+1,n] \mid t[i] = 0$ is satisfied and $i$ is equal to $n$. This example is encoded by using an elem_from_to constraint and by respectively setting:

- FROM to $i$, where $i$ is a variable that is assigned a value from interval $[1,n]$,
- CST_FROM to constant 1,
- TO to $n$, the index of the last entry of the array $t[1..n]$,
- CST_TO to constant 0,
- VALUE to 0, the value we are looking for.
- TABLE to the array of integers $t[1..n]$.

Finally, note that $j$ is not used at all.

See also

common keyword: elem, element (array constraint).

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint,

constraint type: data constraint.

filtering: arc-consistency.

modelling: array constraint, table, variable indexing, variable subscript.
Automaton

Figure 5.242 depicts the automaton associated with the \textit{elem\_from\_to} constraint.

Let us first introduce some notations:

- Let $n$ denote the number of items of the \texttt{TABLE} collection.
- Let $\text{INDEX}_i$ and $\text{VALUE}_i$ respectively be the index and the value attributes of the $i^{th}$ item of the \texttt{TABLE} collection.
- Let $\text{FROM}, \text{CST\_FROM}, \text{TO}, \text{CST\_TO}, \text{VALUE}$ respectively denote the attributes \texttt{ITEM[1].from}, \texttt{ITEM[1].cst\_from}, \texttt{ITEM[1].to}, \texttt{ITEM[1].cst\_to}, \texttt{ITEM[1].value} of the unique item of the \texttt{ITEM} collection.
- Let $\text{IN}$ be a shortcut for condition $1 \leq \text{FROM} \land \text{TO} \leq \text{CST\_FROM}$.  
- Let $F$ and $T$ respectively denote the quantities $\max(1, \text{FROM} + \text{CST\_FROM})$ and $\min(|\text{TABLE}|, \text{TO} + \text{CST\_TO})$.

To each septuple $(\text{FROM}, \text{TO}, F, T, \text{VALUE}, \text{INDEX}_i, \text{VALUE}_i)$ corresponds a signature variable $S_i$ as well as the following signature constraint:

\[
\begin{align*}
& (\text{IN} \land F > T) \quad \iff S_i = 0 \land \\
& (\text{IN} \land F \leq T \land F > \text{INDEX}_i) \quad \iff S_i = 1 \land \\
& (\text{IN} \land F \leq T \land T < \text{INDEX}_i) \quad \iff S_i = 2 \land \\
& (\text{IN} \land F \leq T \land F \leq \text{INDEX}_i \land \text{INDEX}_i \leq T \land \text{VALUE} = \text{VALUE}_i) \quad \iff S_i = 3 \land \\
& (\text{IN} \land F \leq T \land F \leq \text{INDEX}_i \land \text{INDEX}_i \leq T \land \text{VALUE} \neq \text{VALUE}_i) \quad \iff S_i = 4
\end{align*}
\]

Figure 5.242: Automaton of the \textit{elem\_from\_to} constraint

Figure 5.243: Hypergraph of the reformulation corresponding to the automaton of the \textit{elem\_from\_to} constraint
5.127 element

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[393]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>\texttt{element(INDEX,TABLE,VALUE)}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>\texttt{nth}, \texttt{element}, \texttt{array}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>\texttt{INDEX} : \texttt{dvar} \hspace{1em} \texttt{TABLE} : \texttt{collection(value-dvar)} \hspace{1em} \texttt{VALUE} : \texttt{dvar}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>\texttt{INDEX} \geq 1 \hspace{1em} \texttt{INDEX} \leq</td>
<td>\texttt{TABLE}</td>
<td>\texttt{</td>
</tr>
<tr>
<td>Purpose</td>
<td>\texttt{VALUE} is equal to the \texttt{INDEX}^{th} item of \texttt{TABLE}.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>\texttt{(3, (6, 9, 2, 9), 2)}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Typical</td>
<td>\texttt{</td>
<td>TABLE</td>
<td>} &gt; 1 \hspace{1em} \texttt{range(TABLE.value)} &gt; 1</td>
</tr>
<tr>
<td>Symmetry</td>
<td>All occurrences of two distinct values in \texttt{TABLE.value} or \texttt{VALUE} can be swapped; all occurrences of a value in \texttt{TABLE.value} or \texttt{VALUE} can be renamed to any unused value.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arg. properties</td>
<td>\hspace{1em} \textbullet \hspace{1em} Functional dependency: \texttt{VALUE} determined by \texttt{INDEX} and \texttt{TABLE}. \hspace{1em} \textbullet \hspace{1em} Suffix-extensible wrt. \texttt{TABLE}.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Usage</td>
<td>See Usage slof of \texttt{elem}.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remark</td>
<td>In the original \texttt{element} constraint of \texttt{CHIP} the \texttt{index} attribute was not explicitly present in the table of values. It was implicitly defined as the position of a value in the previous table. \hspace{1em} Within some systems (e.g., \texttt{Gecode}), the index of the first entry of the table \texttt{TABLE} corresponds to 0 rather than to 1. \hspace{1em} When the first entry of the table \texttt{TABLE} corresponds to a value \texttt{p} that is different from 1 we can still use the \texttt{element} constraint. We use the reformulation \texttt{I = J - p + 1} and \texttt{I \geq 0}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
element($I$, TABLE, $V$), where $I$ and $J$ are domain variables respectively ranging from 1 to $|TABLE|$ and from $p$ to $p + |TABLE| - 1$.

The element constraint is called $nth$ in Choco (http://choco.sourceforge.net/). It is also sometimes called $element\_var$ when the second argument corresponds to a table of variables.

The case constraint [94] is a generalisation of the element constraint, where the table is replaced by a directed acyclic graph describing the set of solutions.

Systems


See also

common keyword: $elem\_from\_to$, $element\_greatereq$, $element\_lesseq$, $element\_matrix$, $element\_product$, $element\_sparse(array\_constraint)$, $elementn$, $elements\_sparse$, $in\_relation$, $stage\_element$, $sum(data\_constraint)$.

generalisation: $cond\_lex\_cost$ (variable replaced by tuple of variables).

implied by: $elem$.

implies: $elem$.

related: $twin((pairs\ linked\ by\ an\ element\ with\ the\ same\ table))$.

system of constraints: $elements$.

uses in its reformulation: $cycle$, $elements\_alldifferent$, $sort\_permutation$, $tree\_range$, $tree\_resource$.

Keywords

characteristic of a constraint: $core$, $automaton$, $automaton\ without\ counters$, $reified\ automaton\ constraint$, $derived\ collection$.

constraint arguments: pure functional dependency.

constraint network structure: centered cyclic(2) constraint network(1).

constraint type: data constraint.

filtering: arc-consistency.

heuristics: labelling by increasing cost, regret based heuristics.

modelling: array constraint, table, functional dependency, variable indexing, variable subscript, disjunction, assignment to the same set of values, sequence dependent set-up.

modelling exercises: assignment to the same set of values, sequence dependent set-up, zebra puzzle.

puzzles: zebra puzzle.
Derived Collection

\[
\text{col}(\text{ITEM} - \text{collection} (\text{index} - \text{dvar}, \text{value} - \text{dvar}),
\[\text{item}(\text{index} - \text{INDEX}, \text{value} - \text{VALUE})]\)
\]

Arc input(s) ITEM TABLE
Arc generator \(PRODUCT \mapsto \text{collection} (\text{item}, \text{table})\)
Arc arity 2
Arc constraint(s)
- item.index = table.key
- item.value = table.value
Graph property(ies) NARC = 1

Graph model

The original element constraint with three arguments. We use the derived collection ITEM for putting together the INDEX and VALUE parameters of the element constraint. Within the arc constraint we use the implicit attribute key that associates to each item of a collection its position within the collection.

Parts (A) and (B) of Figure 5.244 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the unique arc of the final graph is stressed in bold.

![Graph Diagram](A) (B)

Figure 5.244: Initial and final graph of the element constraint

Signature

Because of the first condition of the arc constraint the final graph cannot have more than one arc. Therefore we can rewrite NARC = 1 to NARC \(\geq 1\) and simplify NARC to NARC.
Automaton

Figure 5.245 depicts the automaton associated with the element constraint. Let \( \text{VALUE}, \text{INDEX} \) be the value attribute of item \( i \) of the TABLE collection. To each triple \((\text{INDEX}, \text{VALUE}, \text{VALUE}_i)\) corresponds a 0-1 signature variable \( S_i \) as well as the following signature constraint: \((\text{INDEX} = i \land \text{VALUE} = \text{VALUE}_i) \Leftrightarrow S_i\).

Figure 5.245: Automaton of the element constraint

Figure 5.246: Hypergraph of the reformulation corresponding to the automaton of the element constraint
5.128 element_greatereq

### Origin
[281]

### Constraint
\[ \text{element\_greatesteq}(\text{ITEM}, \text{TABLE}) \]

### Arguments
- **ITEM**: collection(index\_dvar, value\_dvar)
- **TABLE**: collection(index\_int, value\_int)

### Restrictions
\[
\begin{align*}
\text{required}(&\text{ITEM}, \text{[index, value]})) \\
\text{ITEM}.\text{index} & \geq 1 \\
\text{ITEM}.\text{index} & \leq |\text{TABLE}| \\
|\text{ITEM}| & = 1 \\
|\text{TABLE}| & > 0 \\
\text{required}(&\text{TABLE}, \text{[index, value]})) \\
\text{TABLE}.\text{index} & \geq 1 \\
\text{TABLE}.\text{index} & \leq |\text{TABLE}| \\
\text{distinct}(&\text{TABLE}, \text{index})
\end{align*}
\]

### Purpose
**ITEM[1].value** is greater than or equal to one of the entries (i.e., the value attribute) of the table \(\text{TABLE}\).

### Example
\[
\begin{pmatrix}
\langle \text{index} - 1, \text{value} - 8 \rangle, \\
\langle \text{index} - 1, \text{value} - 6 \rangle, \\
\langle \text{index} - 2, \text{value} - 9 \rangle, \\
\langle \text{index} - 3, \text{value} - 2 \rangle, \\
\langle \text{index} - 4, \text{value} - 9 \rangle
\end{pmatrix}
\]

The `element_greatereq` constraint holds since \(\text{ITEM}[1].\text{value} = 8\) is greater than or equal to \(\text{TABLE}/\text{ITEM}[1].\text{index}.\text{value} = \text{TABLE}[1].\text{value} = 6\).

### Typical
- \(|\text{TABLE}| > 1\)
- \(\text{range}(\text{TABLE}.\text{value}) > 1\)

### Symmetries
- Items of \(\text{TABLE}\) are **permutable**.
- All occurrences of two distinct values in \(\text{ITEM}.\text{value}\) or \(\text{TABLE}.\text{value}\) can be **swapped**: all occurrences of a value in \(\text{ITEM}.\text{value}\) or \(\text{TABLE}.\text{value}\) can be **renamed** to any unused value.

### Usage
Used for modelling variable subscripts in linear constraints [281].

### Reformulation
By introducing an extra variable VAL, the `element_greatereq` constraint can be expressed in term of an `elem` constraint and of an inequality constraint `VALUE \geq VAL`. 
See also

common keyword: element, element_lesseq, element_product (array constraint).
implied by: elem.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.
constraint arguments: binary constraint.
constraint network structure: centered cyclic(2) constraint network(1).
constraint type: data constraint.
filtering: linear programming, arc-consistency.
modelling: array constraint, table, variable subscript, variable indexing.
Arc input(s)  
ITEM TABLE
Arc generator  
\[ \text{PRODUCT} \rightarrow \text{collection}(\text{item}, \text{table}) \]
Arc arity  
2
Arc constraint(s)  
- \text{item.index} = \text{table.index}
- \text{item.value} \geq \text{table.value}
Graph property(ies)  
NARC = 1

Graph model
Similar to the element constraint except that the equality constraint of the second condition of the arc constraint is replaced by a greater than or equal to constraint.

Parts (A) and (B) of Figure 5.247 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the unique arc of the final graph is stressed in bold.

![Graph model](image)

Figure 5.247: Initial and final graph of the element.greatereq constraint

Signature
Since all the index attributes of TABLE are distinct and because of the first arc constraint the final graph cannot have more than one arc. Therefore we can rewrite NARC = 1 to NARC \geq 1 and simplify NARC to NARC.
Automaton

Figure 5.248 depicts the automaton associated with the `element.greataeq` constraint. Let `INDEX` and `VALUE` respectively be the index and the value attributes of the unique item of the `ITEM` collection. Let `INDEX_i` and `VALUE_i` respectively be the index and the value attributes of the `i`th item of the `TABLE` collection. To each quadruple `(INDEX, VALUE, INDEX_i, VALUE_i)` corresponds a 0-1 signature variable $S_i$ as well as the following signature constraint: $(\text{INDEX} = \text{INDEX}_i) \land (\text{VALUE} \geq \text{VALUE}_i) \Leftrightarrow S_i$.

Figure 5.248: Automaton of the `element.greataeq` constraint

Figure 5.249: Hypergraph of the reformulation corresponding to the automaton of the `element.greataeq` constraint
5.129  element\_lesseq

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[281]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>element_lesseq(ITEM,TABLE)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Arguments   | ITEM : collection(index\_dvar,value\_dvar)  
TABLE : collection(index\_int,value\_int) |       |           |
| Restrictions| required(ITEM,[index,value])  
ITEM.index ≥ 1  
ITEM.index ≤ |TABLE|  
|ITEM| = 1  
|TABLE| > 0  
required(TABLE,[index,value])  
TABLE.index ≥ 1  
TABLE.index ≤ |TABLE|  
distinct(TABLE,index) |       |           |
| Purpose     | ITEM\[1\].value is less than or equal to one of the entries (i.e., the value attribute) of the table TABLE. |       |           |
| Example     | \begin{pmatrix}  
(index – 3 value – 1),  
index – 1 value – 6,  
index – 2 value – 9,  
(index – 3 value – 2),  
index – 4 value – 9 \end{pmatrix} |       |           |
| Typical     | |TABLE| > 1  
range(TABLE.value) > 1 |       |           |
| Symmetries  | • Items of TABLE are permutable.  
• All occurrences of two distinct values in ITEM.value or TABLE.value can be swapped; all occurrences of a value in ITEM.value or TABLE.value can be renamed to any unused value. |       |           |
| Usage       | Used for modelling variable subscripts in linear constraints [281]. |       |           |
| Reformulation| By introducing an extra variable VAL, the element\_lesseq((index – INDEX value – VALUE), TABLE) constraint can be expressed in term of an elem((index – INDEX value – VAL), TABLE) constraint and of an inequality constraint VALUE ≤ VAL. |       |           |
See also

commom keyword: element, element_greatereq.

element_product (array constraint).

implied by: elem.

Keywords

characteristic of a constraint: automaton, automaton without counters,

reified automaton constraint.

constraint arguments: binary constraint.

constraint network structure: centered cyclic(2) constraint network(1).

constraint type: data constraint.

filtering: linear programming, arc-consistency.

modelling: array constraint, table, variable subscript, variable indexing.
### Arc input(s)
- ITEM TABLE

### Arc generator
- `$PRODUCT \rightarrow \text{collection}(\text{item,table})$`

### Arc arity
- 2

### Arc constraint(s)
- `item.index = table.index`
- `item.value ≤ table.value`

### Graph property(ies)
- NARC = 1

#### Graph model
Similar to the `element` constraint except that the equality constraint of the second condition of the arc constraint is replaced by a `less than or equal to` constraint.

Parts (A) and (B) of Figure 5.250 respectively show the initial and final graph associated with the `Example` slot. Since we use the NARC graph property, the unique arc of the final graph is stressed in bold.

![Graph diagram](image)

**Figure 5.250**: Initial and final graph of the `element_less_eq` constraint

#### Signature
Since all the `index` attributes of TABLE are distinct and because of the first arc constraint the final graph cannot have more than one arc. Therefore we can rewrite NARC = 1 to NARC ≥ 1 and simplify NARC to NARC.
Automaton

Figure 5.251 depicts the automaton associated with the `element_lesseq` constraint. Let INDEX and VALUE respectively be the index and the value attributes of the unique item of the ITEM collection. Let INDEX$_i$ and VALUE$_i$ respectively be the index and the value attributes of the $i^{th}$ item of the TABLE collection. To each quadruple (INDEX, VALUE, INDEX$_i$, VALUE$_i$) corresponds a 0-1 signature variable $S_i$ as well as the following signature constraint: $(\text{INDEX} = \text{INDEX}_i) \land (\text{VALUE} \leq \text{VALUE}_i) \iff S_i$.

![Automaton Diagram](image)

Figure 5.251: Automaton of the `element_lesseq` constraint

![Hypergraph Diagram](image)

Figure 5.252: Hypergraph of the reformulation corresponding to the automaton of the `element_lesseq` constraint
## 5.130 element_matrix

<table>
<thead>
<tr>
<th>Description</th>
<th>Links</th>
<th>Graph</th>
<th>Automaton</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>CHIP</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>(\text{element_matrix}(MAX_1, MAX_J, INDEX_I, INDEX_J, MATRIX, VALUE))</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Synonyms</strong></td>
<td>\text{elem_matrix}, \text{matrix}.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| **Arguments** | \begin{align*} 
MAX_I & : \text{int} \\
MAX_J & : \text{int} \\
INDEX_I & : \text{dvar} \\
INDEX_J & : \text{dvar} \\
MATRIX & : \text{collection}(i-\text{int}, j-\text{int}, v-\text{int}) \\
VALUE & : \text{dvar} 
\end{align*} |        |           |
| **Restrictions** | \begin{align*} 
MAX_I & \geq 1 \\
MAX_J & \geq 1 \\
INDEX_I & \geq 1 \\
INDEX_I & \leq MAX_I \\
INDEX_J & \geq 1 \\
INDEX_J & \leq MAX_J \\
\text{required}(\text{MATRIX}, [i, j, v]) \\
\text{increasing_seq}(\text{MATRIX}, [i, j]) \\
\text{MATRIX}.i & \geq 1 \\
\text{MATRIX}.i & \leq MAX_I \\
\text{MATRIX}.j & \geq 1 \\
\text{MATRIX}.j & \leq MAX_J \\
|\text{MATRIX}| & = MAX_I * MAX_J 
\end{align*} |        |           |
| **Purpose** | The \text{MATRIX} collection corresponds to the two-dimensional matrix \(\text{MATRIX}[1..MAX_I, 1..MAX_J]\). VALUE is equal to the entry \(\text{MATRIX}[INDEX_I, INDEX_J]\) of the previous matrix. |        |           |
| **Example** | \[
\begin{pmatrix}
4, 3, 1, 3,
\begin{pmatrix}
i - 1 & j - 1 & v - 4, \\
i - 1 & j - 2 & v - 1, \\
i - 1 & j - 3 & v - 7, \\
i - 2 & j - 1 & v - 1, \\
i - 2 & j - 2 & v - 0, \\
i - 2 & j - 3 & v - 8, \\
i - 3 & j - 1 & v - 3, \\
i - 3 & j - 2 & v - 2, \\
i - 3 & j - 3 & v - 1, \\
i - 4 & j - 1 & v - 0, \\
i - 4 & j - 2 & v - 0, \\
i - 4 & j - 3 & v - 6 \\
\end{pmatrix}, 7
\end{pmatrix}
\] |        |           |
The `element_matrix` constraint holds since its last argument VALUE = 7 is equal to the \( v \) attribute of the \( k^{th} \) item of the MATRIX collection such that MATRIX[\( k \)].\( i = INDEX, i = 1 \) and MATRIX[\( k \)].\( j = INDEX, j = 3 \).

**Typical**

\[
\begin{align*}
\text{MAX}_I & > 1 \\
\text{MAX}_J & > 1 \\
|\text{MATRIX}| & > 3 \\
\text{maxval}(\text{MATRIX}.i) & > 1 \\
\text{maxval}(\text{MATRIX}.j) & > 1 \\
\text{range}(\text{MATRIX}.v) & > 1
\end{align*}
\]

**Symmetry**

All occurrences of two distinct values in `MATRIX.v` or VALUE can be swapped; all occurrences of a value in `MATRIX.v` or VALUE can be renamed to any unused value.

**Reformulation**

The `element_matrix(MAX_I, MAX_J, INDEX_I, INDEX_J, MATRIX, VALUE)` constraint can be expressed in term of `MAX_I element(INDEX_J.LINE,VAR) (i \in [1, MAX_I])`, where LINE \( i \) corresponds to the \( i \)-th line of the matrix MATRIX and of one `element(INDEX_I, (VAR_1, VAR_2, \ldots, VAR_{MAX_J}), VALUE)` constraint.

If we consider the **Example** slot we get the following `element` constraints:

- `element(3, (4, 1, 7), 7),`
- `element(3, (1, 0, 8), 8),`
- `element(3, (3, 2, 1), 1),`
- `element(3, (0, 0, 6), 6),`
- `element(1, (7, 8, 1, 6), 7).`

**Systems**

\textit{nth} in Choco, \textit{element} in Gecode.

**See also**

common keyword: \textit{elem, element (array constraint)}.

**Keywords**

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint, derived collection.

constraint arguments: ternary constraint.

constraint network structure: centered cyclic(3) constraint network(1).

constraint type: data constraint.

filtering: arc-consistency.

modelling: array constraint, matrix.
Derived Collection

\[ \text{col}(\text{ITEM}\rightarrow\text{collection}([\text{item} \in \text{INDEX}_I, \text{value} \in \text{VALUE}]), \text{ITEM}\rightarrow\text{collection}([\text{item} \in \text{INDEX}_J, \text{value} \in \text{VALUE}]), \text{PRODUCT}\rightarrow\text{collection}(\text{item}, \text{matrix})) \]

Arc input(s)
ITEM MATRIX

Arc generator
PRODUCT \rightarrow \text{collection}(\text{item}, \text{matrix})

Arc arity
2

Arc constraint(s)
- item.index_i = matrix.i
- item.index_j = matrix.j
- item.value = matrix.v

Graph property(ies)
NARC = 1

Graph model
Similar to the element constraint except that the arc constraint is updated according to the fact that we have a two-dimensional matrix.

Parts (A) and (B) of Figure 5.253 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the unique arc of the final graph is stressed in bold.

Signature
Because of the first condition of the arc constraint the final graph cannot have more than one arc. Therefore we can rewrite NARC = 1 to NARC \geq 1 and simplify NARC to NARC.
Automaton

Figure 5.254 depicts the automaton associated with the *element_matrix* constraint. Let $I_k$, $J_k$ and $V_k$ respectively be the $i$, the $j$ and the $v$ $k^{th}$ attributes of the MATRIX collection. To each sextuple $(INDEX_I, INDEX_J, VALUE, I_k, J_k, V_k)$ corresponds a 0-1 signature variable $S_k$ as well as the following signature constraint: $((INDEX_I = I_k) \land (INDEX_J = J_k) \land (VALUE = V_k)) \iff S_k$.

Figure 5.254: Automaton of the *element_matrix* constraint

![Automaton Diagram](image)

Figure 5.255: Hypergraph of the reformulation corresponding to the automaton of the *element_matrix* constraint

![Hypergraph Diagram](image)
### 5.131 element_product

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[280]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>element_product(Y, TABLE, X, Z)</td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td>element.</td>
<td></td>
</tr>
</tbody>
</table>
| Arguments   | Y : dvar  
TABLE : collection(value=int)  
X : dvar  
Z : dvar |       |
| Restrictions| Y ≥ 1  
Y ≤ |TABLE|  
X ≥ 0  
Z ≥ 0  
required(TABLE, value)  
TABLE.value ≥ 0 |       |
| Purpose     | Z is equal to the Y-th item of TABLE multiplied by X. |       |

#### Example

(3, (6, 9, 2, 9), 5, 10)

The element_product constraint holds since its fourth argument Z = 10 is equal to the 3rd (Y = 3) item of the collection (6, 9, 2, 9) multiplied by X = 5.

#### Typical

X > 0  
Z > 0  
|TABLE| > 1  
range(TABLE.value) > 1  
TABLE.value ≥ 0

#### Arg. properties

- **Functional dependency**: Z determined by Y, TABLE and X.  
- **Suffix-extensible** wrt. TABLE.

#### Usage

The element_product constraint was originally used in configuration problems [280]. In this context, Z denotes the cost of buying X units of type Y at cost TABLE(Y).value.

#### Reformulation

By introducing an extra variable VAL, the element_product(Y, TABLE, X, Z) constraint can be expressed in term of an element(Y, TABLE, VAL) constraint and of a product constraint Z = VAL · X.

#### See also

**common keyword**: elem, element, element_greatereq, element_lesseq (array constraint).
Keywords

- **application area**: configuration problem.
- **constraint arguments**: pure functional dependency.
- **constraint type**: data constraint.
- **modelling**: array constraint, table, functional dependency, variable subscript.
Derived Collection

\[ \text{col} \left( \text{ITEM} - \text{collection}(y - \text{dvar}, x - \text{dvar}, z - \text{dvar}), \right) \]
\[ \left[ \text{item}(y - Y, x - X, z - Z) \right] \]

Arc input(s)  ITEM TABLE
Arc generator  \( \text{PRODUCT} \rightarrow \text{collection}(\text{item, table}) \)
Arc arity  2
Arc constraint(s)
• \( \text{item}.y = \text{table}\.key \)
• \( \text{item}.z = \text{item}.x \times \text{table}\.value \)
Graph property(ies)  \( \text{NARC} = 1 \)

Graph model
We use the derived collection ITEM for putting together the \( Y \), the \( X \) and \( Z \) parameters of the \texttt{element.product} constraint. Within the arc constraint we use the implicit attribute \texttt{key} that associates to each item of a collection its position within the collection.

Parts (A) and (B) of Figure 5.256 respectively show the initial and final graph associated with the \texttt{Example} slot. Since we use the \texttt{NARC} graph property, the unique arc of the final graph is stressed in bold.

![Graph](image)

**Figure 5.256:** Initial and final graph of the \texttt{element.product} constraint

Signature
Because of the first condition of the arc constraint the final graph cannot have more than one arc. Therefore we can rewrite \( \text{NARC} = 1 \) to \( \text{NARC} \geq 1 \) and simplify \( \text{NARC} \) to \( \text{NARC} \).
5.132 element_sparse

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>CHIP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>element_sparse(ITEM, TABLE, DEFAULT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Usual name</td>
<td>element</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>ITEM : collection(index−dvar, value−dvar)</td>
<td>TABLE : collection(index−int, value−int)</td>
<td>DEFAULT : int</td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(ITEM, [index, value])</td>
<td>ITEM.index ≥ 1</td>
<td>ITEM</td>
</tr>
<tr>
<td>Purpose</td>
<td>ITEM[1].value is equal to one of the entries of the table TABLE or to the default value DEFAULT if the entry ITEM[1].index does not exist in TABLE.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>(index − 2 value − 5), (index − 1 value − 6), (index − 2 value − 5), (index − 4 value − 2), (index − 8 value − 9)</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Typical</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symmetries</td>
<td>Items of TABLE are permutable.</td>
<td>All occurrences of two distinct values in ITEM.value, TABLE.value or DEFAULT can be swapped; all occurrences of a value in ITEM.value, TABLE.value or DEFAULT can be renamed to any unused value.</td>
<td></td>
</tr>
<tr>
<td>Usage</td>
<td>A sometimes more compact form of the element constraint: we are not obliged to specify explicitly the table entries that correspond to the specified default value. This can sometimes reduce drastically memory utilisation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remark</td>
<td>The original constraint of CHIP had an additional parameter SIZE giving the maximum value of ITEM.index.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Reformulation

Let I and V respectively denote ITEM[1].index and ITEM[1].value. The element_sparse((ITEM, TABLE, DEFAULT)) constraint can be expressed in term of a reified constraint of the form:

\[(I = \text{TABLE}[1].\text{index} \land V = \text{TABLE}[1].\text{value}) \lor\]
\[(I = \text{TABLE}[2].\text{index} \land V = \text{TABLE}[2].\text{value}) \lor\]
\[\ldots\]
\[(I = \text{TABLE}[\text{TABLE}.\text{index}] . \text{index} \land V = \text{TABLE}[\text{TABLE}.\text{value}]) \lor\]
\[(I \neq \text{TABLE}[1].\text{index}) \land\]
\[(I \neq \text{TABLE}[2].\text{index}) \land\]
\[\ldots\]
\[(I \neq \text{TABLE}[\text{TABLE}.\text{index}] . \text{index}) \land\]
\[(V = \text{DEFAULT})].\]

See also

**common keyword:** elem, element (array constraint), elements_sparse (sparse table).
**implies:** elements_sparse.
**system of constraints:** elements_sparse.

Keywords

**characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint, derived collection.
**constraint arguments:** binary constraint.
**constraint network structure:** centered cyclic(2) constraint network(1).
**constraint type:** data constraint.
**filtering:** arc-consistency.
**modelling:** array constraint, table, sparse table, sparse functional dependency, variable indexing.
Derived Collections

\[
\begin{align*}
\text{col} & \left( \text{DEF} - \text{collection}(\text{index} - \text{int}, \text{value} - \text{int}), \right. \\
& \quad \left. \text{item}(\text{index} - 0, \text{value} - \text{DEFAULT}) \right), \\
\text{col} & \left( \text{TABLE}_{\text{DEF}} - \text{collection}(\text{index} - \text{dvar}, \text{value} - \text{dvar}), \right. \\
& \quad \left. \text{item}(\text{index} - \text{TABLE}.\text{index}, \text{value} - \text{TABLE}.\text{value}), \right. \\
& \quad \left. \text{item}(\text{index} - \text{DEF}.\text{index}, \text{value} - \text{DEF}.\text{value}) \right).
\end{align*}
\]

Arc input(s) \quad \text{ITEM \ TABLE, DEF}

Arc generator \quad \textit{PRODUCT} \rightarrow \text{collection}(\text{item, table_def})

Arc arity \quad 2

Arc constraint(s)
- \text{item.value} = \text{table_def.value}
- \text{item.index} = \text{table_def.index} \lor \text{table_def.index} = 0

Graph property(ies) \quad \textbf{NARC} \geq 1

Graph model

The final graph has between one and two arc constraints: it has two arcs when the default value \text{DEFAULT} occurs also in the table \text{TABLE}; otherwise it has only one arc.

Parts (A) and (B) of Figure 5.257 respectively show the initial and final graph associated with the \textbf{Example} slot. Since we use the \textbf{NARC} graph property the arcs of the final graph are outline with thick lines.

![Diagram](image)

Figure 5.257: Initial and final graph of the \texttt{element.sparse} constraint
Automaton

Figure 5.258 depicts the automaton associated with the `element_sparse` constraint. Let INDEX and VALUE respectively be the index and the value attributes of the unique item of the ITEM collection. Let INDEX and VALUE respectively be the index and the value attributes of the $i^{th}$ item of the TABLE collection. To each quintuple $(\text{INDEX}, \text{VALUE}, \text{DEFAULT}, \text{INDEX}_i, \text{VALUE}_i)$ corresponds a signature variable $S_i$ as well as the following signature constraint:

\[
\begin{align*}
(\text{INDEX} \neq \text{INDEX}_i, \ \text{VALUE} \neq \text{DEFAULT}) & \iff S_i = 0 \wedge \\
(\text{INDEX} = \text{INDEX}_i, \ \text{VALUE} = \text{VALUE}_i) & \iff S_i = 1 \wedge \\
(\text{INDEX} \neq \text{INDEX}_i, \ \text{VALUE} = \text{DEFAULT}) & \iff S_i = 2
\end{align*}
\]

Figure 5.258: Automaton of the `element_sparse` constraint

Figure 5.259: Hypergraph of the reformulation corresponding to the automaton of the `element_sparse` constraint
5.133  \textit{elementn}

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>P. Flener</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>\textit{elementn}($\text{INDEX}$, TABLE, ENTRIES)</td>
<td></td>
</tr>
</tbody>
</table>
| Arguments   | \begin{align*}
\text{INDEX} & : \text{dvar} \\
\text{TABLE} & : \text{collection} \text{value} \rightarrow \text{int} \\
\text{ENTRIES} & : \text{collection} \text{entry} \rightarrow \text{dvar}
\end{align*} |             |
| Restrictions| \begin{align*}
\text{INDEX} & \geq 1 \\
\text{INDEX} & \leq |\text{TABLE}| - |\text{ENTRIES}| + 1 \\
|\text{TABLE}| & > 0 \\
|\text{ENTRIES}| & > 0 \\
|\text{TABLE}| & \geq |\text{ENTRIES}| \\
\text{required}(\text{TABLE}.\text{value}) & \\
\text{required}(\text{ENTRIES}.\text{entry})
\end{align*} |             |
| Purpose     | $\forall i \in [1,|\text{ENTRIES}|]: \text{ENTRIES}[i].\text{entry} = \text{TABLE}[\text{INDEX} + i - 1].\text{value}$ |             |
| Example     | \begin{pmatrix} 3, & 6, & 9, & 2, & 9 \\ 2, & 9 \end{pmatrix} | The \textit{elementn} constraint holds since its third argument \text{ENTRIES} = (2, 9) is set to the subsequence starting at the third (i.e., INDEX = 3) item of the table \text{TABLE} = (6, 9, 2, 9). |
| Typical     | \begin{align*}
|\text{TABLE}| & > 1 \\
\text{range}(\text{TABLE}.\text{value}) & > 1 \\
|\text{ENTRIES}| & > 1
\end{align*} |             |
| Symmetry    | All occurrences of two distinct values in \text{TABLE}.\text{value} or \text{ENTRIES}.\text{entry} can be swapped; all occurrences of a value in \text{TABLE}.\text{value} or \text{ENTRIES}.\text{entry} can be renamed to any unused value. |             |
| Arg. properties | \textbf{Suffix-extensible} wrt. \text{TABLE}. |             |
| Usage       | The \textit{elementn} constraint is useful for extracting of subsequence of fixed length from a given sequence. |             |
| Reformulation| Let $I_1 = \text{INDEX}$, $I_2 = \text{INDEX} + 1, \ldots, I_{|\text{ENTRIES}|} = \text{INDEX} + |\text{ENTRIES}| - 1$. The \textit{elementn}($\text{INDEX}$, TABLE, \text{entries} $E_1, E_2, \ldots, E_{|\text{ENTRIES}|}$) constraint can be expressed in term of a conjunction of \text{ENTRIES} \textit{element} constraints of the form: \begin{align*}
\text{element}(I_1, \text{TABLE}, E_1) \\
\text{element}(I_2, \text{TABLE}, E_2) \\
\ldots
\text{element}(\text{INDEX} + |\text{ENTRIES}| - 1, \text{TABLE}, E_{|\text{ENTRIES}|})
\end{align*} |             |
See also

common keyword: element [(data constraint)].

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint,

constraint network structure: Berge-acyclic constraint network.

constraint type: data constraint, sliding sequence constraint.

filtering: arc-consistency.

modelling: table.
Automaton

Figure 5.260 depicts the automaton associated with the elementn constraint of the Example slot. Let $I$ and $E_k$ respectively denote the INDEX argument and the entry attribute of the $k^{th}$ item of the ENTRIES collection. Figure 5.261 depicts the reformulation of the elementn constraint.

Figure 5.260: Automaton of the elementn constraint given in the example

Figure 5.261: Hypergraph of the reformulation corresponding to the automaton of the elementn constraint
5.134 elements

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from <code>element</code>.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td><code>elements(ITEMS, TABLE)</code></td>
<td></td>
</tr>
</tbody>
</table>
| Arguments   | `ITEMS : collection(index−dvar, value−dvar)`
`TABLE : collection(index−int, value−dvar)` |       |
| Restrictions| `required(ITEMS,[index,value])`
`ITEMS.index ≥ 1`
`ITEMS.index ≤ |TABLE|`
`required(TABLE,[index,value])`
`TABLE.index ≥ 1`
`TABLE.index ≤ |TABLE|`
`distinct(TABLE, index)` |       |
| Purpose     | All the items of `ITEMS` should be equal to one of the entries of the table `TABLE`. |       |
| Example     | `\[
(index - 4 \text{ value} - 9, index - 1 \text{ value} - 6),
(index - 2 \text{ value} - 9),
(index - 3 \text{ value} - 2),
(index - 4 \text{ value} - 9)
\]` |       |

The `elements` constraint holds since each item of its first argument `ITEMS` corresponds to an item of the `TABLE` collection: the first item `(index - 4 \text{ value} - 9)` of `ITEMS` corresponds to the fourth item of `TABLE`, while the second item `(index - 1 \text{ value} - 6)` of `ITEMS` corresponds to the first item of `TABLE`.

| Typical     | `|ITEMS| > 1`
`range(ITEMS.index) > 1`
`|TABLE| > 1`
`range(TABLE.value) > 1` |
|-------------|---------------------|
| Symmetries  | • Items of `ITEMS` are permutable.
• Items of `TABLE` are permutable.
• All occurrences of two distinct values in `ITEMS.value` or `TABLE.value` can be swapped; all occurrences of a value in `ITEMS.value` or `TABLE.value` can be renamed to any unused value. |
| Arg. properties | Functional dependency: `ITEMS.value` determined by `ITEMS.index` and `TABLE`. |
Usage
Used for replacing several element constraints sharing exactly the same table by one single constraint.

Reformulation
The $\text{elements}((\text{index} - I_1 \text{ value} - V_1, \text{index} - I_2 \text{ value} - V_2, \ldots, \text{index} - I_{|\text{ITEMS}|} \text{ value} - V_{|\text{ITEMS}|}, \text{TABLE})$ constraint can be expressed in term of a conjunction of $|\text{ITEMS}|$ $\text{elem}$ constraints of the form:

\[
\text{elem}(\text{index} - I_1 \text{ value} - V_1, \text{TABLE}), \\
\text{elem}(\text{index} - I_2 \text{ value} - V_2, \text{TABLE}), \\
\ldots \\
\text{elem}(\text{index} - I_{|\text{ITEMS}|} \text{ value} - V_{|\text{ITEMS}|}, \text{TABLE}).
\]

See also
implied by: $\text{elements\_alldifferent}$.
part of system of constraints: $\text{elem, element}$.

Keywords
constraint arguments: pure functional dependency.
constraint type: data constraint, system of constraints.
filtering: arc-consistency.
modelling: table, shared table, functional dependency.
Arc input(s) | ITEMS TABLE
Arc generator | \( \text{PRODUCT} \rightarrow \text{collection(items, table)} \)
Arc arity | 2
Arc constraint(s) | 
| \( \text{items.index} = \text{table.index} \) 
| \( \text{items.value} = \text{table.value} \)
Graph property(ies) | \( \text{NARC} = |\text{ITEMS}| \)

Graph model
Parts (A) and (B) of Figure 5.262 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

![Graph diagram with arcs highlighted]

Figure 5.262: Initial and final graph of the elements constraint

Signature
Since all the index attributes of TABLE collection are distinct and because of the first condition \( \text{items.index} = \text{table.index} \) of the arc constraint, a source vertex of the final graph can have at most one successor. Therefore \( |\text{ITEMS}| \) is the maximum number of arcs of the final graph and we can rewrite \( \text{NARC} = |\text{ITEMS}| \) to \( \text{NARC} \geq |\text{ITEMS}| \). So we can simplify \( \text{NARC} \) to \( \text{NARC} \).
5.135 elements_alldifferent

**DESCRIPTION**

<table>
<thead>
<tr>
<th>Origin</th>
<th>Derived from elements and alldifferent.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint</td>
<td>elements_alldifferent(ITEMS, TABLE)</td>
</tr>
<tr>
<td>Synonyms</td>
<td>elements_alldiff, elements_alldistinct.</td>
</tr>
<tr>
<td>Arguments</td>
<td>ITEMS : collection(index−dvar, value−dvar)</td>
</tr>
<tr>
<td></td>
<td>TABLE : collection(index−int, value−dvar)</td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(ITEMS, [index, value])</td>
</tr>
<tr>
<td></td>
<td>ITEMS.index ≥ 1</td>
</tr>
<tr>
<td></td>
<td>ITEMS.index ≤</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>required(TABLE, [index, value])</td>
</tr>
<tr>
<td></td>
<td>TABLE.index ≥ 1</td>
</tr>
<tr>
<td></td>
<td>TABLE.index ≤</td>
</tr>
<tr>
<td></td>
<td>distinct(TABLE, index)</td>
</tr>
<tr>
<td>Purpose</td>
<td>All the items of the ITEMS collection should be equal to one of the entries of the table TABLE and all the variables ITEMS.index should take distinct values.</td>
</tr>
</tbody>
</table>

**Example**

```
( index−2 value−9,
  index−1 value−6,
  index−4 value−9,
  index−3 value−2,
  index−1 value−6,
  index−2 value−9,
  index−3 value−2,
  index−4 value−9 )
```

The elements_alldifferent constraint holds since, as depicted by Figure 5.263, there is a one to one correspondence between the items of the ITEMS collection and the items of the TABLE collection.

Figure 5.263: Illustration of the one to one correspondence between the items of ITEMS and the items of TABLE.
Typical

\[ |\text{ITEMS}| > 1 \]
\[ \text{range}(\text{ITEMS}.\text{value}) > 1 \]
\[ |\text{TABLE}| > 1 \]
\[ \text{range}(\text{TABLE}.\text{value}) > 1 \]

Symmetries

- Arguments are permutable w.r.t. permutation (ITEMS, TABLE).
- Items of ITEMS are permutable.
- Items of TABLE are permutable.
- All occurrences of two distinct values in ITEMS.value or TABLE.value can be swapped; all occurrences of a value in ITEMS.value or TABLE.value can be renamed to any unused value.

Arg. properties

Functional dependency: ITEMS.value determined by ITEMS.index and TABLE.

Usage

Used for replacing by one single \text{elements\_alldifferent} constraint an \text{alldifferent} and a set of \text{element} constraints having the following structure:

- The union of the index variables of the \text{element} constraints is equal to the set of variables of the \text{alldifferent} constraint.
- All the \text{element} constraints share exactly the same table.

For instance, the constraint given in the \text{Example} slot is equivalent to the conjunction of the following set of constraints:

\[
\text{alldifferent}((\var - 2, \var - 1, \var - 4, \var - 3))
\]

\[
\begin{align*}
\text{element} & \quad (\langle \text{index} - 2, \text{value} - 9 \rangle, \\
& \quad \langle \text{index} - 1, \text{value} - 6 \rangle, \\
& \quad \langle \text{index} - 2, \text{value} - 9 \rangle, \\
& \quad \langle \text{index} - 3, \text{value} - 2 \rangle, \\
& \quad \langle \text{index} - 4, \text{value} - 9 \rangle) \\
\text{element} & \quad (\langle \text{index} - 1, \text{value} - 6 \rangle, \\
& \quad \langle \text{index} - 1, \text{value} - 6 \rangle, \\
& \quad \langle \text{index} - 2, \text{value} - 9 \rangle, \\
& \quad \langle \text{index} - 3, \text{value} - 2 \rangle, \\
& \quad \langle \text{index} - 4, \text{value} - 9 \rangle) \\
\text{element} & \quad (\langle \text{index} - 3, \text{value} - 2 \rangle, \\
& \quad \langle \text{index} - 1, \text{value} - 6 \rangle, \\
& \quad \langle \text{index} - 2, \text{value} - 9 \rangle, \\
& \quad \langle \text{index} - 3, \text{value} - 2 \rangle, \\
& \quad \langle \text{index} - 4, \text{value} - 9 \rangle) \\
\text{element} & \quad (\langle \text{index} - 4, \text{value} - 9 \rangle, \\
& \quad \langle \text{index} - 1, \text{value} - 6 \rangle, \\
& \quad \langle \text{index} - 2, \text{value} - 9 \rangle, \\
& \quad \langle \text{index} - 3, \text{value} - 2 \rangle, \\
& \quad \langle \text{index} - 4, \text{value} - 9 \rangle)
\end{align*}
\]
As a practical example of utilisation of the `elements_alldifferent` constraint we show how to model the link between a permutation consisting of one single cycle and its expanded form. For instance, to the permutation 3, 6, 5, 2, 4, 1 corresponds the sequence 3 5 4 2 6 1. Let us note $S_1, S_2, S_3, S_4, S_5, S_6$ the permutation and $V_1, V_2, V_3, V_4, V_5, V_6$ its expanded form (see Figure 5.264).

The constraint:

$$
\text{elements\_alldifferent} = \{\text{index} - V_1, \text{value} - V_2, \\
\text{index} - V_2, \text{value} - V_3, \\
\text{index} - V_3, \text{value} - V_4, \\
\text{index} - V_4, \text{value} - V_5, \\
\text{index} - V_5, \text{value} - V_6, \\
\text{index} - V_6, \text{value} - V_1, \\
\text{index} - 1, \text{value} - S_1, \\
\text{index} - 2, \text{value} - S_2, \\
\text{index} - 3, \text{value} - S_3, \\
\text{index} - 4, \text{value} - S_4, \\
\text{index} - 5, \text{value} - S_5, \\
\text{index} - 6, \text{value} - S_6\}$$

models the fact that $S_1, S_2, S_3, S_4, S_5, S_6$ corresponds to a permutation with one single cycle. It also expresses the link between the variables $S_1, S_2, S_3, S_4, S_5, S_6$ and $V_1, V_2, V_3, V_4, V_5, V_6$.

Figure 5.264: Two representations of a permutation containing one single cycle

Reformulation

The `elements\_alldifferent` constraint can be expressed in term of a conjunction of `ITEMS elem` constraints and of one `alldifferent` constraint of the form:

- \(\text{elem}(\text{index} - I_1, \text{value} - V_1, \text{TABLE})\),
- \(\text{elem}(\text{index} - I_2, \text{value} - V_2, \text{TABLE})\),

... 

- \(\text{elem}(\text{index} - I_{|\text{ITEMS}|}, \text{value} - V_{|\text{ITEMS}|}, \text{TABLE})\),
- \(\text{alldifferent}(I_1, I_2, ..., I_{|\text{ITEMS}|})\).

See also

- implies: elements.
- used in reformulation: alldifferent, elem, element.
Keywords

characteristic of a constraint: disequality.
combinatorial object: permutation.
constraint type: data constraint.
modelling: array constraint, table, functional dependency.
Arc input(s) | ITEMS TABLE  
---|---
Arc generator | $PRODUCT \rightarrow \text{collection}(\text{items, table})$  
Arc arity | 2  
Arc constraint(s) |  
| • $\text{items.index} = \text{table.index}$  
| • $\text{items.value} = \text{table.value}$  
Graph property(ies) | $\text{NVERTEX} = |\text{ITEMS}| + |\text{TABLE}|$  

Graph model
The fact that all variables $\text{ITEMS.index}$ are pairwise different is derived from the conjunctions of the following facts:

- From the graph property $\text{NVERTEX} = |\text{ITEMS}| + |\text{TABLE}|$ it follows that all vertices of the initial graph belong also to the final graph,
- A vertex $v$ belongs to the final graph if there is at least one constraint involving $v$ that holds,
- From the first condition $\text{items.index} = \text{table.index}$ of the arc constraint, and from the restriction $\text{distinct}(\text{TABLE.index})$ it follows: for all vertices $v$ generated from the collection $\text{ITEMS}$ at most one constraint involving $v$ holds.

Parts (A) and (B) of Figure 5.265 respectively show the initial and final graph associated with the Example slot. Since we use the $\text{NVERTEX}$ graph property, the vertices of the final graph are stressed in bold.

![Figure 5.265: Initial and final graph of the elements alldifferent constraint](image)

Signature
Since the final graph cannot have more than $|\text{ITEMS}| + |\text{TABLE}|$ vertices one can simplify $\text{NVERTEX}$ to $\text{NVERTEX}$.  


### 5.136 elements_sparse

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Derived from <code>element_sparse</code>.</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>elements_sparse(ITEMS, TABLE, DEFAULT)</td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>ITEMS : <code>collection(index-dvar,value-dvar)</code>&lt;br&gt;TABLE : <code>collection(index-int,value-int)</code>&lt;br&gt;DEFAULT : int</td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>required(ITEMS, [index, value])&lt;br&gt;ITEMS.index ≥ 1&lt;br&gt;required(TABLE, [index, value])&lt;br&gt;TABLE.index ≥ 1&lt;br&gt;distinct(TABLE, index)</td>
<td></td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>All the items of ITEMS should be equal to one of the entries of the table TABLE or to the default value DEFAULT if the entry ITEMS.index does not occurs among the values of the index attribute of the TABLE collection.</td>
<td></td>
</tr>
</tbody>
</table>

#### Example

\[
\begin{pmatrix}
\langle \text{index} - 8, \text{value} - 9, \rangle, \\
\langle \text{index} - 3, \text{value} - 5, \rangle, \\
\langle \text{index} - 2, \text{value} - 5, \rangle, \\
\langle \text{index} - 1, \text{value} - 6, \rangle, \\
\langle \text{index} - 2, \text{value} - 5, \rangle, \\
\langle \text{index} - 4, \text{value} - 2, \rangle, \\
\langle \text{index} - 8, \text{value} - 9 \rangle, \\
\end{pmatrix}
\]

The `elements_sparse` constraint holds since:

- The first and third items (items `(index - 8 value - 9)` and `(index - 2 value - 5)`) of its ITEMS collection respectively correspond to the fourth and second item of its TABLE collection.
- The index attribute of the second item of its ITEMS collection (i.e., value 3) does not correspond to any index of the TABLE collection. Therefore the value attribute of the second item of the ITEMS collection is set the the default value 5 given by the last argument of the `elements_sparse` constraint.

<table>
<thead>
<tr>
<th><strong>Typical</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Symmetries

- Items of ITEMS are permutable.
- Items of TABLE are permutable.
- All occurrences of two distinct values in ITEMS.value, TABLE.value or DEFAULT can be swapped; all occurrences of a value in ITEMS.value, TABLE.value or DEFAULT can be renamed to any unused value.

Usage

Used for replacing several element constraints sharing exactly the same sparse table by one single constraint.

Reformulation

Let \( I_k \) and \( V_k \) respectively denote ITEMS\([k]\).index and ITEMS\([k]\).value (\( k \in [1,|\text{ITEMS}|] \)). The elements_sparse(ITEMS, TABLE, DEFAULT) constraint can be expressed in term of \(|\text{ITEMS}|\) reified constraints of the form:

\[
((I_k = \text{TABLE}[1].\text{index} \land V_k = \text{TABLE}[1].\text{value}) \lor \\
(I_k = \text{TABLE}[2].\text{index} \land V_k = \text{TABLE}[2].\text{value}) \lor \\
\cdots \\
(I_k = \text{TABLE}[|\text{TABLE}|].\text{index} \land V_k = \text{TABLE}[|\text{TABLE}|].\text{value})) \lor \\
((I_k \neq \text{TABLE}[1].\text{index}) \land \\
(I_k \neq \text{TABLE}[2].\text{index}) \land \\
\cdots \\
(I_k \neq \text{TABLE}[|\text{TABLE}|].\text{index}) \land \\
(V_k = \text{DEFAULT})).
\]

See also

common keyword: elem, element (data constraint), element_sparse (sparse table).

implied by: element_sparse.

part of system of constraints: element_sparse.

Keywords

characteristic of a constraint: derived collection.

constraint type: data constraint, system of constraints.

filtering: arc-consistency.

modelling: table, shared table, sparse table, sparse functional dependency.
Derived Collections

\[
\text{col} \left( \text{DEF}\text{-}\text{collection}(\text{index}\text{-}\text{int},\text{value}\text{-}\text{int}), \right.
\]
\[
\left. \text{item}(\text{index}\text{-}0,\text{value}\text{-}\text{DEFAULT}) \right)
\]
\[
\text{col} \left( \text{TABLE}\text{-}\text{DEF}\text{-}\text{collection}(\text{index}\text{-}\text{dvar},\text{value}\text{-}\text{dvar}), \right.
\]
\[
\left. \text{item}(\text{index}\text{-}\text{TABLE}\text{.index},\text{value}\text{-}\text{TABLE}\text{.index}), \right)
\]
\[
\left. \text{item}(\text{index}\text{-}\text{DEF}\text{.index},\text{value}\text{-}\text{DEF}\text{.value}) \right)
\]

Arc input(s) ITEMS TABLE_DEF
Arc generator \( PRODUCT \rightarrow \text{collection}(\text{items,table_def}) \)
Arc arity 2
Arc constraint(s)
- items.value = table_def.value
- items.index = table_def.index \(\lor\) table_def.index = 0

Graph property(ies) \( \text{NSOURCE} = |\text{ITEMS}| \)

Graph model

An item of the ITEMS collection may have up to two successors (see for instance the third item of the ITEMS collection of the Example slot). Therefore we use the graph property \( \text{NSOURCE} = |\text{ITEMS}| \) for enforcing the fact that each item of the ITEMS collection has at least one successor.

Parts (A) and (B) of Figure 5.266 respectively show the initial and final graph associated with the Example slot. Since we use the NSOURCE graph property, the vertices of the final graph are drawn with a double circle.

![Graph Model](image)

Figure 5.266: Initial and final graph of the elements sparse constraint

Signature

On the one hand note that ITEMS is equal to the number of sources of the initial graph. On the other hand note that, in the initial graph, all the vertices that are not sources correspond to sinks. Since isolated vertices are eliminated from the final graph the sinks of the initial graph cannot become sources of the final graph. Therefore the maximum number of sources of the final graph is equal to ITEMS. We can rewrite \( \text{NSOURCE} = |\text{ITEMS}| \) to \( \text{NSOURCE} \geq |\text{ITEMS}| \) and simplify \( \text{NSOURCE} \) to \( \text{NSOURCE} \).
5.137  eq

DESCRIPTION  LINKS

Origin  Arithmetic.

Constraint  eq(VAR1, VAR2)

Synonym  xeqy.

Arguments  
VAR1 : dvar
VAR2 : dvar

Restriction

Purpose  Enforce the fact that two variables are equal.

Example  (8, 8)

The eq constraint holds since 8 is equal to 8.

Symmetries

• Arguments are permutable w.r.t. permutation (VAR1, VAR2).
• All occurrences of a value in VAR1 or VAR2 can be renamed to any unused value.

Arg. properties

• Functional dependency: VAR2 determined by VAR1.
• Functional dependency: VAR1 determined by VAR2.

Systems  eq in Choco, rel in Gecode, xeqy in JaCoP, #= in SICStus.

See also  common keyword: gt, lt (binary constraint, arithmetic constraint).
          generalisation: all_equal (equality between more than two variables),
          eq_cst (constant added), eq_set (variable replaced by set variable).
          implies: abs_value, geq, leq, same_sign.
          negation: neq.

Keywords  constraint arguments: binary constraint, pure functional dependency.
          constraint type: predefined constraint, arithmetic constraint.
          filtering: arc-consistency.
          modelling: functional dependency.
## 5.138 eq_cst

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Arithmetic.</td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td><code>eq_cst(VAR1, VAR2, CST2)</code></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>VAR1 : dvar, VAR2 : dvar, CST2 : int</td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>Enforce the fact that the first variable is equal to the sum of the second variable and the constant.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>(8, 2, 6)</td>
</tr>
<tr>
<td><strong>Typical</strong></td>
<td>CST2 ≠ 0</td>
</tr>
<tr>
<td><strong>Symmetries</strong></td>
<td>- Arguments are permutable w.r.t. permutation (VAR1) (VAR2, CST2). - One and the same constant can be added to VAR1 and VAR2. - One and the same constant can be added to VAR1 and CST2.</td>
</tr>
<tr>
<td><strong>Arg. properties</strong></td>
<td>- Functional dependency: VAR1 determined by VAR2 and CST2. - Functional dependency: VAR2 determined by VAR1 and CST2. - Functional dependency: CST2 determined by VAR1 and VAR2.</td>
</tr>
<tr>
<td><strong>See also</strong></td>
<td>implies: geq_cst, leq_cst. negation: neq_cst. specialisation: eq(constant set to 0).</td>
</tr>
<tr>
<td><strong>Keywords</strong></td>
<td>constraint arguments: binary constraint, pure functional dependency. constraint type: predefined constraint, arithmetic constraint. filtering: arc-consistency. modelling: functional dependency.</td>
</tr>
</tbody>
</table>
### 5.139 eq_set

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Used for defining <code>alldifferent_between_sets</code>.</td>
</tr>
<tr>
<td>Constraint</td>
<td><code>eq_set(SET1, SET2)</code></td>
</tr>
</tbody>
</table>
| Arguments   | SET1 : `svar`  
SET2 : `svar` |
| Purpose     | Constraint the set `SET1` to be equal to the set `SET2`. |
| Example     | `({3, 5}, {3, 5})` |

**Symmetries**
- Arguments are `permutable` w.r.t. permutation (SET1, SET2).
- All occurrences of a value in SET1 or SET2 can be `renamed` to any unused value.

**Systems**
- `eq` in Choco, `rel` in Gecode.

**Used in**
- `alldifferent_between_sets`.

**See also**
- `specialisation`: `eq(set variable replaced by variable)`.

**Keywords**
- `characteristic of a constraint`: equality.
- `constraint arguments`: binary constraint, constraint involving set variables.
- `constraint type`: predefined constraint.
### 5.140 equal_sboxes

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>LOGIC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometry, derived from [318]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>equal_sboxes(K, DIMS, OBJECTS, SBOXES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Synonym</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>equal.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Types</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARIABLES : collection(v−dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTEGERS : collection(v−int)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POSITIVES : collection(v−int)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K : int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIMS : sint</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OBJECTS : collection(oid−int, sid−int, x − VARIABLES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBOXES : collection(sid−int, t − INTEGERS, l − POSITIVES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES</td>
<td>≥ 1</td>
</tr>
<tr>
<td></td>
<td>INTEGERS</td>
<td>≥ 1</td>
</tr>
<tr>
<td></td>
<td>POSITIVES</td>
<td>≥ 1</td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES, v)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES</td>
<td>= K</td>
</tr>
<tr>
<td></td>
<td>required(INTEGERS, v)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTEGERS</td>
<td>= K</td>
</tr>
<tr>
<td></td>
<td>required(POSITIVES, v)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>POSITIVES</td>
<td>= K</td>
</tr>
<tr>
<td></td>
<td>POSITIVES.v &gt; 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>K &gt; 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DIMS ≥ 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DIMS &lt; K</td>
<td></td>
</tr>
<tr>
<td></td>
<td>increasing_seq(OBJECTS, [oid])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>required(OBJECTS, [oid, sid, x])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OBJECTS.oid ≥ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OBJECTS.oid ≤</td>
<td>OBJECTS</td>
</tr>
<tr>
<td></td>
<td>OBJECTS.sid ≥ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OBJECTS.sid ≤</td>
<td>SBOXES</td>
</tr>
<tr>
<td></td>
<td>SBOXES</td>
<td>≥ 1</td>
</tr>
<tr>
<td></td>
<td>required(SBOXES, [sid, t, l])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SBOXES.sid ≥ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SBOXES.sid ≤</td>
<td>SBOXES</td>
</tr>
<tr>
<td></td>
<td>do_not_overlap(SBOXES)</td>
<td></td>
</tr>
</tbody>
</table>
Holds if, for each pair of objects \((O_i, O_j)\), \(i \neq j\), \(O_i\) and \(O_j\) coincide exactly with respect to a set of dimensions depicted by \(\text{DIMS}\). \(O_i\) and \(O_j\) are objects that take a shape among a set of shapes. Each shape is defined as a finite set of shifted boxes, where each shifted box is described by a box in a \(K\)-dimensional space at a given offset (from the origin of the shape) with given sizes. More precisely, a shifted box is an entity defined by its shape id \(\text{sid}\), shift offset \(t\), and sizes \(l\). Then, a shape is defined as the union of shifted boxes sharing the same shape id. An object is an entity defined by its unique object identifier \(\text{oid}\), shape id \(\text{sid}\) and origin \(x\).

Two objects \(O_i\) and object \(O_j\) are equal with respect to a set of dimensions depicted by \(\text{DIMS}\) if and only if, for all shifted box \(s_i\) associated with \(O_i\) there exists a shifted box \(s_j\) such that, for all dimensions \(d \in \text{DIMS}\), (1) the origins of \(s_i\) and \(s_j\) coincide and, (2) the ends of \(s_i\) and \(s_j\) also coincide.

Figure 5.267 shows the objects of the example. Since these objects coincide exactly the \text{equal_sboxes} constraint holds.

Figure 5.267: The three mutually coinciding objects of the example

Typical

\(|\text{OBJECTS}| > 1\)
Symmetries

- Items of OBJECTS are permutable.
- Items of SBOXES are permutable.
- Items of OBJECTS.x, SBOXES.t and SBOXES.l are permutable (same permutation used).

Arg. properties

Suffix-contractible wrt. OBJECTS.

Remark

One of the eight relations of the Region Connection Calculus [318]. The constraint equal_sboxes is a restriction of the original relation since it requires to have exactly the same partition between the different objects.

See also

common keyword: contains_sboxes, coveredby_sboxes, covers_sboxes, disjoint_sboxes, inside_sboxes, meet_sboxes(rcc8), non_overlap_sboxes(geometrical constraint,logic), overlap_sboxes(rcc8).

Keywords

constraint type: logic.

geometry: geometrical constraint, rcc8.
Logic

- $\text{origin}(O_1, S_1, D) \overset{\text{def}}{=} O_1.x(D) + S_1.t(D)$
- $\text{end}(O_1, S_1, D) \overset{\text{def}}{=} O_1.x(D) + S_1.t(D) + S_1.1(D)$
- $\text{equal\_sboxes}(\text{Dims}, O_1, S_1, O_2, S_2) \overset{\text{def}}{=} \forall D \in \text{Dims}
  \left( \begin{array}{l}
  \text{origin}(O_1, S_1, D) = \\
  \text{origin}(O_2, S_2, D) \\
  \text{end}(O_1, S_1, D) = \\
  \text{end}(O_2, S_2, D)
  \end{array} \right)
- $\text{equal\_objects}(\text{Dims}, O_1, O_2) \overset{\text{def}}{=} \forall S_1 \in \text{sboxes}([O_1.\text{sid}])
  \exists S_2 \in \text{sboxes}( [02.\text{sid}])
  \text{equal\_sboxes}( \text{Dims}, O_1, S_1, O_2, S_2)
- $\text{all\_equal}(\text{Dims}, \text{OIDS}) \overset{\text{def}}{=} \forall O_1 \in \text{objects(\text{OIDS})}
  \forall O_2 \in \text{objects(\text{OIDS})}
  O_1.\text{oid} = \Rightarrow
  O_2.\text{oid} = 1
- $\text{all\_equal}(\text{DIMENSIONS}, \text{OIDS})$
### 5.141 equivalent

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Logic</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>equivalent(VAR, VARIABLES)</td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td>eq.</td>
<td></td>
</tr>
</tbody>
</table>
| Arguments   | VAR : dvar  
VARIABLES : collection(var–dvar) |           |
| Restrictions| VAR ≥ 0  
VAR ≤ 1  
| Variables| | |
| Purpose     | Let VARIABLES be a collection of 0-1 variables VAR₁, VAR₂. Enforce VAR = (VAR₁ ⇔ VAR₂). | |
| Example     | (1, (0, 0))  
(0, (0, 1))  
(0, (1, 0))  
(1, (1, 1)) | |
| Symmetries  | • Items of VARIABLES are permutable.  
• All occurrences of 0 in VAR and in VARIABLES.var can be set to 1. | |
| Arg. properties | Functional dependency: VAR determined by VARIABLES. | |
| Systems     | ifOnlyIf in Choco, rel in Gecode, eqbool in JaCoP, #<=> in SICStus. | |
| See also    | common keyword: and, imply, nand, nor, or, xor (Boolean constraint). | |
| Keywords    | characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.  
constraint arguments: pure functional dependency.  
constraint network structure: Berge-acyclic constraint network.  
constraint type: Boolean constraint.  
filtering: arc-consistency.  
modelling: functional dependency. | |
Automaton

Figure 5.268 depicts the automaton associated with the equivalent constraint. To the first argument \( \text{VAR} \) of the equivalent constraint corresponds the first signature variable. To each variable \( \text{VAR}_i \) of the second argument \( \text{VARIABLES} \) of the equivalent constraint corresponds the next signature variable. There is no signature constraint.

![Automaton Diagram](image)

**Figure 5.268: Automaton of the equivalent constraint**

![Hypergraph Diagram](image)

**Figure 5.269: Hypergraph of the reformulation corresponding to the automaton of the equivalent constraint**
5.142 exactly

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from at least and at most.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>exactly(N, VARIABLES, VALUE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td>count.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>N : int</td>
<td>VARIABLES : collection(var–dvar)</td>
<td>VALUE : int</td>
</tr>
<tr>
<td>Restrictions</td>
<td>N ≥ 0</td>
<td>N ≤</td>
<td>VARIABLES</td>
</tr>
<tr>
<td>Purpose</td>
<td>Exactly N variables of the VARIABLES collection are assigned value VALUE.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>(2, ⟨4, 2, 4, 5⟩, 4)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The exactly constraint holds since exactly \( N = 2 \) variables of the VARIABLES = \( ⟨4, 2, 4, 5⟩ \) collection are assigned value VALUE = 4.

Typical

\[
\begin{align*}
N & > 0 \\
N & < |VARIABLES| \\
|VARIABLES| & > 1
\end{align*}
\]

Symmetries

- Items of VARIABLES are permutable.
- An occurrence of a value of VARIABLES.var that is different from VALUE can be replaced by any other value that is also different from VALUE.

Arg. properties

- Functional dependency: \( N \) determined by VARIABLES and VALUE.
- Aggregate: \( N(+)\), VARIABLES(union), VALUE(id).

Systems

occurrence in Choco, count in Gecode, exactly in Gecode, count in JaCoP, exactly in MiniZinc, count in SICStus.

See also

generalisation: among (constant replaced by variable and value replaced by list of values).
implies: at least (= \( N \) replaced by \( \geq N \)), at most (= \( N \) replaced by \( \leq N \)).
Keywords

characteristic of a constraint: automaton, automaton with counters.
constraint arguments: pure functional dependency.
constraint network structure: alpha-acyclic constraint network(2).
constraint type: value constraint, counting constraint.
filtering: arc-consistency.
modelling: functional dependency.
Graph model

Since each arc constraint involves only one vertex (VALUE is fixed), we employ the SELF arc generator in order to produce a graph with a single loop on each vertex.

Parts (A) and (B) of Figure 5.270 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the loops of the final graph are stressed in bold. The exactly constraint holds since exactly two variables are assigned value 4.

![Graph](image-url)

Figure 5.270: Initial and final graph of the exactly constraint
Automaton

Figure 5.271 depicts the automaton associated with the exactly constraint. To each variable $\text{VAR}_i$ of the collection $\text{VARIABLES}$ corresponds a 0-1 signature variable $S_i$. The following signature constraint links $\text{VAR}_i$ and $S_i$: $\text{VAR}_i = \text{VALUE} \Leftrightarrow S_i$.

![Automaton of the exactly constraint](image1)

Figure 5.271: Automaton of the exactly constraint

![Hypergraph of the reformulation corresponding to the automaton of the exactly constraint](image2)

Figure 5.272: Hypergraph of the reformulation corresponding to the automaton of the exactly constraint
## 5.143 gcd

<table>
<thead>
<tr>
<th>Description</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[126]</td>
</tr>
<tr>
<td>Constraint</td>
<td>( \gcd(X, Y, Z) )</td>
</tr>
<tr>
<td>Arguments</td>
<td>( X : \text{dvar} ) ( Y : \text{dvar} ) ( Z : \text{dvar} )</td>
</tr>
<tr>
<td>Restrictions</td>
<td>( X &gt; 0 ) ( Y &gt; 0 ) ( Z &gt; 0 )</td>
</tr>
<tr>
<td>Purpose</td>
<td>Enforce the fact that ( Z ) is the greatest common divisor of ( X ) and ( Y ).</td>
</tr>
<tr>
<td>Example</td>
<td>( (24, 60, 12) )</td>
</tr>
</tbody>
</table>

The \( \gcd \) constraint holds since 12 is the greatest common divisor of 24 and 60.

<table>
<thead>
<tr>
<th>Typical</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( X &gt; 1 ) ( Y &gt; 1 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symmetry</th>
<th>Arguments are permutable w.r.t. permutation ((X, Y) (Z)).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arg. properties</td>
<td>Functional dependency: ( X ) determined by ( Y ) and ( Z ).</td>
</tr>
<tr>
<td>Algorithm</td>
<td>In [126] a filtering algorithm for the ( \gcd ) constraint was automatically derived from the Euclidian algorithm by using constructive disjunction and abstract interpretation in order to approximate the behaviour of the while loop of the Euclidian algorithm.</td>
</tr>
<tr>
<td>See also</td>
<td>common keyword: power (abstract interpretation).</td>
</tr>
</tbody>
</table>
5.144  geost

**DESCRIPTION**

Origin  Generalisation of *diffn*.

**Constraint**  \( \text{geost}(K, \text{OBJECTS}, \text{SBOXES}) \)

**Types**

- **VARIABLES** : \( \text{collection}(v – \text{dvar}) \)
- **INTEGERS** : \( \text{collection}(v – \text{int}) \)
- **POSITIVES** : \( \text{collection}(v – \text{int}) \)

**Arguments**

- \( K \) : \( \text{int} \)
- \( \text{OBJECTS} \) : \( \text{collection}(\text{oid} – \text{int}, \text{sid} – \text{dvar}, x – \text{VARIABLES}) \)
- \( \text{SBOXES} \) : \( \text{collection}(\text{sid} – \text{int}, t – \text{INTEGERS}, l – \text{POSITIVES}) \)

**Restrictions**

\[
|\text{VARIABLES}| \geq 1 \\
|\text{INTEGERS}| \geq 1 \\
|\text{POSITIVES}| \geq 1 \\
\text{required}(\text{VARIABLES}, v) \\
|\text{VARIABLES}| = K \\
\text{required}(\text{INTEGERS}, v) \\
|\text{INTEGERS}| = K \\
\text{required}(\text{POSITIVES}, v) \\
|\text{POSITIVES}| = K \\
\text{POSITIVES}.v > 0 \\
K > 0 \\
\text{required}(\text{OBJECTS}, [\text{oid}, \text{sid}, x]) \\
\text{distinct}(\text{OBJECTS}, \text{oid}) \\
\text{OBJECTS}.\text{oid} \geq 1 \\
\text{OBJECTS}.\text{oid} \leq |\text{OBJECTS}| \\
\text{OBJECTS}.\text{sid} \geq 1 \\
\text{OBJECTS}.\text{sid} \leq |\text{SBOXES}| \\
|\text{SBOXES}| \geq 1 \\
\text{required}(\text{SBOXES}, [\text{sid}, t, l]) \\
\text{SBOXES}.\text{sid} \geq 1 \\
\text{SBOXES}.\text{sid} \leq |\text{SBOXES}| \\
\text{do not overlap}(\text{SBOXES})
\]
Holds if, for each pair of objects \((O_i, O_j), i < j\), \(O_i\) and \(O_j\) do not overlap with respect to a set of dimensions \(\{1, 2, \ldots, K\}\). \(O_i\) and \(O_j\) are objects that take a shape among a set of shapes. Each shape is defined as a finite set of shifted boxes, where each shifted box is described by a box in a \(K\)-dimensional space at a given offset (from the origin of the shape) with given sizes. More precisely, a shifted box is an entity defined by its shape id \(\text{sid}\), shift offset \(\text{x}\), and sizes \(\text{id}\). Then, a shape is defined as the union of shifted boxes sharing the same shape id. An object is an entity defined by its unique object identifier \(\text{oid}\), shape id \(\text{sid}\) and origin \(\text{x}\).

An object \(O_i\) does not overlap an object \(O_j\) with respect to the set of dimensions \(\{1, 2, \ldots, K\}\) if and only if for all shifted box \(s_i\) associated with \(O_i\) and for all shifted box \(s_j\) associated with \(O_j\) there exists a dimension \(d \in \{1, 2, \ldots, K\}\) such that the start of \(s_i\) in dimension \(d\) is greater than or equal to the end of \(s_j\) in dimension \(d\), or the start of \(s_j\) in dimension \(d\) is greater than or equal to the end of \(s_i\) in dimension \(d\).

### Example

<table>
<thead>
<tr>
<th>(\text{oid} - 1)</th>
<th>(\text{sid} - 1)</th>
<th>(\text{x} - (1, 2))</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,</td>
<td>(\text{oid} - 2)</td>
<td>(\text{sid} - 5)</td>
</tr>
<tr>
<td>(\text{oid} - 3)</td>
<td>(\text{sid} - 8)</td>
<td>(\text{x} - (4, 1))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\text{sid} - 1)</th>
<th>(\text{t} - (0, 0))</th>
<th>(\text{l} - (2, 1))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{sid} - 1)</td>
<td>(\text{t} - (0, 1))</td>
<td>(\text{l} - (1, 2))</td>
</tr>
<tr>
<td>(\text{sid} - 1)</td>
<td>(\text{t} - (1, 2))</td>
<td>(\text{l} - (3, 1))</td>
</tr>
<tr>
<td>(\text{sid} - 2)</td>
<td>(\text{t} - (0, 0))</td>
<td>(\text{l} - (3, 1))</td>
</tr>
<tr>
<td>(\text{sid} - 2)</td>
<td>(\text{t} - (0, 1))</td>
<td>(\text{l} - (1, 3))</td>
</tr>
<tr>
<td>(\text{sid} - 2)</td>
<td>(\text{t} - (2, 1))</td>
<td>(\text{l} - (1, 1))</td>
</tr>
<tr>
<td>(\text{sid} - 3)</td>
<td>(\text{t} - (0, 0))</td>
<td>(\text{l} - (2, 1))</td>
</tr>
<tr>
<td>(\text{sid} - 3)</td>
<td>(\text{t} - (1, 1))</td>
<td>(\text{l} - (1, 2))</td>
</tr>
<tr>
<td>(\text{sid} - 3)</td>
<td>(\text{t} - (-2, 2))</td>
<td>(\text{l} - (3, 1))</td>
</tr>
<tr>
<td>(\text{sid} - 4)</td>
<td>(\text{t} - (0, 0))</td>
<td>(\text{l} - (3, 1))</td>
</tr>
<tr>
<td>(\text{sid} - 4)</td>
<td>(\text{t} - (0, 1))</td>
<td>(\text{l} - (1, 1))</td>
</tr>
<tr>
<td>(\text{sid} - 4)</td>
<td>(\text{t} - (2, 1))</td>
<td>(\text{l} - (1, 3))</td>
</tr>
<tr>
<td>(\text{sid} - 5)</td>
<td>(\text{t} - (0, 0))</td>
<td>(\text{l} - (2, 1))</td>
</tr>
<tr>
<td>(\text{sid} - 5)</td>
<td>(\text{t} - (1, 1))</td>
<td>(\text{l} - (1, 1))</td>
</tr>
<tr>
<td>(\text{sid} - 5)</td>
<td>(\text{t} - (0, 2))</td>
<td>(\text{l} - (2, 1))</td>
</tr>
<tr>
<td>(\text{sid} - 6)</td>
<td>(\text{t} - (0, 0))</td>
<td>(\text{l} - (3, 1))</td>
</tr>
<tr>
<td>(\text{sid} - 6)</td>
<td>(\text{t} - (0, 1))</td>
<td>(\text{l} - (1, 1))</td>
</tr>
<tr>
<td>(\text{sid} - 6)</td>
<td>(\text{t} - (2, 1))</td>
<td>(\text{l} - (1, 1))</td>
</tr>
<tr>
<td>(\text{sid} - 7)</td>
<td>(\text{t} - (0, 0))</td>
<td>(\text{l} - (3, 2))</td>
</tr>
<tr>
<td>(\text{sid} - 8)</td>
<td>(\text{t} - (0, 0))</td>
<td>(\text{l} - (2, 3))</td>
</tr>
</tbody>
</table>

Parts (A), (B) and (C) of Figure 5.273 respectively represent the potential shapes associated with the three objects of the example. Part (D) shows the position of the three objects of the example, where the first, second and third objects were respectively assigned shapes 1, 5 and 8. The coordinates of the leftmost lowest corner of each object are stressed in bold. The constraint holds since the three objects do not overlap (i.e., see part (D) if Figure 5.273).

### Typical

\(|\text{OBJECTS}| > 1\)
A possible placement where object 1 is assigned shape S1 and object 2 is assigned shape S5 and object 3 is assigned shape S8

Figure 5.273: The three objects of the example
Symmetries
• Items of OBJECTS are permutable.
• Items of SBOXES are permutable.
• Items of OBJECTS.x, SBOXES.t and SBOXES.l are permutable (same permutation used).
• SBOXES.l.v can be decreased to any value $\geq 1$.

Usage
The geost constraint allows to model directly a large number of placement problems.

Remark
In the two-dimensional case, when rectangles heights are all equal to one and when rectangles starts in the first dimension are all fixed, the geost constraint can be rewritten as a $k_{\text{alldifferent}}$ constraint corresponding to a system of $\text{alldifferent}$ constraints derived from the maximum cliques of the corresponding interval graph.

Algorithm
A sweep-based filtering algorithm for this constraint is described in [36]. Unlike previous sweep filtering algorithms which move a line for finding a feasible position for the origin of an object, this algorithm performs a recursive traversal of the multidimensional placement space. It explores all points of the domain of the origin of the object under focus, one by one, in increasing lexicographic order, until a point is found that is not infeasible for any non-overlapping constraints. To make the search efficient, instead of moving each time to the successor point, the search is arranged so that it skips points that are known to be infeasible for some non-overlapping constraint.

Within the context of breaking symmetries six different ways of integrating within geost a chain of lexicographical ordering constraints like $\text{lex\_chain\_less}$ for enforcing a lexicographic ordering on the origin coordinates of identical objects, are described in [2].

Systems
geost in Choco, geost in JaCoP, geost in SICStus.

See also
common keyword: calendar (multi-site employee scheduling with calendar constraints, scheduling with machine choice, calendars and preemption),
diffn (geometrical constraint, non-overlapping),
lex\_chain\_less, lex\_chain\_lesseq (symmetry),
non\_overlap\_sboxes (geometrical constraint, non-overlapping),
visible (geometrical constraint, sweep).

generalisation: geost\_time (temporal dimension added to geometrical dimensions).
specialisation: $k_{\text{alldifferent}}$ (when rectangles heights are all equal to 1 and rectangles starts in the first dimension are all fixed), $\text{lex\_alldifferent}$ (object replaced by vector).

Keywords
application area: floor planning problem.
combinatorial object: pentomino.
constraint arguments: business rules.
constraint type: logic, decomposition, timetabling constraint, predefined constraint, relaxation.
filtering: sweep.
geometry: geometrical constraint, non-overlapping.
heuristics: heuristics for two-dimensional rectangle placement problems.

modelling: multi-site employee scheduling with calendar constraints, scheduling with machine choice, calendars and preemption, disjunction, assignment dimension, assigning and scheduling tasks that run in parallel, assignment to the same set of values, relaxation dimension.

modelling exercises: multi-site employee scheduling with calendar constraints, scheduling with machine choice, calendars and preemption, assigning and scheduling tasks that run in parallel, assignment to the same set of values, relaxation dimension.

problems: strip packing, two-dimensional orthogonal packing, pallet loading.

puzzles: squared squares, packing almost squares, Partridge, pentomino, Shikaku, smallest square for packing consecutive dominoes, smallest square for packing rectangles with distinct sizes, smallest rectangle area, Conway packing problem.

symmetry: symmetry.
### 5.145 geost_time

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Generalisation of diffn.</td>
</tr>
<tr>
<td>Constraint</td>
<td>geost_time(K, DIMS, OBJECTS, SBOXES)</td>
</tr>
<tr>
<td>Types</td>
<td>VARIABLES : collection(v→dvar)</td>
</tr>
<tr>
<td></td>
<td>INTEGERS : collection(v→int)</td>
</tr>
<tr>
<td></td>
<td>POSITIVES : collection(v→int)</td>
</tr>
<tr>
<td>Arguments</td>
<td>K : int</td>
</tr>
<tr>
<td></td>
<td>DIMS : sint</td>
</tr>
<tr>
<td></td>
<td>OBJECTS : collection(oid→int, sid→dvar, x→VARIABLES, start→dvar, duration→dvar, end→dvar)</td>
</tr>
<tr>
<td></td>
<td>SBOXES : collection(sid→int, t→INTEGERS, l→POSITIVES)</td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
</tr>
</tbody>
</table>
Holds if (1) the difference between the end in time and the start in time of each object is equal to its duration in time, and if (2) for each pair of objects \((O_i, O_j), i < j\), \(O_i\) and \(O_j\) do not overlap with respect to a set of dimensions depicted by DIMS as well as to the time axis. \(O_i\) and \(O_j\) are objects that take a shape among a set of shapes. Each \textit{shape} is defined as a finite set of shifted boxes, where each shifted box is described by a box in a \(K\)-dimensional space at a given offset (from the origin of the shape) with given sizes. More precisely, a \textit{shifted box} is an entity defined by its shape id \(\text{id}\), shift offset \(t\), and sizes \(1\). Then, a shape is defined as the union of shifted boxes sharing the same shape id. An \textit{object} is an entity defined by its unique object identifier \(\text{id}\), shape id \(\text{id}\) and origin \(x\).

An object \(O_i\) \textit{does not overlap} an object \(O_j\) with respect to a set of dimensions depicted by DIMS as well as to the time axis if and only if:

- The start in time of \(O_i\) is greater than or equal to the end in time of \(O_j\).
- The start in time of \(O_j\) is greater than or equal to the end in time of \(O_i\).
- For all shifted box \(s_i\) associated with \(O_i\) and for all shifted box \(s_j\) associated with \(O_j\) there exists a dimension \(d \in \text{DIMS}\) such that the start of \(s_i\) in dimension \(d\) is greater than or equal to the end of \(s_j\) in dimension \(d\), or the start of \(s_j\) in dimension \(d\) is greater than or equal to the end of \(s_i\) in dimension \(d\).

\[
\begin{pmatrix}
2, \{0, 1\}, \\
oid - 1 & \text{id} - 1 & x - (1, 2) & \text{start} - 0 & \text{duration} - 1 & \text{end} - 1, \\
oid - 2 & \text{id} - 5 & x - (2, 1) & \text{start} - 0 & \text{duration} - 1 & \text{end} - 1, \\
oid - 3 & \text{id} - 8 & x - (4, 1) & \text{start} - 0 & \text{duration} - 1 & \text{end} - 1 \\
\text{id} - 1 & t - (0, 0) & 1 - (2, 1), \\
\text{id} - 1 & t - (0, 1) & 1 - (1, 2), \\
\text{id} - 1 & t - (1, 2) & 1 - (3, 1), \\
\text{id} - 1 & t - (1, 2) & 1 - (3, 1), \\
\text{id} - 2 & t - (0, 0) & 1 - (3, 1), \\
\text{id} - 2 & t - (0, 1) & 1 - (1, 3), \\
\text{id} - 2 & t - (2, 1) & 1 - (1, 1), \\
\text{id} - 3 & t - (0, 0) & 1 - (2, 1), \\
\text{id} - 3 & t - (1, 1) & 1 - (1, 2), \\
\text{id} - 3 & t - (2, 2) & 1 - (3, 1), \\
\text{id} - 4 & t - (0, 0) & 1 - (3, 1), \\
\text{id} - 4 & t - (0, 0) & 1 - (1, 1), \\
\text{id} - 4 & t - (2, 1) & 1 - (1, 1), \\
\text{id} - 5 & t - (0, 0) & 1 - (2, 1), \\
\text{id} - 5 & t - (1, 1) & 1 - (1, 1), \\
\text{id} - 5 & t - (0, 2) & 1 - (2, 1), \\
\text{id} - 6 & t - (0, 0) & 1 - (3, 1), \\
\text{id} - 6 & t - (0, 1) & 1 - (1, 1), \\
\text{id} - 6 & t - (2, 1) & 1 - (1, 1), \\
\text{id} - 7 & t - (0, 0) & 1 - (3, 2), \\
\text{id} - 8 & t - (0, 0) & 1 - (2, 3)
\end{pmatrix}
\]

Parts (A), (B) and (C) of Figure 5.274 respectively represent the potential shapes associated with the three objects of the example. Part (D) shows the position of the three objects of the example, where the first, second and third objects were respectively assigned shapes 1, 5 and 8. The coordinates of the leftmost lowest corner of each object are stressed.
in bold. The \texttt{geost\_time} constraint holds since the three objects do not overlap: even if the time intervals associated with each object overlap (i.e., they are in fact identical), their corresponding shapes do not overlap (i.e., see part (D) if Figure 5.274).

Figure 5.274: The three objects of the example

\begin{itemize}
\item Typical \hspace{1cm} $|\text{OBJECTS}| > 1$
\item Symmetries:
\begin{itemize}
\item Items of \texttt{OBJECTS} are \textit{permutable}.
\item Items of \texttt{SBOXES} are \textit{permutable}.
\item Items of \texttt{OBJECTS}.x, \texttt{SBOXES}.t and \texttt{SBOXES}.l are \textit{permutable} (same permutation used).
\item \texttt{SBOXES}.v can be \textit{decreased} to any value $\geq 1$.
\item One and the same constant can be \textit{added} to the \texttt{start} and \texttt{end} attributes of all items of \texttt{OBJECTS}.
\end{itemize}
\item Usage: The \texttt{geost\_time} constraint allows to model directly a large number of placement problems. Figure 5.275 sketches ten typical use of the \texttt{geost\_time} constraint:
\begin{itemize}
\item The first case (A) corresponds to a non-overlapping constraint among three segments.
\item The second, third and fourth cases (B,C,D) correspond to a non-overlapping constraint between rectangles where (B) and (C) are special cases where the sizes of all rectangles in the second dimension are equal to $1$; this can be interpreted as a \textit{machine assignment problem} where each rectangle corresponds to a non-pre-emptive
task that has to be placed in time and assigned to a specific machine so that no two
tasks assigned to the same machine overlap in time. In Part (B) the duration of each
task is fixed, while in Part (C) the duration depends on the machine to which the task
is actually assigned. This dependence can be expressed by the element constraint,
which specifies the dependence between the shape variable and the assignment vari-
able of each task.

- The fifth case (E) corresponds to a non-overlapping constraint between rectangles
where each rectangle can have two orientations. This is achieved by associating with
each rectangle two shapes of respective sizes $l \cdot h$ and $h \cdot l$. Since their orientation is
not initially fixed, an element_less_eq constraint can be used for enforcing the three
rectangles to be included within the bounding box defined by the origin's coordinates
1, 1 and sizes 8, 3.

- The sixth case (F) corresponds to a non-overlapping constraint between more com-
plex objects where each object is described by a given set of rectangles.

- The seventh case (G) describes a rectangle placement problem where one has to first
assign each rectangle to a strip so that all rectangles that are assigned to the same
strip do not overlap.

- The eighth case (H) corresponds to a non-overlapping constraint between parallel-
lepipeds.

- The ninth case (I) can be interpreted as a non-overlapping constraint between parallel-
lepipeds that are assigned to the same container. The first dimension corresponds
to the identifier of the container, while the next three dimensions are associated with
the position of a parallelepiped inside a container.

- Finally the tenth case (J) describes a rectangle placement problem over three con-
secutive time-slots: rectangles assigned to the same time-slot should not overlap in time.
We initially start with the three rectangles 1, 2 and 3. Rectangle 3 is no more present
at instant 2 (the arrow $\downarrow$ within rectangle 3 at time 1 indicates that rectangle 3 will
disappear at the next time-point), while rectangle 4 appears at instant 2 (the arrow $\uparrow$
within rectangle 4 at time 2 denotes the fact that the rectangle 4 appears at instant 2).
Finally rectangle 2 disappears at instant 3 and is replaced by rectangle 5.

Algorithm

A sweep-based filtering algorithm for this constraint is described in [36]. Unlike previous
sweep filtering algorithms which move a line for finding a feasible position for the origin of
an object, this algorithm performs a recursive traversal of the multidimensional placement
space. It explores all points of the domain of the origin of the object under focus, one by one,
in increasing lexicographic order, until a point is found that is not infeasible for any non-overlapping constraints. To make the search efficient, instead of moving each time
to the successor point, the search is arranged so that it skips points that are known to be
infeasible for some non-overlapping constraint.

Systems

geost in Choco, geost in JaCoP.

See also

common keyword: diffn, non_overlap_sboxes (geometrical constraint,non-overlapping),
visible (geometrical constraint,sweep).
specialisation: geost (temporal dimension removed).

Keywords

constraint type: decomposition, timetabling constraint, predefined constraint.
Figure 5.275: Ten typical examples of use of the `geost_time` constraint (ground instances)
filtering: sweep.
geometry: geometrical constraint, non-overlapping.
modelling: assignment dimension, assignment to the same set of values, assigning and scheduling tasks that run in parallel, disjunction.
modelling exercises: assignment to the same set of values, assigning and scheduling tasks that run in parallel.
## 5.146 geq

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Arithmetic.</td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>$\text{geq(VAR1, VAR2)}$</td>
</tr>
<tr>
<td><strong>Synonyms</strong></td>
<td>rel, xgteqy.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>VAR1 : dvar, VAR2 : dvar</td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>Enforce the fact that the first variable is greater than or equal to the second variable.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>(8, 1)</td>
</tr>
<tr>
<td>The geq constraint holds since 8 is greater than or equal to 1.</td>
<td></td>
</tr>
<tr>
<td><strong>Typical</strong></td>
<td>VAR1 &gt; VAR2</td>
</tr>
</tbody>
</table>
| **Symmetries** | • VAR1 can be replaced by any value $\geq$ VAR2.  
• VAR2 can be replaced by any value $\leq$ VAR1. |
| **Systems** | $\text{geq}$ in Choco, $\text{rel}$ in Gecode, $\text{xgteqy}$ in JaCoP, $\#=$ in SICStus. |
| **See also** |常见关键词: neq (binary constraint, arithmetic constraint).  
通用化: $\text{geq.cst}$ (constant added).  
由: abs.value, eq.gt, sign.of.  
无条件 (如果交换参数): leq.  
否定: lt. |
| **Keywords** | constraint arguments: binary constraint.  
constraint type: predefined constraint, arithmetic constraint.  
filtering: arc-consistency. |
# 5.147 `geq_cst`

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Arithmetic.</td>
</tr>
<tr>
<td>Constraint</td>
<td><code>geq_cst(VAR1, VAR2, CST2)</code></td>
</tr>
</tbody>
</table>
| Arguments   | `VAR1` : `dvar`  
             | `VAR2` : `dvar`  
             | `CST2` : `int`  |
| Purpose     | Enforce the fact that the first variable is greater than or equal to the sum of the second variable and the constant. |
| Example     | `(8, 1, 7)`  
             | The `geq_cst` constraint holds since 8 is greater than or equal to 1 + 7. |
| Typical     | `CST2 ≠ 0`  
             | `VAR1 > VAR2 + CST2` |
| Symmetries  | • Arguments are **permutable** w.r.t. permutation `(VAR1) (VAR2, CST2)`.  
             | • `VAR1` can be replaced by any value $\geq VAR2 + CST2$.  
             | • `VAR2` can be replaced by any value $\leq VAR1 − CST2$.  
             | • `CST2` can be replaced by any value $\leq VAR1 − VAR2$. |
| See also    | **common keyword**: `leq_cst` (*binary constraint, arithmetic constraint*).  
             | **implied by**: `eq_cst`.  
             | **specialisation**: `geq` (*constant set to 0*). |
| Keywords    | **constraint arguments**: binary constraint.  
             | **constraint type**: predefined constraint, arithmetic constraint.  
             | **filtering**: arc-consistency. |
5.148 global_cardinality

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>CHARME [278]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>\texttt{global.cardinality(VARIABLES, VALUES)}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>count, distribute, distribution, gcc, card_var_gcc, egcc, extended_global_cardinality.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>\texttt{VARIABLES : collection(var-dvar)} \texttt{VALUES : collection(val-int,noccurrence-dvar)}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(VARIABLES.var) \texttt{VALUES : required([VALUES.val,noccurrence])} distinct([VALUES.val]) VALUES.noccurrence ≥ 0 VALUES.noccurrence ≤</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>Each value VALUES[i].val (with ( i \in [1,</td>
<td>VALUES</td>
<td>] )) should be taken by exactly VALUES[i].noccurrence variables of the VARIABLES collection.</td>
</tr>
</tbody>
</table>
| Example     | \begin{pmatrix} 3, 3, 3, 8, 6, \vspace{0.5cm} \\
|             | \text{val} = 3 \text{noccurrence} = 2, \vspace{0.5cm} \\
|             | \text{val} = 5 \text{noccurrence} = 0, \vspace{0.5cm} \\
|             | \text{val} = 6 \text{noccurrence} = 1 \end{pmatrix} |       |           |
| Typical     | \( |\text{VARIABLES}| > 1 \) \texttt{range(VARIABLES.var)} > 1 \texttt{|VALUES|} > 1 \texttt{|VARIABLES| ≥ |VALUES|} \texttt{in_attr}(VARIABLES, var, VALUES, val) |       |           |
| Symmetries  | - Items of VARIABLES are permutable. \vspace{0.5cm}
|             | - Items of VALUES are permutable. \vspace{0.5cm}
|             | - An occurrence of a value of VARIABLES.var that does not belong to VALUES.val can be replaced by any other value that also does not belong to VALUES.val. \vspace{0.5cm}
|             | - All occurrences of two distinct values in VARIABLES.var or VALUES.val can be swapped; all occurrences of a value in VARIABLES.var or VALUES.val can be renamed to any unused value. |       |           |

The global_cardinality constraint holds since values 3, 5 and 6 respectively occur 2, 0 and 1 times within the collection (3, 3, 8, 6) and since no constraint was specified for value 8.
Arg. properties

- Functional dependency: VALUES.nocurrence determined by VARIABLES and VALUES.val.
- Contractible wrt. VALUES.

Usage

We show how to use the global_cardinality constraint in order to model the magic series problem [390, page 155] with one single global_cardinality constraint. A non-empty finite series $S = (s_0, s_1, \ldots, s_n)$ is magic if and only if there are $s_i$ occurrences of $i$ in $S$ for each integer $i$ ranging from 0 to $n$. This leads to the following model:

\[
\text{global_cardinality}
\begin{pmatrix}
\langle \text{var} - s_0, \text{var} - s_1, \ldots, \text{var} - s_n \rangle, \\
\text{val} - 0 \ noccurrence - s_0, \\
\text{val} - 1 \ noccurrence - s_1, \\
\vdots \\
\text{val} - n \ noccurrence - s_n
\end{pmatrix}
\]

Remark

This is a generalised form of the original global_cardinality constraint: in the original global_cardinality constraint [322], one specifies for each value its minimum and maximum number of occurrences (i.e., see global_cardinality_low_up). Here we give for each value $v$ a domain variable that indicates how many times value $v$ is effectively used. By setting the minimum and maximum values of this variable to the appropriate constants we can express the same thing as in the original global_cardinality constraint. However, as shown in the magic series problem, we can also use this variable in other constraints. By reduction from 3-SAT, Claude-Guy Quimper shows in [311] that it is NP-hard to achieve arc-consistency for the count variables.

A last difference with the original global_cardinality constraint comes from the fact that there is no constraint on the values that are not explicitly mentioned in the VALUES collection. In the original global_cardinality these values could not be assigned to the variables of the VARIABLES collection. However allowing values that are not mentioned in VALUES to be assigned to variables of VARIABLES can potentially avoid mentioning a huge number of unconstrained values in the VALUES collection, and as a side effect, prevent eventually\(^5\) generating a dense graph (i.e., see DFS-bottleneck) for the corresponding underlying flow model.

Within [78] the global_cardinality constraint is called distribution. Within [330] the global_cardinality constraint is called card_var_gcc. Within [66] the global_cardinality constraint is called egcc or rgcc. This later case corresponds to the fact that some variables are duplicated within the VARIABLES collection.

The global_cardinality constraint can be seen as a system (i.e., a conjunction) of among constraints.

When all count variables (i.e., the variables VALUES[i].nocurrence with $i \in [1,|\text{VALUES}|]$) do not occur in any other constraints of the problem, it may be operationally more efficient to replace the global_cardinality constraint by a global_cardinality_low_up constraint where each count variable VALUES[i].nocurrence is replaced by the corresponding interval [VALUES[i].noccurrence, VALUES[i].noccurrence]. This stands for two reasons:

\(^5\) Of course one could also, while generating a flow model, detect all unconstrained values in order to generate one single vertex in the flow model for the set of unconstrained values.
First, by using a `global_cardinality_low_up` constraint rather than a `global_cardinality` constraint, we avoid the filtering algorithm related to the count variables.

Second, unlike the `global_cardinality` constraint where we need to fix all its variables to get entailment, the `global_cardinality_low_up` constraint can be entailed before all its variables get fixed. As a result, this potentially avoid unnecessary calls to its filtering algorithm.

An implicit necessary condition inferred by double counting with the `global_cardinality` constraint is depicted by the following expression:

\[
\sum_{i=1}^{\text{VARIABLES}} \text{VARIABLES}[i].\text{var} = \sum_{i=1}^{\text{VALUES}} \text{VALUES}[i].\text{noccurrence} \cdot \text{VALUES}[i].\text{val}
\]

Within [297, pages 50–51] the previous condition where terms involving identical variables are grouped together (i.e., rule 5 of MALICE [296]) is mentioned as a crucial deduction rule for the autoref problem.

W.-J. van Hoeve et al. present two soft versions of the `global_cardinality` constraint in [399].

In MiniZinc (http://www.g12.cs.mu.oz.au/minizinc/) there is also a `distribute` constraint where the `val` attribute is not necessarily initially fixed and where a same value may occur more than once. Their is also a `global_cardinality_closed` constraint where all variables must be assigned a value from the `val` attribute.

A flow algorithm that handles the original `global_cardinality` constraint is described in [322]. The two approaches that were used to design bound-consistency algorithms for alldifferent were generalised for the `global_cardinality` constraint. The algorithm in [314] identifies Hall intervals and the one in [215] exploits convexity to achieve a fast implementation of the flow-based arc-consistency algorithm. The later algorithm can also compute bound-consistency for the count variables [216, 213]. An improved algorithm for achieving arc-consistency is described in [313].

See also

- `common keyword: count, max_nvalue, min_nvalue (value constraint, counting constraint), nvalue(counting constraint), open_global_cardinality_low_up(assignment, counting constraint).
- `cost variant: global_cardinality_with_cost(cost associated with each variable, value pair).
- `implied by: global_cardinality_with_cost(forget about cost), same_and_global_cardinality(conjoin same and global_cardinality).`
**soft variant:** open\_global\_cardinality (a set variable defines the set of variables that are actually considered).

**specialisation:**  
- alldifferent (each value should occur at most once),  
- cardinality\_atleast,  
- cardinality\_atmost (individual count variable for each value replaced by single count variable),  
- cardinality\_atmost\_partition (individual count variable for each value replaced by single count variable and variable ∈ partition replaced by variable),  
- global\_cardinality\_low\_up (variable replaced by fixed interval).

**system of constraints:** colored\_matrix (one global\_cardinality constraint for each row and each column of a matrix of variables).

**uses in its reformulation:** tree\_range, tree\_resource.

---

**Keywords**

**application area:** assignment.

**characteristic of a constraint:** core, automaton, automaton with array of counters.

**complexity:** 3-SAT.

**constraint arguments:** pure functional dependency.

**constraint type:** value constraint, counting constraint, system of constraints.

**filtering:** Hall interval, bound-consistency, flow, duplicated variables, DFS-bottleneck.

**modelling:** functional dependency.

**modelling exercises:** magic series.

**puzzles:** magic series, autoref.
For all items of VALUES:

<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>$SELF \rightarrow \text{collection}(\text{variables})$</td>
</tr>
<tr>
<td>Arc arity</td>
<td>1</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>$\text{variables}.\text{var} = \text{VALUES}.\text{val}$</td>
</tr>
</tbody>
</table>

Graph property(ies) $\text{NVERTEX} = \text{VALUES}.\text{noccurrence}$

**Graph model**

Since we want to express one unary constraint for each value we use the “For all items of VALUES” iterator. Part (A) of Figure 5.276 shows the initial graphs associated with each value 3, 5 and 6 of the VALUES collection of the Example slot. Part (B) of Figure 5.276 shows the two corresponding final graphs respectively associated with values 3 and 6 that are both assigned to the variables of the VARIABLES collection (since value 5 is not assigned to any variable of the VARIABLES collection the final graph associated with value 5 is empty). Since we use the $\text{NVERTEX}$ graph property, the vertices of the final graphs are stressed in bold.

![Diagram of graphs](image)

Figure 5.276: Initial and final graph of the global cardinality constraint
Automaton

Figure 5.277 depicts the automaton associated with the global_cardinality constraint. To each item of the collection VARIABLES corresponds a signature variable $S_i$ that is equal to 0. To each item of the collection VALUES corresponds a signature variable $S_{i+|VARIABLES|}$ that is equal to 1.

```
\{c[\_]=0\} \\
S \\
\{c[VAR_1]=c[VAR_1]+1\} \\
1, \\
\{c[VAL_1]=c[VAL_1]-NOCCURRENCE_1\} \\
\text{t: arith(C, =, 0)} \\
\{c[VAL_1]=c[VAL_1]-NOCCURRENCE_1\}
```

Figure 5.277: Automaton of the global_cardinality constraint
5.149  **global_cardinality_low_up**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Used for defining <code>sliding_distribution</code>.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>global_cardinality_low_up(VARIABLES, VALUES)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>gcc_low_up, gcc</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>VARIABLES : collection(var−dvar)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>VALUES : collection(val−int, omin−int, omax−int)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>required(VARIABLES, var)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>`</td>
<td>VALUES</td>
<td>&gt; 0`</td>
</tr>
<tr>
<td><code>required(VVALUES, [val, omin, omax])</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>distinct(VVALUES, val)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>VALUES.omin ≥ 0</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>`VALUES.omin ≤</td>
<td>VARIABLES</td>
<td>`</td>
</tr>
<tr>
<td><code>VALUES.omax ≤ VALUES.omin</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Each value `VALUES[i].val (1 ≤ i ≤</td>
<td>VALUES</td>
<td>)<code>should be taken by at least</code>VALUES[i].omin<code>and at most</code>VALUES[i].omax<code>variables of the</code>VARIABLES` collection.</td>
</tr>
<tr>
<td>Example</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| `\begin{bmatrix}
\langle 3,8,6,\rangle,
\langle val−3\ominom−2\omax−3,\rangle \\
\langle val−5\ominom−0\omax−1,\rangle \\
\langle val−6\ominom−1\omax−2,\rangle \\
\end{bmatrix}` |       |       |
| The `global_cardinality_low_up` constraint holds since values 3, 5 and 6 are respectively used 2 (2 ≤ 2 ≤ 3), 0 (0 ≤ 0 ≤ 1) and 1 (1 ≤ 1 ≤ 2) times within the collection ⟨3,8,6⟩ and since no constraint was specified for value 8. |       |       |
| Typical     |       |       |
| `|VARIABLES| > 1` |       |       |
| `range(VARIABLES.var) > 1` |       |       |
| `|VALUES| > 1` |       |       |
| `VALUES.omin ≤ |VARIABLES|` |       |       |
| `VALUES.omax > 0` |       |       |
| `VALUES.omax < |VARIABLES|` |       |       |
| `|VARIABLES| > |VALUES|` |       |       |
| `in_attr(VARIABLES.var, VALUES, val)` |       |       |
Symmetries

- Items of VARIABLES are permutable.
- An occurrence of a value of VARIABLES.var that does not belong to VALUES.val can be replaced by any other value that also does not belong to VALUES.val.
- Items of VALUES are permutable.
- VALUES.omin can be decreased to any value ≥ 0.
- VALUES.omax can be increased to any value ≤ |VARIABLES|.
- All occurrences of two distinct values in VARIABLES.var or VALUES.val can be swapped; all occurrences of a value in VARIABLES.var or VALUES.val can be renamed to any unused value.

Arg. properties

Contractible wrt. VALUES.

Remark

Within the context of linear programming [198, page 376] provides relaxations of the global_cardinality_low_up constraint.

In MiniZinc (http://www.g12.cs.mu.oz.au/minizinc/) there is also a global_cardinality_low_up_closed constraint where all variables must be assigned a value from the val attribute.

Algorithm

A filtering algorithm achieving arc-consistency for the global_cardinality_low_up constraint is given in [322].

The global_cardinality_low_up constraint is entailed if and only if for each value v equal to VALUES[i].val (with 1 ≤ i ≤ |VALUES|) the following two conditions hold:

1. The number of variables of the VARIABLES collection assigned value v is greater than or equal to VALUES[i].omin.
2. The number of variables of the VARIABLES collection that can potentially be assigned value v is less than or equal to VALUES[i].omax.

Reformulation

A reformulation of the global_cardinality_low_up, involving linear constraints, preserving bound-consistency was introduced in [67]. For each potential interval [l, u] of consecutive values this model uses |VARIABLES| 0-1 variables $B_{i,l,u}, B_{2,l,u}, \ldots, B_{|VARIABLES|,l,u}$ for modelling the fact that each variable of the collection VARIABLES is assigned a value within interval [l, u] (i.e., $\forall i \in [1, |VARIABLES|] : B_{i,l,u} \Leftrightarrow l \leq VARIABLES[i].var \wedge VARIABLES[i].var \leq u$), as well as one domain variable $C_{l,u}$ for counting how many values of [l, u] are assigned to variables of VARIABLES (i.e. $C_{l,u} = B_{1,l,u} + B_{2,l,u} + \ldots + B_{|VARIABLES|,l,u}$). The lower and upper bounds of variable $C_{l,u}$ are respectively initially set with respect to the minimum and maximum number of possible occurrences of the values of interval [l, u]. Finally, assuming that $s$ is the smallest value that can be assigned to the variables of VARIABLES, the constraint $C_{s,u} = C_{s,k} + C_{k+1,u}$ is stated for each $k \in [s, u-1]$.

Systems

globalCardinality in Choco, global_cardinality_low_up in MiniZinc.

Used in

sliding_distribution.

See also

common keyword: open_global_cardinality (assignment,counting constraint).

generalisation: global_cardinality (fixed interval replaced by variable).
implied by: increasing_global_cardinality (a global_cardinality_low_up constraint where the variables are increasing), same_and_global_cardinality_low_up.

related: ordered_global_cardinality (restrictions are done on nested sets of values, all starting from first value).

shift of concept: global_cardinality_low_up_no_loop (assignment of a variable to its position is ignored).

soft variant: open_global_cardinality_low_up (a set variable defines the set of variables that are actually considered).

specialisation: alldifferent (each value should occur at most once).

system of constraints: sliding_distribution (one global_cardinality_low_up constraint for each sliding sequence of SEQ consecutive variables).

Keywords

application area: assignment.
constraint type: value constraint, counting constraint.
filtering: flow, arc-consistency, bound-consistency, DFS-bottleneck, entailment.
For all items of VALUES:

<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>SELF→collection(variables)</td>
</tr>
<tr>
<td>Arc arity</td>
<td>1</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>variables.var = VALUES.val</td>
</tr>
</tbody>
</table>
| Graph property(ies) | • NVERTEX ≥ VALUES.omin  
                      • NVERTEX ≤ VALUES.omax |

Graph model

Since we want to express one unary constraint for each value we use the “For all items of VALUES” iterator. Part (A) of Figure 5.278 shows the initial graphs associated with each value 3, 5 and 6 of the VALUES collection of the Example slot. Part (B) of Figure 5.278 shows the two corresponding final graphs respectively associated with values 3 and 6 that are both assigned to the variables of the VARIABLES collection (since value 5 is not assigned to any variable of the VARIABLES collection the final graph associated with value 5 is empty). Since we use the NVERTEX graph property, the vertices of the final graphs are stressed in bold.

Figure 5.278: Initial and final graph of the global_cardinality_low_up constraint
5.150  global_cardinality_low_up_no_loop

**DESCRIPTION**

Derived from global_cardinality_low_up and tree.

**LINKS**

<table>
<thead>
<tr>
<th>Constraint</th>
<th>global_cardinality_low_up_no_loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINLOOP,</td>
<td>MAXLOOP,</td>
</tr>
<tr>
<td>VARIABLES,</td>
<td>VALUES</td>
</tr>
</tbody>
</table>

**GRAPH**

**Synonym**

gcc_low_up_no_loop.

**Arguments**

- **MINLOOP** : int
- **MAXLOOP** : int
- **VARIABLES** : collection(var−dvar)
- **VALUES** : collection(val−int, omin−int, omx−int)

**Restrictions**

- MINLOOP ≥ 0
- MINLOOP ≤ MAXLOOP
- MAXLOOP ≤ |VARIABLES|
- required(VARIABLES, var)
- |VALUES| > 0
- required(VALUES, [val, omin, omx])
- distinct(VALUES, val)
- VALUES.omin ≥ 0
- VALUES.omx ≤ |VARIABLES|
- VALUES.omin ≤ VALUES.omx

VALUES[i].omin (1 ≤ i ≤ |VALUES|) is less than or equal to the number of variables VARIABLES[j].var (j ≠ i, 1 ≤ j ≤ |VARIABLES|) that are assigned value VALUES[i].val.

VALUES[i].omax (1 ≤ i ≤ |VALUES|) is greater than or equal to the number of variables VARIABLES[j].var (j ≠ i, 1 ≤ j ≤ |VARIABLES|) that are assigned value VALUES[i].val.

The number of assignments of the form VARIABLES[i].var = i (i ∈ [1, |VARIABLES|]) is greater than or equal to MINLOOP and less than or equal to MAXLOOP.

**Purpose**

- Values 1, 5 and 6 are respectively assigned to the set of variables {VARIABLES[2].var} (i.e., omin = 1 ≤ 1 ≤ omx = 1), {} (i.e., omin = 0 ≤ omission)

**Example**

\[
\begin{pmatrix}
1, 1, (1, 1, 8, 6), \\
val - 1 & omin - 1 & omx - 1, \\
val - 5 & omin - 0 & omx - 0, \\
val - 6 & omin - 1 & omx - 2
\end{pmatrix}
\]

The global_cardinality_low_up_no_loop constraint holds since:

- Values 1, 5 and 6 are respectively assigned to the set of variables {VARIABLES[2].var} (i.e., omin = 1 ≤ 1 ≤ omx = 1), {} (i.e., omin = 0 ≤ omission)
\(0 \leq \text{omax} = 0\) and \{\text{VARIABLES}[4].\text{var}\} (i.e., \text{omin} = 1 \leq 1 \leq \text{omax} = 2). Note that, due to the definition of the constraint, the fact that \text{VARIABLES}[1].\text{var} is assigned to 1 is not counted.

- In addition the number of assignments of the form \text{VARIABLES}[i].\text{var} = i (i \in [1, 4]) is greater than or equal to \text{MINLOOP} = 1 and less than or equal to \text{MAXLOOP} = 1.

**Typical**

<table>
<thead>
<tr>
<th>VARIABLES</th>
<th>&gt; 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{range}(\text{VARIABLES}.\text{var})</td>
<td>&gt; 1</td>
</tr>
<tr>
<td></td>
<td>\text{VALUES}</td>
</tr>
<tr>
<td>\text{VALUES}.\text{omin}</td>
<td>\leq</td>
</tr>
<tr>
<td>\text{VALUES}.\text{omax}</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>\text{VALUES}.\text{omax}</td>
<td>&lt;</td>
</tr>
<tr>
<td>\text{VARIABLES}</td>
<td>&gt;</td>
</tr>
</tbody>
</table>

**Symmetries**

- Items of \text{VALUES} are permutable.
- \text{VALUES}.\text{omin} can be decreased to any value \(\geq 0\).
- \text{VALUES}.\text{omax} can be increased to any value \(\leq |\text{VARIABLES}|\).

**Usage**

Within the context of the tree constraint the \text{global_cardinality_low_up_no_loop} constraint allows to model a minimum and maximum degree constraint on each vertex of our trees.

**Algorithm**

The flow algorithm that handles the original \text{global_cardinality} constraint [322] can be adapted to the context of the \text{global_cardinality_low_up_no_loop} constraint. This is done by creating an extra value node representing the loops corresponding to the roots of the trees.

**See also**

- **generalisation:** \text{global_cardinality_no_loop}(fixed interval replaced by variable).
- **implied by:** \text{same_and_global_cardinality_low_up}.
- **related:** \text{tree}(graph partitioning by a set of trees with degree restrictions).
- **root concept:** \text{global_cardinality_low_up}(assignment of a variable to its position is ignored).

**Keywords**

- **constraint type:** value constraint.
- **filtering:** flow.
For all items of VALUES:

<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>SELF→collection(variables)</td>
</tr>
<tr>
<td>Arc arity</td>
<td>1</td>
</tr>
</tbody>
</table>
| Arc constraint(s) | variables.var = VALUES.val  
variables.key ≠ VALUES.val |
| Graph property(ies) | NVERTEX ≥ VALUES.omin  
NVERTEX ≤ VALUES.omax |

<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>SELF→collection(variables)</td>
</tr>
<tr>
<td>Arc arity</td>
<td>1</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>variables.var = variables.key</td>
</tr>
</tbody>
</table>
| Graph property(ies) | NARC ≥ MINLOOP  
NARC ≤ MAXLOOP |

**Graph model**

Since, within the context of the first graph constraint, we want to express one unary constraint for each value we use the “For all items of VALUES” iterator. Part (A) of Figure 5.279 shows the initial graphs associated with each value 1, 5 and 6 of the VALUES collection of the Example slot. Part (B) of Figure 5.279 shows the two corresponding final graphs respectively associated with values 1 and 6 that are both assigned to the variables of the VARIABLES collection (since value 5 is not assigned to any variable of the VARIABLES collection the final graph associated with value 5 is empty). Since we use the NVERTEX graph property, the vertices of the final graphs are stressed in bold.

![Graphs](image)

**Figure 5.279:** Initial and final graph of the `global_cardinality_low_up_no_loop` constraint
5.151  global_cardinality_no_loop

Origin        Derived from global_cardinality and tree.
Constraint    global_cardinality_no_loop(NLOOP, VARIABLES, VALUES)
Synonym       gcc_no_loop.
Arguments     NLOOP : dvar
              VARIABLES : collection(var−dvar)
              VALUES : collection(val−int,noccurrence−dvar)
Restrictions  NLOOP ≥ 0
              NLOOP ≤ |VARIABLES|
              required(VARIABLES,var)
              |VALUES| > 0
              required(VVALUES,[val,noccurrence])
              distinct(VVALUES,val)
              VALUES.noccurrence ≥ 0
              VALUES.noccurrence ≤ |VARIABLES|

VALUES[i].noccurrence (1 ≤ i ≤ |VALUES|) is equal to the number of variables VARIABLES[j].var (j ≠ i, 1 ≤ j ≤ |VARIABLES|) that are assigned value VARIABLES[i].val.
The number of assignments of the form VARIABLES[i].var = i (i ∈ [1, |VARIABLES|]) is equal to NLOOP.

Example

The global_cardinality_no_loop constraint holds since:

- Values 1, 5 and 6 are respectively assigned to the set of variables {VARIABLES[2].var} (i.e., 1 occurrence of value 1), {} (i.e., no occurrence of value 5) and {VARIABLES[4].var} (i.e., 1 occurrence of value 6). Note that, due to the definition of the constraint, the fact that VARIABLES[1].var is assigned to 1 is not counted.
- In addition the number of assignments of the form VARIABLES[i].var = i (i ∈ [1, 4]) is equal to NLOOP = 1.

Typical

|VARIABLES| > 1
range(VARIABLES.var) > 1
|VALUES| > 1
|VARIABLES| > |VALUES|
Symmetry

Items of VALUES are permutable.

Arg. properties

- Functional dependency: NLOOP determined by VARIABLES.
- Functional dependency: VALUES.noccurrence determined by VARIABLES and VALUES.val.

Usage

Within the context of the tree constraint the global_cardinality_no_loop constraint allows to model a minimum and maximum degree constraint on each vertex of our trees.

Algorithm

The flow algorithm that handles the original global_cardinality constraint [322] can be adapted to the context of the global_cardinality_no_loop constraint. This is done by creating an extra value node representing the loops corresponding to the roots of the trees.

See also

related: tree (graph partitioning by a set of trees with degree restrictions).

root concept: global_cardinality (assignment of a variable to its position is ignored).

specialisation: global_cardinality_low_up_no_loop (variable replaced by fixed interval).

Keywords

constraint arguments: pure functional dependency.
constraint type: value constraint.
filtering: flow.
modelling: functional dependency.
For all items of VALUES:

<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>SELF→collection(variables)</td>
</tr>
<tr>
<td>Arc arity</td>
<td>1</td>
</tr>
</tbody>
</table>
| Arc constraint(s) | • variables.var = VALUES.val  
• variables.key ≠ VALUES.val |
| Graph property(ies) | NVERTEX = VALUES.noccurrence |

<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>SELF→collection(variables)</td>
</tr>
<tr>
<td>Arc arity</td>
<td>1</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>variables.var = variables.key</td>
</tr>
<tr>
<td>Graph property(ies)</td>
<td>NARC = NLOOP</td>
</tr>
</tbody>
</table>

**Graph model**

Since, within the context of the first graph constraint, we want to express one unary constraint for each value we use the “For all items of VALUES” iterator. Part (A) of Figure 5.280 shows the initial graphs associated with each value 1, 5 and 6 of the VALUES collection of the Example slot. Part (B) of Figure 5.280 shows the two corresponding final graphs respectively associated with values 1 and 6 that are both assigned to the variables of the VARIABLES collection (since value 5 is not assigned to any variable of the VARIABLES collection the final graph associated with value 5 is empty). Since we use the NVERTEX graph property, the vertices of the final graphs are stressed in bold.

Figure 5.280: Initial and final graph of the global_cardinality_no_loop constraint
5.152  global_cardinality_with_costs

DESCRIPTION  LINKS  GRAPH

Origin  [324]

Constraint  global_cardinality_with_costs(VARIABLES, VALUES, MATRIX, COST)

Synonyms  gcc, cost_gcc.

Arguments  VARIABLES : collection(var−dvar)
VALUES : collection(val−int,noccurrence−dvar)
MATRIX : collection(i−int,j−int,c−int)
COST : dvar

Restrictions  required(VARIABLES, var)
|VALUES| > 0
required(VALUES, [val, noccurrence])
distinct(VALUES, val)
VALUES.noccurrence ≥ 0
VALUES.noccurrence ≤ |VARIABLES|
required(MATRIX, [i, j, c])
increasing_seq(MATRIX, [i, j])
MATRIX.i ≥ 1
MATRIX.i ≤ |VARIABLES|
MATRIX.j ≥ 1
MATRIX.j ≤ |VALUES|
|MATRIX| = |VARIABLES| * |VALUES|

Each value VALUES[i].val should be taken by exactly VALUES[i].noccurrence variables of the VARIABLES collection. In addition the COST of an assignment is equal to the sum of the elementary costs associated with the fact that we assign variable i of the VARIABLES collection to the jth value of the VALUES collection. These elementary costs are given by the MATRIX collection.

Purpose
The \textit{global\_cardinality\_with\_costs} constraint holds since:

- Values 3, 5 and 6 respectively occur 3, 0 and 1 times within the collection \((3, 3, 3, 6)\).
- The \texttt{COST} argument corresponds to the sum of the costs respectively associated with the first, second, third and fourth items of \((3, 3, 3, 6)\), namely 4, 1, 3 and 6.

Typical

\[
\begin{align*}
|\text{VARIABLES}| & > 1 \\
\text{range}(\text{VARIABLES.var}) & > 1 \\
|\text{VALUES}| & > 1 \\
\text{range}(\text{VALUES.noccurrence}) & > 1 \\
\text{range}(\text{MATRIX.c}) & > 1 \\
|\text{VARIABLES}| & > |\text{VALUES}|
\end{align*}
\]

Arg. properties

- Functional dependency: \texttt{VALUES.noccurrence} determined by \texttt{VARIABLES}.
- Functional dependency: \texttt{COST} determined by \texttt{VARIABLES}, \texttt{VALUES} and \texttt{MATRIX}.

Usage

A classical utilisation of the \textit{global\_cardinality\_with\_costs} constraint corresponds to the following assignment problem. We have a set of persons \(P\) as well as a set of jobs \(J\) to perform. Each job requires a number of persons restricted to a specified interval. In addition each person \(p\) has to be assigned to one specific job taken from a subset \(J_p\) of \(J\). There is a cost \(C_{pj}\) associated with the fact that person \(p\) is assigned to job \(j\). The previous problem is modelled with one single \textit{global\_cardinality\_with\_costs} constraint where the persons and the jobs respectively correspond to the items of the \texttt{VARIABLES} and \texttt{VALUES} collection.

The \textit{global\_cardinality\_with\_costs} constraint can also be used for modelling a conjunction \texttt{alldifferent}(\(X_1, X_2, \ldots, X_n\)) and \(\alpha_1 \cdot X_1 + \alpha_2 \cdot X_2 + \cdots + \alpha_n \cdot X_n = \text{COST}\). For this purpose we set the domain of the \texttt{noccurrence} variables to \(\{0, 1\}\) and the cost attribute \(c\) of a variable \(X_i\) and one of its potential value \(j\) to \(\alpha_i \cdot j\). In practice this can be used for the magic squares and the magic hexagon problems where all the \(\alpha_i\) are set to 1.

Algorithm

A filtering algorithm achieving \textit{arc-consistency} independently on each side (i.e., the greater than or equal to side and the less than or equal to side) of the
global_cardinality_with_costs constraint is described in [324, 326]. This algorithm
assumes for each value a fixed minimum and maximum number of occurrences. If we
rather have occurrence variables, the Reformulation slot explains how to also obtain some
propagation from the cost variable back to the occurrence variables.

**Reformulation**

Let \( n \) and \( m \) respectively denote the number of items of the VARIABLES and of the VALUES collections. Let \( v_1, v_2, \ldots, v_m \) denote the values in \( \text{VALUES}[1].\text{val}, \text{VALUES}[2].\text{val}, \ldots, \text{VALUES}[m].\text{val} \). In addition let \( \text{LINE}_i \) (with \( i \in [1, n] \)) denote the values \( \langle \text{MATRIX}[m \cdot (i - 1) + 2].\text{c}, \ldots, \text{MATRIX}[m \cdot i].\text{c} \rangle \), i.e., line \( i \) of the matrix \( \text{MATRIX} \).

By introducing \( 2 \cdot n \) auxiliary variables \( U_1, U_2, \ldots, U_n \) and \( C_1, C_2, \ldots, C_n \), the \text{global_cardinality_with_costs(VARIABLES, VALUES, MATRIX, COST)} constraint can be expressed in term of the conjunction of one \text{global_cardinality(VARIABLES, VALUES)} constraint, \( 2 \cdot n \) \text{element} constraints and one arithmetic constraint \text{sum_ctr}.

For each variable \( V_i \) (with \( i \in [1, |\text{VARIABLES}|] \)) of the VARIABLES collection a first \text{element}(\( U_i, \langle v_1, v_2, \ldots, v_m \rangle, V_i \)) constraint provides the correspondence between the variable \( V_i \) and the index of the value \( U_i \) to which it is assigned. A second \text{element}(\( U_i, \text{LINE}_i, C_i \)) links the previous index \( U_i \) to the cost \( C_i \) variable associated with variable \( V_i \). Finally the total cost \( \text{COST} \) is equal to the sum \( C_1 + C_2 + \cdots + C_n \).

In the context of the Example slot we get the following conjunction of constraints:

\[
\text{global_cardinality}(3, 3, 3, 6),
\]

\[
\text{element}(1, \langle 3, 5, 6 \rangle, 3), \\
\text{element}(1, \langle 3, 5, 6 \rangle, 3), \\
\text{element}(1, \langle 3, 5, 6 \rangle, 3), \\
\text{element}(3, \langle 3, 5, 6 \rangle, 6), \\
\text{element}(1, \langle 4, 1, 7 \rangle, 4), \\
\text{element}(1, \langle 1, 0, 8 \rangle, 1), \\
\text{element}(1, \langle 3, 2, 1 \rangle, 3), \\
\text{element}(3, \langle 0, 0, 6 \rangle, 6), \\
14 = 4 + 1 + 3 + 6.
\]

We now show how to add implied constraints that can also propagate from the cost variable back to the occurrence variables. Let \( O_1, O_2, \ldots, O_m \) respectively denote the variables \( \text{VALUES}[1].\text{noccurrence}, \text{VALUES}[2].\text{noccurrence}, \ldots, \text{VALUES}[m].\text{noccurrence} \). The idea is to get for each value \( v_i \) (with \( i \in [1, m] \)) an idea of its minimum and maximum contribution in the total cost \( \text{COST} \) that is linked to the number of times it is assigned to a variables of VARIABLES. E.g., if value \( v_i \) (with \( i \in [1, m] \)) is used twice, then the corresponding minimum (respectively maximum) contribution in the total cost \( \text{COST} \) will be at least equal to the sum of the two smallest (respectively largest) costs attached to row \( i \). Let \( D_i \) (with \( i \in [1, m] \)) denotes the contribution that stems from the variables of VARIABLES that are assigned value \( v_i \). For each value \( v_i \) (with \( i \in [1, m] \)) we create one \text{element} constraint for linking \( O_i + 1 \) to the corresponding minimum contribution \( \text{LOW}_i \). The table of that \text{element} constraint has \( n + 1 \) entries, where entry \( j \) (with \( j \in [0, n] \)) corresponds to the sum of the \( j^{th} \) smallest entries of row \( i \) of the cost matrix \( \text{MATRIX} \). Similarly we create for each value \( v_i \) (with \( i \in [1, m] \)) one \text{element} constraint for linking \( O_i + 1 \) to the corresponding maximum contribution \( \text{UP}_i \). The table of that
element constraint also has $n + 1$ entries, where entry $j$ (with $j \in [0, n]$) corresponds to the sum of the $j^{th}$ largest entries of row $i$ of the cost matrix $\text{MATRIX}$.

In the context of the cost matrix of the Example slot we get the following conjunction of implied constraints:

\[
\text{COST} = D_1 + D_2 + D_3, \\
n = O_1 + O_2 + O_3, \\
P_1 = O_1 + 1, \\
P_2 = O_2 + 1, \\
P_3 = O_3 + 1, \\
\text{element}(P_1, \langle 0, 0, 1, 4, 8 \rangle, \text{LOW}_1), \\
\text{element}(P_2, \langle 0, 0, 1, 3 \rangle, \text{LOW}_2), \\
\text{element}(P_3, \langle 0, 1, 7, 14, 22 \rangle, \text{LOW}_3), \\
\text{element}(P_1, \langle 0, 4, 7, 8 \rangle, \text{UP}_1), \\
\text{element}(P_2, \langle 0, 2, 3, 3 \rangle, \text{UP}_2), \\
\text{element}(P_3, \langle 0, 8, 15, 21, 22 \rangle, \text{UP}_3), \\
\text{LOW}_1 \leq D_1, D_1 \leq \text{UP}_1, \\
\text{LOW}_2 \leq D_2, D_2 \leq \text{UP}_2, \\
\text{LOW}_3 \leq D_3, D_3 \leq \text{UP}_3.
\]
For all items of VALUES:

<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>(SELF \rightarrow \text{collection(variables)})</td>
</tr>
<tr>
<td>Arc arity</td>
<td>1</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>variables.var = VALUES.val</td>
</tr>
<tr>
<td>Graph property(ies)</td>
<td>(N\text{VERTEX} = \text{VALUES.noccurrence})</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>(PRODUCT \rightarrow \text{collection}(\text{variables}.values))</td>
</tr>
<tr>
<td>Arc arity</td>
<td>2</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>variables.var = values.val</td>
</tr>
<tr>
<td>Graph property(ies)</td>
<td>(\text{SUM_WEIGHT_ARC}(\text{MATRIX}[(\text{variables}.key - 1) *</td>
</tr>
</tbody>
</table>

**Graph model**

The first graph constraint enforces each value of the VALUES collection to be taken by a specific number of variables of the VARIABLES collection. It is identical to the graph constraint used in the global_cardinality constraint. The second graph constraint expresses that the COST variable is equal to the sum of the elementary costs associated with each variable-value assignment. All these elementary costs are recorded in the MATRIX collection. More precisely, the cost \(c_{ij}\) is recorded in the attribute \(c\) of the \(((i - 1) \cdot |\text{VALUES}| + j)^{th}\) entry of the MATRIX collection. This is ensured by the increasing restriction that enforces the fact that the items of the MATRIX collection are sorted in lexicographically increasing order according to attributes \(i\) and \(j\).

Parts (A) and (B) of Figure 5.281 respectively show the initial and final graph associated with the second graph constraint of the Example slot.
Figure 5.281: Initial and final graph of the global_cardinality_with_costs constraint
5.153 global_contiguity

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint</td>
</tr>
<tr>
<td>Synonym</td>
</tr>
<tr>
<td>Argument</td>
</tr>
<tr>
<td>Restrictions</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enforce all variables of the VARIABLES collection to be assigned value 0 or 1. In addition, all variables assigned to value 1 appear contiguously.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>((0, 1, 1, 0))</td>
</tr>
</tbody>
</table>

The global_contiguity constraint holds since the sequence 0 1 1 0 contains no more than one group of contiguous 1.

<table>
<thead>
<tr>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symmetry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items of VARIABLES can be reversed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arg. properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contractible wrt. VARIABLES.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>The article [252] introducing this constraint refers to hardware configuration problems.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>A filtering algorithm for this constraint is described in [252].</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>See also</th>
</tr>
</thead>
<tbody>
<tr>
<td>common keyword: group, inflexion(sequence).</td>
</tr>
<tr>
<td>implies: consecutive_values, multi_global_contiguity, no_valley.</td>
</tr>
<tr>
<td>related: roots.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>characteristic of a constraint: convex, automaton, automaton without counters, reified automaton constraint.</td>
</tr>
<tr>
<td>combinatorial object: sequence.</td>
</tr>
<tr>
<td>constraint network structure: Berge-acyclic constraint network.</td>
</tr>
<tr>
<td>filtering: arc-consistency.</td>
</tr>
<tr>
<td>final graph structure: connected component.</td>
</tr>
</tbody>
</table>
Arc input(s)  VARIABLES
Arc generator  \[\text{PATH} \rightarrow \text{collection}(\text{variables1, variables2})\]
\[\text{LOOP} \rightarrow \text{collection}(\text{variables1, variables2})\]
Arc arity  2
Arc constraint(s)  
• variables1.var = variables2.var
• variables1.var = 1
Graph property(ies)  \text{NCC} \leq 1

Graph model
Each connected component of the final graph corresponds to one set of contiguous variables that all take value 1.

Parts (A) and (B) of Figure 5.282 respectively show the initial and final graph associated with the Example slot. The global contiguity constraint holds since the final graph does not contain more than one connected component. This connected component corresponds to 2 contiguous variables that are both assigned to 1.

![Figure 5.282: Initial and final graph of the global contiguity constraint](image)
Automaton

Figure 5.283 depicts the automaton associated with the global.contiguity constraint. To each variable $\text{VAR}_i$ of the collection $\text{VARIABLES}$ corresponds a signature variable that is equal to $\text{VAR}_i$. There is no signature constraint.

Figure 5.283: Automaton of the global.contiguity constraint

Figure 5.284: Hypergraph of the reformulation corresponding to the automaton of the global.contiguity constraint
### 5.154 golomb

<table>
<thead>
<tr>
<th>Description</th>
<th>Links</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td></td>
<td>Variables : collection(var−dvar)</td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td>required(VARIABLES, var)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VARIABLES.var ≥ 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>strictly_increasing(VARIABLES)</td>
</tr>
<tr>
<td>Purpose</td>
<td></td>
<td>Given a strictly increasing sequence X1, X2, . . . , Xn, enforce all differences X_i − X_j between two variables X_i and X_j (i &gt; j) to be distinct.</td>
</tr>
<tr>
<td>Example</td>
<td></td>
<td>((0, 1, 4, 6))</td>
</tr>
</tbody>
</table>

Figure 5.285 gives a graphical interpretation of the solution given in the example in term of a graph: each vertex corresponds to a value of (0, 1, 4, 6), while each arc depicts a difference between two values. The golomb constraint holds since one can note that these differences 1, 4, 6, 3, 5, 2 are all-distinct.

![Figure 5.285: Graphical representation of the solution 0,1,4,6](image)

**Typical**

| VARIABLES | > 2 |

**Symmetry**

One and the same constant can be added to the var attribute of all items of VARIABLES.

**Arg. properties**

Contractible wrt. VARIABLES.

**Usage**

This constraint refers to the Golomb ruler problem. We quote the definition from [359]: “A Golomb ruler is a set of integers (marks) a_1 < · · · < a_k such that all the differences a_i − a_j (i > j) are distinct.”
Remark

Different constraints models for the Golomb ruler problem were presented in [371].

Algorithm

At a first glance, one could think that, because it looks so similar to the \texttt{alldifferent} constraint, we could have a perfect polynomial filtering algorithm. However this is not true since one retrieves the \textit{same} variable in different vertices of the graph. This leads to the fact that one has incompatible arcs in the bipartite graph (the two classes of vertices correspond to the pair of variables and to the fact that the difference between two pairs of variables takes a specific value). However one can still reuse a similar filtering algorithm as for the \texttt{alldifferent} constraint, but this will not lead to perfect pruning.

See also

\texttt{common keyword: alldifferent (all different).}
\texttt{implies: strictly increasing.}

Keywords

\texttt{characteristic of a constraint: disequality, difference, all different, derived collection.}
\texttt{puzzles: Golomb ruler.}
**Derived Collection**

\[
\text{col}( \text{PAIRS} \rightarrow \text{collection}(x \rightarrow \text{dvar}, y \rightarrow \text{dvar}), \\
> \text{item}(x \rightarrow \text{VARIABLES}.\text{var}, y \rightarrow \text{VARIABLES}.\text{var}))
\]

**Arc input(s)**
- PAIRS

**Arc generator**
- \( \text{CLIQUE} \rightarrow \text{collection}(\text{pairs}_1, \text{pairs}_2) \)

**Arc arity**
- 2

**Arc constraint(s)**
- \( \text{pairs}_1.y - \text{pairs}_1.x = \text{pairs}_2.y - \text{pairs}_2.x \)

**Graph property(ies)**
- \( \text{MAX\_NSCC} \leq 1 \)

**Graph model**

When applied on the collection of items \( \langle \text{VAR1}, \text{VAR2}, \text{VAR3}, \text{VAR4} \rangle \), the generator of derived collection generates the following collection of items: \( \langle \text{VAR2 VAR1}, \text{VAR3 VAR1}, \text{VAR3 VAR2}, \text{VAR4 VAR1}, \text{VAR4 VAR2}, \text{VAR4 VAR3} \rangle \). Note that we use a binary arc constraint between two vertices and that this binary constraint involves four variables.

Parts (A) and (B) of Figure 5.286 respectively show the initial and final graph associated with the Example slot. Since we use the \( \text{MAX\_NSCC} \) graph property we show one of the largest strongly connected component of the final graph. The constraint holds since all the strongly connected components have at most one vertex: the differences 1, 2, 3, 4, 5, 6 that one can construct from the values 0, 1, 4, 6 assigned to the variables of the \text{VARIABLES} collection are all-distinct.

![Graph](image-url)

**Figure 5.286:** Initial and final graph of the golomb constraint
5.155 \textbf{graph\_crossing}

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>N. Beldiceanu</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>\texttt{graph_crossing(NCROSS,NODES)}</td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>crossing,ncross.</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>\begin{itemize} \item \texttt{NCROSS} : \texttt{dvar} \item \texttt{NODES} : \texttt{collection(succ_dvar,x_int,y_int)} \end{itemize}</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>\begin{itemize} \item \texttt{NCROSS} \geq 0 \item \texttt{required(NODES,[succ,x,y])} \item \texttt{NODES.succ} \geq 1 \item \texttt{NODES.succ} \leq</td>
<td>\texttt{NODES}</td>
</tr>
<tr>
<td>Purpose</td>
<td>\texttt{NCROSS} is the number of proper intersections between line-segments, where each line-segment is an arc of the directed graph defined by the arc linking a node and its unique successor.</td>
<td></td>
</tr>
</tbody>
</table>

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{example.png}
\caption{Figure 5.287 shows the line-segments associated with the \texttt{NODES} collection. One can note the following line-segments intersection:}
\end{figure}

- Arcs 8 \to 9 and 7 \to 3 cross,
- Arcs 5 \to 3 and 6 \to 2 cross also.

Consequently, the \texttt{graph\_crossing} constraint holds since its first argument \texttt{NCROSS} is set to 2.

<table>
<thead>
<tr>
<th>Typical</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{</td>
<td>NODES</td>
</tr>
<tr>
<td>\texttt{range(NODES.succ)} \geq 1</td>
<td></td>
</tr>
<tr>
<td>\texttt{range(NODES.x)} \geq 1</td>
<td></td>
</tr>
<tr>
<td>\texttt{range(NODES.y)} \geq 1</td>
<td></td>
</tr>
</tbody>
</table>
Symmetries

- Attributes of NODES are permutable w.r.t. permutation \((\text{succ}) (x, y)\) (permutation applied to all items).
- One and the same constant can be added to the \(x\) attribute of all items of NODES.
- One and the same constant can be added to the \(y\) attribute of all items of NODES.

Arg. properties

- Functional dependency: NCROSS determined by NODES.

Usage

This is a general crossing constraint that can be used in conjunction with one graph covering constraint such as cycle, tree or map. In many practical problems ones want not only to cover a graph with specific patterns but also to avoid too much crossing between the arcs of the final graph.

Remark

We did not give a specific crossing constraint for each graph covering constraint. We feel that it is better to start first with a more general constraint before going in the specificity of the pattern that is used for covering the graph.

See also

- common keyword: crossing (line-segments intersection), cycle, map, tree (graph constraint, graph partitioning constraint), two_layer_edge_crossing (line-segments intersection).

Keywords

- constraint arguments: pure functional dependency.
- constraint type: graph constraint, graph partitioning constraint.
- geometry: geometrical constraint, line-segments intersection.
Figure 5.287: A graph covering with 2 line-segments intersections
Arc input(s) NODES

Arc generator  $\text{CLIQUE}(<) \mapsto \text{collection}(n_1,n_2)$

Arc arity 2

Arc constraint(s)

- $\max(n_1.x, \text{NODES}[n_1.\text{succ}].x) \geq \min(n_2.x, \text{NODES}[n_2.\text{succ}].x)$
- $\max(n_2.x, \text{NODES}[n_2.\text{succ}].x) \geq \min(n_1.x, \text{NODES}[n_1.\text{succ}].x)$
- $\max(n_1.y, \text{NODES}[n_1.\text{succ}].y) \geq \min(n_2.y, \text{NODES}[n_2.\text{succ}].y)$
- $\max(n_2.y, \text{NODES}[n_2.\text{succ}].y) \geq \min(n_1.y, \text{NODES}[n_1.\text{succ}].y)$
- $(n_2.x - \text{NODES}[n_1.\text{succ}].x) \cdot (\text{NODES}[n_1.\text{succ}].y - n_1.y) \neq 0$
- $(\text{NODES}[n_1.\text{succ}].x - n_1.x) \cdot (n_2.y - \text{NODES}[n_1.\text{succ}].y) \neq 0$
- $(\text{NODES}[n_1.\text{succ}].x - n_1.x) \cdot (n_2.y - \text{NODES}[n_1.\text{succ}].y) \neq 0$
- $\text{sign}\left(\frac{(n_2.x - \text{NODES}[n_1.\text{succ}].x) \cdot (\text{NODES}[n_1.\text{succ}].y - n_1.y)}{(\text{NODES}[n_1.\text{succ}].x - n_1.x) \cdot (n_2.y - \text{NODES}[n_1.\text{succ}].y)}\right) \neq 0$
- $\text{sign}\left(\frac{(n_2.x - \text{NODES}[n_1.\text{succ}].x) \cdot (\text{NODES}[n_1.\text{succ}].y - n_1.y)}{(\text{NODES}[n_1.\text{succ}].x - n_1.x) \cdot (n_2.y - \text{NODES}[n_1.\text{succ}].y)}\right) \neq 0$
- $\text{sign}\left(\frac{(n_2.x - n_1.x) \cdot (\text{NODES}[n_2.\text{succ}].y - \text{NODES}[n_1.\text{succ}].y)}{(n_2.x - n_1.x) \cdot (\text{NODES}[n_2.\text{succ}].y - \text{NODES}[n_1.\text{succ}].y)}\right) \neq 0$

Graph property(ies)  $\text{NARC}=\text{NCROSS}$

Graph model

Each node is described by its coordinates $x$ and $y$, and by its successor $\text{succ}$ in the final covering. Note that the co-ordinates are initially fixed. We use the arc generator $\text{CLIQUE}(<)$ in order to avoid counting twice the same line-segment crossing.

Parts (A) and (B) of Figure 5.288 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold. Each arc of the final graph corresponds to a proper intersection between two line-segments.
Figure 5.288: Initial and final graph of the graph_crossing constraint
### 5.156 graph_isomorphism

#### DESCRIPTION

**Origin**

[258]

**Constraint**

`graph_isomorphism(NODES_PATTERN, NODES_TARGET, FUNCTION)`

**Arguments**

- `NODES_PATTERN`: `collection(index=int, succ=sint)`
- `NODES_TARGET`: `collection(index=int, succ=sint)`
- `FUNCTION`: `collection(image=dvar)`

**Restrictions**

- `required(NODES_PATTERN, [index, succ])`
- `NODES_PATTERN.index ≥ 1`
- `NODES_PATTERN.index ≤ |NODES_PATTERN|`
- `distinct(NODES_PATTERN, index)`
- `NODES_PATTERN.succ ≥ 1`
- `NODES_PATTERN.succ ≤ |NODES_PATTERN|`
- `required(NODES_TARGET, [index, succ])`
- `NODES_TARGET.index ≥ 1`
- `NODES_TARGET.index ≤ |NODES_TARGET|`
- `distinct(NODES_TARGET, index)`
- `NODES_TARGET.succ ≥ 1`
- `NODES_TARGET.succ ≤ |NODES_TARGET|`
- `|NODES_TARGET| = |NODES_PATTERN|`
- `required(FUNCTION, [image])`
- `FUNCTION.image ≥ 1`
- `FUNCTION.image ≤ |NODES_TARGET|`
- `distinct(FUNCTION, image)`
- `|FUNCTION| = |NODES_PATTERN|`

#### PURPOSE

Given two directed graphs `PATTERN` and `TARGET` enforce a one to one correspondence, defined by the function `FUNCTION`, between the vertices of the graph `PATTERN` and the vertices of the graph `TARGET` so that:

1. if there is an arc from \( u \) to \( v \) in the graph `PATTERN`, then there is also an arc from the image of \( u \) to the image of \( v \) in the graph `TARGET`,

2. if there is no arc from \( u \) to \( v \) in the graph `PATTERN`, then there is also no arc from the image of \( u \) to the image of \( v \) in the graph `TARGET`.

Both, the `PATTERN` and `TARGET` are fixed, and the vertices of both graphs are respectively defined by the two collections of vertices `NODES_PATTERN` and `NODES_TARGET`. 
Example

\[
\begin{array}{c}
\text{index} - 1 \quad \text{succ} \rightarrow \{2, 4\}, \\
\text{index} - 2 \quad \text{succ} \rightarrow \{1, 3, 4\}, \\
\text{index} - 3 \quad \text{succ} \rightarrow \emptyset, \\
\text{index} - 4 \quad \text{succ} \rightarrow \emptyset, \\
\text{index} - 1 \quad \text{succ} \rightarrow \emptyset, \\
\text{index} - 2 \quad \text{succ} \rightarrow \{1, 3, 4\}, \\
\text{index} - 3 \quad \text{succ} \rightarrow \emptyset, \\
\text{index} - 4 \quad \text{succ} \rightarrow \{1, 2\}, \\
\{4, 2, 3, 1\}
\end{array}
\]

Figure 5.289 gives the pattern (see Part (A)) and target graph (see Part (B)) of the Example slot as well as the one to one correspondence (see Part (C)) between the pattern graph and the target graph. The graph isomorphism constraint since the pattern and target graphs have the same number of vertices and arcs and since:

- To the arc from vertex 1 to vertex 4 in the pattern graph corresponds the arc from vertex 4 to 1 in the target graph.
- To the arc from vertex 1 to vertex 2 in the pattern graph corresponds the arc from vertex 4 to 2 in the target graph.
- To the arc from vertex 2 to vertex 1 in the pattern graph corresponds the arc from vertex 2 to 4 in the target graph.
- To the arc from vertex 2 to vertex 4 in the pattern graph corresponds the arc from vertex 2 to 1 in the target graph.
- To the arc from vertex 2 to vertex 3 in the pattern graph corresponds the arc from vertex 2 to 3 in the target graph.

Typical

\[|\text{NODES\_PATTERN}| > 1\]

Symmetries

- Items of \text{NODES\_PATTERN} are permutable.
- Items of \text{NODES\_TARGET} are permutable.

Algorithm

A constraint approach is described in [373].

See also related: subgraph isomorphism.

Keywords

- constraint arguments: constraint involving set variables.
- constraint type: predefined constraint, graph constraint.
Figure 5.289: (A) The pattern graph, (B) the target graph and (C) the correspondence between the vertices of the pattern graph and the vertices of the target graph.
5.157 group

**DESCRIPTION**

Origin: CHIP

**Constraints**

\[
\text{group} \left( \begin{array}{c}
\text{NGROUP,} \\
\text{MIN_SIZE,} \\
\text{MAX_SIZE,} \\
\text{MIN_DIST,} \\
\text{MAX_DIST,} \\
\text{NVAL,} \\
\text{VARIABLES,} \\
\text{VALUES}
\end{array} \right)
\]

**Arguments**

- \text{NGROUP} : dvar
- \text{MIN_SIZE} : dvar
- \text{MAX_SIZE} : dvar
- \text{MIN_DIST} : dvar
- \text{MAX_DIST} : dvar
- \text{NVAL} : dvar
- \text{VARIABLES} : collection(var-dvar)
- \text{VALUES} : collection(val-int)

**Restrictions**

- \text{NGROUP} \geq 0
- \text{MIN_SIZE} \geq 0
- \text{MAX_SIZE} \geq \text{MIN_SIZE}
- \text{MIN_DIST} \geq 0
- \text{MAX_DIST} \geq \text{MIN_DIST}
- \text{MAX_DIST} \leq |\text{VARIABLES}|
- \text{NVAL} \geq \text{MAX_SIZE}
- \text{NVAL} \geq \text{NGROUP}
- \text{NVAL} \leq |\text{VARIABLES}|
- \text{required}(\text{VARIABLES}, \text{var})
- \text{required}(\text{VALUES}, \text{val})
- \text{distinct}(\text{VALUES}, \text{val})
Let $n$ be the number of variables of the collection $\text{VARIABLES}$. Let $X_i, X_{i+1}, \ldots, X_j$ ($1 \leq i \leq j \leq n$) be consecutive variables of the collection of variables $\text{VARIABLES}$ such that all the following conditions simultaneously apply:

- All variables $X_i, \ldots, X_j$ take their value in the set of values $\text{VALUES}$,
- $i = 1$ or $X_{i-1}$ does not take a value in $\text{VALUES}$,
- $j = n$ or $X_{j+1}$ does not take a value in $\text{VALUES}$.

We call such a set of variables a group. The constraint group is true if all the following conditions hold:

- There are exactly $\text{NGROUP}$ groups of variables,
- $\text{MIN\_SIZE}$ is the number of variables of the smallest group,
- $\text{MAX\_SIZE}$ is the number of variables of the largest group,
- $\text{MIN\_DIST}$ is the minimum number of variables between two consecutive groups or between one border and one group,
- $\text{MAX\_DIST}$ is the maximum number of variables between two consecutive groups or between one border and one group,
- $\text{NVAL}$ is the number of variables that take their value in the set of values $\text{VALUES}$.

**Example**

Given the fact that groups are formed by even values in $\{0, 2, 4, 6, 8\}$ (i.e., values expressed by the $\text{VALUES}$ collection), the group constraint holds since:

- Its first argument, $\text{NGROUP}$, is set to value 2 since the sequence $2 \ 8 \ 1 \ 7 \ 4 \ 5 \ 1 \ 1 \ 1$ contains two groups of even values (i.e., group $2 \ 8$ and group $4$).
- Its second argument, $\text{MIN\_SIZE}$, is set to value 1 since the smallest group of even values involves only one single value (i.e., value $4$).
- Its third argument, $\text{MAX\_SIZE}$, is set to value 2 since the largest group of even values involves two values (i.e., group $2 \ 8$).
- Its fourth argument, $\text{MIN\_DIST}$, is set to value 2 since the smallest group of odd values involves two values (i.e., group $1 \ 7$).
- Its fifth argument, $\text{MAX\_DIST}$, is set to value 4 since the largest group of odd values involves four values (i.e., group $5 \ 1 \ 1 \ 1$).
- Its sixth argument, $\text{NVAL}$, is set to value 3 since the total number of even values of the sequence $2 \ 8 \ 1 \ 7 \ 4 \ 5 \ 1 \ 1 \ 1$ is equal to 3 (i.e., values $2, 8$ and $4$).
1078

Typical

NGROUP > 0
MIN_SIZE > 0
MAX_SIZE > MIN_SIZE
MIN_DIST > 0
MAX_DIST > MIN_DIST
MAX_DIST < |VARIABLES|
NVAL > MAX_SIZE
NVAL > NGROUP
NVAL < |VARIABLES|
|VARIABLES| > 1
range(VARIABLES.var) > 1
|VALUES| > 0
|VARIABLES| > |VALUES|

Symmetries

- Items of VARIABLES can be reversed.
- Items of VALUES are permutable.
- An occurrence of a value of VARIABLES.var that belongs to VALUES.val (resp. does not belong to VALUES.val) can be replaced by any other value in VALUES.val (resp. not in VALUES.val).

Arg. properties

- Functional dependency: NGROUP determined by VARIABLES and VALUES.
- Functional dependency: MIN_SIZE determined by VARIABLES and VALUES.
- Functional dependency: MAX_SIZE determined by VARIABLES and VALUES.
- Functional dependency: MIN_DIST determined by VARIABLES and VALUES.
- Functional dependency: MAX_DIST determined by VARIABLES and VALUES.
- Functional dependency: NVAL determined by VARIABLES and VALUES.

Usage

A typical use of the group constraint in the context of timetabling is as follow: The value of the $i^{th}$ variable of the VARIABLES collection corresponds to the type of shift (i.e., night, morning, afternoon, rest) performed by a specific person on day $i$. A complete period of work is represented by the variables of the VARIABLES collection. In this context the group constraint expresses for a person:

- The number of periods of consecutive night-shift during a complete period of work.
- The total number of night-shift during a complete period of work.
- The maximum number of allowed consecutive night-shift.
- The minimum number of days, which do not correspond to a night-shift, between two consecutive sequences of night-shift.

Remark

For this constraint we use the possibility to express directly more than one constraint on the parameters of the final graph we want to obtain. For more propagation, it is crucial to keep this in one single constraint, since strong relations relate the different parameters of a graph. This constraint is very similar to the group constraint introduced in CHIP, except that here, the MIN_DIST and MAX_DIST constraints apply also for the two borders: we cannot start or end with a group of $k$ consecutive variables that take their values outside VALUES and such that $k$ is less than MIN_DIST or $k$ is greater than MAX_DIST.
See also

common keyword: change_continuity (timetabling constraint, sequence),
global_contiguity (sequence),
group_skip_isolated_item (timetabling constraint, sequence),
multi_global_contiguity (sequence),
pattern, stretch_circuit (timetabling constraint),
stretch_path (timetabling constraint, sequence).

shift of concept: consecutive_groups_of_ones.

used in graph description: in, not_in.

Keywords

characteristic of a constraint: automaton, automaton with counters.
combinatorial object: sequence.
constraint network structure: alpha-acyclic constraint network(2),
alpha-acyclic constraint network(3).
constraint type: timetabling constraint.
final graph structure: connected component, vpartition, consecutive loops are connected.
miscellaneous: obscure.
modelling: functional dependency.
### Arc input(s)
VARIABLES

### Arc generator
- `PATH↦collection(variables1,variables2)`
- `LOOP↦collection(variables1,variables2)`

### Arc arity
- 2

### Arc constraint(s)
- `in(variables1.var, VALUES)`
- `in(variables2.var, VALUES)`

### Graph property(ies)
- `NCC = NGROUP`
- `MIN_NCC = MIN_SIZE`
- `MAX_NCC = MAX_SIZE`
- `NVERTEX = NVAL`

### Graph model
We use two graph constraints for modelling the group constraint: a first one for specifying the constraints on `NGROUP`, `MIN_SIZE`, `MAX_SIZE` and `NVAL`, and a second one for stating the constraints on `MIN_DIST` and `MAX_DIST`. In order to generate the initial graph related to the first graph constraint we use:

- The arc generators `PATH` and `LOOP`.
- The binary constraint `variables1.var ∈ VALUES ∧ variables2.var ∈ VALUES`.

On the first graph constraint of the Example slot this produces an initial graph depicted in part (A) of Figure 5.290. We use `PATH LOOP` and the binary constraint `variables1.var ∈ VALUES ∧ variables2.var ∈ VALUES` in order to catch the two following situations:

- A binary constraint has to be used in order to get the notion of group: Consecutive variables that take their value in VALUES.
- If we only use `PATH` then we would lose the groups that are composed from one single variable since the predecessor and the successor arc would be destroyed; this is why we use also the `LOOP` arc generator.

Part (B) of Figure 5.290 shows the final graph associated with the first graph constraint of the Example slot. Since we use the `NVERTEX` graph property, the vertices of the final graph are stressed in bold. In addition, since we use the `MIN_NCC` and the `MAX_NCC` graph properties, we also show the smallest and largest connected components of the final graph.

The group constraint of the Example slot holds since:
Figure 5.290: Initial and final graph of the group constraint
• The final graph of the first graph constraint has two connected components. Therefore the number of groups $\text{NGROUP}$ is equal to two.

• The number of vertices of the smallest connected component of the final graph of the first graph constraint is equal to 1. Therefore $\text{MIN\_SIZE}$ is equal to 1.

• The number of vertices of the largest connected component of the final graph of the first graph constraint is equal to 2. Therefore $\text{MAX\_SIZE}$ is equal to 2.

• The number of vertices of the smallest connected component of the final graph of the second graph constraint is equal to 2. Therefore $\text{MIN\_DIST}$ is equal to 2.

• The number of vertices of the largest connected component of the final graph of the second graph constraint is equal to 4. Therefore $\text{MAX\_DIST}$ is equal to 4.

• The number of vertices of the final graph of the first graph constraint is equal to three. Therefore $\text{NVAL}$ is equal to 3.
Automaton

Figures 5.291, 5.293, 5.294, 5.296, 5.297 and 5.299 depict the different automata associated with the group constraint. For the automata that respectively compute $\text{NGROUP}$, $\text{MIN_SIZE}$, $\text{MAX_SIZE}$, $\text{MIN_DIST}$, $\text{MAX_DIST}$ and $\text{NVAL}$ we have a 0-1 signature variable $S_i$ for each variable $\text{VAR}_i$ of the collection $\text{VARIABLES}$. The following signature constraint links $\text{VAR}_i$ and $S_i$: $\text{VAR}_i \in \text{VALUES} \Leftrightarrow S_i$.

![Automaton Diagram]

Figure 5.291: Automaton for the $\text{NGROUP}$ parameter of the group constraint

![Hypergraph Diagram]

Figure 5.292: Hypergraph of the reformulation corresponding to the automaton of the $\text{NGROUP}$ parameter of the group constraint
Figure 5.293: Automaton for the MIN\_SIZE parameter of the group constraint

Figure 5.294: Automaton for the MAX\_SIZE parameter of the group constraint

Figure 5.295: Hypergraphs of the reformulations corresponding to the automata of the MIN\_SIZE and MAX\_SIZE parameters of the group constraint
Figure 5.296: Automaton for the MIN\_DIST parameter of the group constraint

Figure 5.297: Automaton for the MAX\_DIST parameter of the group constraint

Figure 5.298: Hypergraphs of the reformulations corresponding to the automata of the MIN\_DIST and MAX\_DIST parameters of the group constraint
Figure 5.299: Automaton for the NVAL parameter of the group constraint

Figure 5.300: Hypergraph of the reformulation corresponding to the automaton of the NVAL parameter of the group constraint
5.158  \textbf{group\_skip\_isolated\_item}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
\textbf{Origin} & Derived from \textit{group}. \\
\hline
\textbf{Constraint} & \textit{group\_skip\_isolated\_item} \begin{pmatrix} 
NGROUP, \\
MIN\_SIZE, \\
MAX\_SIZE, \\
NVAL, \\
VARIABLES, \\
VALUES \end{pmatrix} \\
\hline
\textbf{Arguments} & \begin{array}{c}
\text{NGROUP} : \text{dvar} \\
\text{MIN\_SIZE} : \text{dvar} \\
\text{MAX\_SIZE} : \text{dvar} \\
\text{NVAL} : \text{dvar} \\
\text{VARIABLES} : \text{collection(\text{var} \rightarrow \text{dvar})} \\
\text{VALUES} : \text{collection(\text{val} \rightarrow \text{int})} \\
\end{array} \\
\hline
\textbf{Restrictions} & \begin{array}{c}
\text{NGROUP} \geq 0 \\
\text{MIN\_SIZE} \geq 0 \\
\text{MAX\_SIZE} \geq \text{MIN\_SIZE} \\
\text{NVAL} \geq \text{MAX\_SIZE} \\
\text{NVAL} \geq \text{NGROUP} \\
\text{NVAL} \leq |\text{VARIABLES}| \\
\text{required}(\text{VARIABLES}, \text{var}) \\
\text{required}(\text{VALUES}, \text{val}) \\
\text{distinct}(\text{VALUES}, \text{val}) \\
\end{array} \\
\hline
\end{tabular}
\caption{}
\end{table}

Let \( n \) be the number of variables of the collection \text{VARIABLES}. Let \( X_i, X_{i+1}, \ldots, X_j \) \((1 \leq i < j \leq n)\) be consecutive variables of the collection of variables \text{VARIABLES} such that the following conditions apply:

- All variables \( X_i, \ldots, X_j \) take their value in the set of values \text{VALUES},
- \( i = 1 \) or \( X_{i-1} \) does not take a value in \text{VALUES},
- \( j = n \) or \( X_{j+1} \) does not take a value in \text{VALUES}.

\textbf{Purpose}

We call such a set of variables a \textit{group}. The constraint \textit{group\_skip\_isolated\_item} is true if all the following conditions hold:

- There are exactly \text{NGROUP} groups of variables,
- The number of variables of the smallest group is \text{MIN\_SIZE},
- The number of variables of the largest group is \text{MAX\_SIZE},
- The number of variables that take their value in the set of values \text{VALUES} is equal to \text{NVAL}. 

## Example

\[
\begin{pmatrix}
\{\text{var} - 2, \\
\text{var} - 8, \\
\text{var} - 1, \\
\text{var} - 7, \\
\text{var} - 4, \\
\text{var} - 5, \\
\text{var} - 1, \\
\text{var} - 1, \\
\text{var} - 1 \}
\end{pmatrix}, \\
\langle 0, 2, 4, 6, 8 \rangle
\]

Given the fact that groups are formed by even values in \{0, 2, 4, 6, 8\} (i.e., values expressed by the VALUES collection), and the fact that isolated even values are ignored, the \textit{group skip isolated item} constraint holds since:

- Its first argument, \text{NGROUP}, is set to value 1 since the sequence 2 8 1 7 4 5 1 1 1 contains only one group of even values involving more than one even value (i.e., group 2 8).
- Its second and third arguments, \text{MIN\_SIZE} and \text{MAX\_SIZE}, are both set to 2 since the only group of even values with more than one even value involves two values (i.e., group 2 8).
- The fourth argument, \text{NVAL}, is fixed to 2 since it corresponds to the total number of even values belonging to groups involving more than one even value (i.e., value 4 is discarded since it is an isolated even value of the sequence 2 8 1 7 4 5 1 1 1).

## Typical

<table>
<thead>
<tr>
<th>Condition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{NGROUP} &gt; 0</td>
<td></td>
</tr>
<tr>
<td>\text{MIN_SIZE} &gt; 0</td>
<td></td>
</tr>
<tr>
<td>\text{NVAL} &gt; \text{MAX_SIZE}</td>
<td></td>
</tr>
<tr>
<td>\text{NVAL} &gt; \text{NGROUP}</td>
<td></td>
</tr>
<tr>
<td>\text{NVAL} &lt; \text{</td>
<td>VALUES</td>
</tr>
<tr>
<td>\text{</td>
<td>VALUES</td>
</tr>
<tr>
<td>\text{range}(\text{VALUES_var}) &gt; 1</td>
<td></td>
</tr>
<tr>
<td>\text{</td>
<td>VALUES</td>
</tr>
<tr>
<td>\text{</td>
<td>VALUES</td>
</tr>
</tbody>
</table>

## Symmetries

- Items of VARIABLES can be \textit{reversed}.
- Items of VALUES are \textit{permutable}.
- An occurrence of a value of VARIABLES\_var that belongs to VALUES\_val (resp. does not belong to VALUES\_val) can be \textit{replaced} by any other value in VALUES\_val (resp. not in VALUES\_val).

## Arg. properties

- \textit{Functional dependency}: \text{NGROUP} determined by VARIABLES and VALUES.
- \textit{Functional dependency}: \text{MIN\_SIZE} determined by VARIABLES and VALUES.
- \textit{Functional dependency}: \text{MAX\_SIZE} determined by VARIABLES and VALUES.
- \textit{Functional dependency}: \text{NVAL} determined by VARIABLES and VALUES.
**Usage**

This constraint is useful in order to specify rules about how rest days should be allocated to a person during a period of \( n \) consecutive days. In this case VALUES are the codes for the rest days (perhaps one single value) and VARIABLES corresponds to the amount of work done during \( n \) consecutive days. We can then express a rule like: in a month one should have at least 4 periods of at least 2 rest days (isolated rest days are not counted as rest periods).

**See also**

- common keyword: change_continuity, group, stretch_path (timetabling constraint, sequence).
- used in graph description: in.

**Keywords**

- characteristic of a constraint: automaton, automaton with counters.
- combinatorial object: sequence.
- constraint type: timetabling constraint.
- final graph structure: strongly connected component.
- miscellaneous: obscure.
Arc input(s) VARIABLES
Arc generator $CHAIN \rightarrow \text{collection}(\text{variables1}, \text{variables2})$
Arc arity 2
Arc constraint(s)
- $\text{in}(\text{variables1}.\text{var}, \text{VALUES})$
- $\text{in}(\text{variables2}.\text{var}, \text{VALUES})$

Graph property(ies)
- $\text{NSCC} = \text{NGROUP}$
- $\text{MIN_NSCC} = \text{MIN_SIZE}$
- $\text{MAX_NSCC} = \text{MAX_SIZE}$
- $\text{NVERTEX} = \text{NVAL}$

Graph model
We use the $CHAIN$ arc generator in order to produce the initial graph. In the context of the Example slot, this creates the graph depicted in part (A) of Figure 5.301. We use $CHAIN$ together with the arc constraint $\text{variables1}.\text{var} \in \text{VALUES} \land \text{variables2}.\text{var} \in \text{VALUES}$ in order to skip the isolated variables that take a value in VALUES that we do not want to count as a group. This is why, on the example, value 4 is not counted as a group.

Part (B) of Figure 5.301 shows the final graph associated with the Example slot. The group skip isolated item constraint of the Example slot holds since:

- The final graph contains one strongly connected component. Therefore the number of groups is equal to one.
- The unique strongly connected component of the final graph contains two vertices. Therefore $\text{MIN_SIZE}$ and $\text{MAX_SIZE}$ are both equal to 2.
- The number of vertices of the final graph is equal to two. Therefore $\text{NVAL}$ is equal to 2.
Figure 5.301: Initial and final graph of the group_skip_isolated_item constraint
Automaton

Figures 5.302, 5.304, 5.305 and 5.307 depict the different automata associated with the `group_skip_isolated_item` constraint. For the automata that respectively compute `NGROUP`, `MIN_SIZE`, `MAX_SIZE` and `NVAL` we have a 0-1 signature variable $S_i$ for each variable $VAR_i$ of the collection `VARIABLES`. The following signature constraint links $VAR_i$ and $S_i$: $VAR_i \in VALUES \iff S_i$.

![Automaton diagram](image)

Figure 5.302: Automaton for the `NGROUP` parameter of the `group_skip_isolated_item` constraint

![Hypergraph diagram](image)

Figure 5.303: Hypergraph of the reformulation corresponding to the automaton of the `NGROUP` parameter of the `group_skip_isolated_item` constraint
Figure 5.304: Automaton for the MIN_SIZE parameter of the group_skip_isolated_item constraint
Figure 5.305: Automaton for the `MAX_SIZE` parameter of the `group_skip_isolated_item` constraint

Figure 5.306: Hypergraphs of the reformulations corresponding to the automata of the `MIN_SIZE` and `MAX_SIZE` parameters of the `group_skip_isolated_item` constraint
\( \text{MAX}_\text{NSCC}, \text{MIN}_\text{NSCC}, \text{NSCC}, \text{NVERTEX} \quad \text{CHAIN} \quad \text{AUTOMATON} \)

**Figure 5.307:** Automaton for the \( \text{NVAL} \) parameter of the \text{group\_skip\_isolated\_item} constraint

**Figure 5.308:** Hypergraph of the reformulation corresponding to the automaton of the \( \text{NVAL} \) parameter of the \text{group\_skip\_isolated\_item} constraint
### 5.159  \( \texttt{gt} \)

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Arithmetic.</td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>( \texttt{gt}(\texttt{VAR1}, \texttt{VAR2}) )</td>
</tr>
<tr>
<td><strong>Synonyms</strong></td>
<td>\texttt{rel}, \texttt{xgty}.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>\texttt{VAR1} : \texttt{dvar} &lt;br&gt; \texttt{VAR2} : \texttt{dvar}</td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>Enforce the fact that the first variable is strictly greater than the second variable.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>( (8, 1) ) &lt;br&gt;The ( \texttt{gt} ) constraint holds since ( 8 ) is strictly greater than ( 1 ).</td>
</tr>
<tr>
<td><strong>Symmetries</strong></td>
<td>• \texttt{VAR1} can be replaced by any value ( &gt; ) \texttt{VAR2}. &lt;br&gt;• \texttt{VAR2} can be replaced by any value ( &lt; ) \texttt{VAR1}.</td>
</tr>
<tr>
<td><strong>Systems</strong></td>
<td>\texttt{gt} in \texttt{Choco}, \texttt{rel} in \texttt{Gecode}, \texttt{xgty} in \texttt{JaCoP}, \texttt{#&gt;} in \texttt{SICStus}.</td>
</tr>
<tr>
<td><strong>See also</strong></td>
<td>\texttt{common keyword}: eq (binary constraint, arithmetic constraint). &lt;br&gt;\texttt{implies}: \texttt{geq, neq}. &lt;br&gt;\texttt{implies (if swap arguments)}: \texttt{lt}. &lt;br&gt;\texttt{negation}: \texttt{leq}.</td>
</tr>
<tr>
<td><strong>Keywords</strong></td>
<td>\texttt{constraint arguments}: binary constraint. &lt;br&gt;\texttt{constraint type}: predefined constraint, arithmetic constraint. &lt;br&gt;\texttt{filtering}: arc-consistency.</td>
</tr>
</tbody>
</table>
### 5.160 highest_peak

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Derived from peak.</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td><code>highest_peak(HEIGHT, VARIABLES)</code></td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>HEIGHT : dvar, VARIABLES : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>HEIGHT ≥ 0, VARIABLES.var ≥ 0, required(VARIABLES.var)</td>
<td></td>
</tr>
</tbody>
</table>

A variable $V_k$ ($1 < k < m$) of the sequence of variables $\text{VARIABLES} = V_1, \ldots, V_m$ is a **peak** if and only if there exists an $i$ ($1 < i < k$) such that $V_{i-1} < V_i$ and $V_i = V_{i+1} = \ldots = V_k$ and $V_k > V_{k+1}$. HEIGHT is the maximum value of the peak variables. If no such variable exists HEIGHT is equal to 0.

**Example**

\[
\begin{pmatrix}
\text{var} - 1, \\
\text{var} - 1, \\
\text{var} - 4, \\
\text{var} - 8, \\
\text{var} - 6, \\
\text{var} - 2, \\
\text{var} - 7, \\
\text{var} - 1
\end{pmatrix}
\]

The `highest_peak` constraint holds since 8 is the maximum peak of the sequence 1 1 4 8 6 2 7 1.

![Figure 5.309: The sequence and its highest peak](image-url)
**Typical**

- \( \text{HEIGHT} > 0 \)
- \(|\text{VARIABLES}| > 2\)
- \(\text{range}(\text{VARIABLES}.\text{var}) > 1\)

**Symmetry**

- Items of \(\text{VARIABLES}\) can be reversed.

**See also**

- **common keyword**: deepest_valley, peak (*sequence*).

**Keywords**

- **characteristic of a constraint**: automaton, automaton with counters.
- **combinatorial object**: sequence.
- **constraint network structure**: sliding cyclic(1) constraint network(2).
Automaton

Figure 5.310 depicts the automaton associated with the highest peak constraint. To each pair of consecutive variables \((\text{VAR}_i, \text{VAR}_{i+1})\) of the collection VARIABLES corresponds a signature variable \(S_i\). The following signature constraint links \(\text{VAR}_i, \text{VAR}_{i+1}\) and \(S_i\):

\[
\text{VAR}_i > \text{VAR}_{i+1} \iff S_i = 0 \land \text{VAR}_i = \text{VAR}_{i+1} \iff S_i = 1 \land \text{VAR}_i < \text{VAR}_{i+1} \iff S_i = 2.
\]

Figure 5.310: Automaton of the highest peak constraint

Figure 5.311: Hypergraph of the reformulation corresponding to the automaton of the highest peak constraint
### 5.161  imply

<table>
<thead>
<tr>
<th>Description</th>
<th>Links</th>
<th>Automaton</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Logic</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>imply(VAR, VARIABLES)</td>
<td></td>
</tr>
<tr>
<td><strong>Synonyms</strong></td>
<td>rel, ifthen.</td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>VAR : dvar, VARIABLES : collection(var − dvar)</td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>VAR ≥ 0, VAR ≤ 1,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>Let VARIABLES be a collection of 0-1 variables VAR₁, VAR₂. Enforce VAR = (VAR₁ ⇒ VAR₂).</td>
<td></td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td></td>
<td>(1, (0, 0))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1, (0, 1))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0, (1, 0))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1, (1, 1))</td>
</tr>
<tr>
<td><strong>Symmetry</strong></td>
<td>All occurrences of 0 in VAR and in VARIABLES.var can be set to 1.</td>
<td></td>
</tr>
<tr>
<td><strong>Arg. properties</strong></td>
<td>Functional dependency: VAR determined by VARIABLES.</td>
<td></td>
</tr>
<tr>
<td><strong>Systems</strong></td>
<td>reifiedLeftImp in Choco, rel in Gecode, ifthenbool in JaCoP, #&gt;= in SICStus.</td>
<td></td>
</tr>
<tr>
<td><strong>See also</strong></td>
<td>common keyword: and, equivalent, nand, nor, or, xor (Boolean constraint).</td>
<td></td>
</tr>
<tr>
<td><strong>Keywords</strong></td>
<td>characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>constraint arguments: pure functional dependency.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>constraint network structure: Berge-acyclic constraint network.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>constraint type: Boolean constraint.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>filtering: arc-consistency.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>modelling: functional dependency.</td>
<td></td>
</tr>
</tbody>
</table>
Automaton

Figure 5.312 depicts the automaton associated with the `imply` constraint. To the first argument `VAR` of the `imply` constraint corresponds the first signature variable. To each variable `VAR_i` of the second argument `VARIABLES` of the `imply` constraint corresponds the next signature variable. There is no signature constraint.

![Automaton Diagram]

Figure 5.312: Automaton of the `imply` constraint

![Hypergraph Diagram]

Figure 5.313: Hypergraph of the reformulation corresponding to the automaton of the `imply` constraint
### 5.162  in

<table>
<thead>
<tr>
<th><strong>DESCRIPTION</strong></th>
<th><strong>LINKS</strong></th>
<th><strong>GRAPH</strong></th>
<th><strong>AUTOMATON</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Domain definition.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>in(VAR, VALUES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Synonyms</strong></td>
<td>dom, in_set, member.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>VAR : dvar</td>
<td>VALUES : collection(val−int)</td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>Enforce the domain variable VAR to take a value within the values described by the VALUES collection.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>(3, (1, 3))</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Typical</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Symmetries</strong></td>
<td>Items of VALUES are permutable.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VAR can be set to any value of VALUES.val.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>One and the same constant can be added to VAR as well as to the val attribute of all items of VALUES.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Arg. properties</strong></td>
<td>Extensible wrt. VALUES.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Remark</strong></td>
<td>Entailment occurs immediately after posting this constraint.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Systems</strong></td>
<td>member in Choco, rel in Gecode, dom in Gecode, in in JaCoP, member in MiniZinc, in in SICStus, in_set in SICStus.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Used in</strong></td>
<td>among, cardinality.atmost_partition, group, group.skip.isolated.item, in_same_partition, open_among.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
See also

**common keyword:** domain (*domain definition*), in_interval, in_same_partition, in_set (*value constraint*).

**implied by:** maximum, minimum.

**implies:** between_min_max.

**negation:** not_in.

**Keywords**

**characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint, derived collection.

**constraint arguments:** unary constraint.

**constraint network structure:** centered cyclic(1) constraint network(1).

**constraint type:** value constraint.

**filtering:** arc-consistency.

**modelling:** included, domain definition.
**Derived Collection**

\[
\text{col}([\text{VARIABLES(var-dvar)}, \text{item(var-VAR)})]
\]

**Arc input(s)**

VARIABLES VALUES

**Arc generator**

\( \text{PRODUCT} \rightarrow \text{collection}([\text{variables}, \text{values}]) \)

**Arc arity**

2

**Arc constraint(s)**

\( \text{variables}.\text{var} = \text{values}.\text{val} \)

**Graph property(ies)**

\( \text{NARC} = 1 \)

**Graph model**

Parts (A) and (B) of Figure 5.314 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the unique arc of the final graph is stressed in bold.

![Graph model](image)

(A) (B)

Figure 5.314: Initial and final graph of the in constraint

**Signature**

Since all the val attributes of the VALUES collection are distinct and because of the arc constraint \( \text{variables}.\text{var} = \text{values}.\text{val} \) the final graph contains at most one arc. Therefore we can rewrite \( \text{NARC} = 1 \) to \( \text{NARC} \geq 1 \) and simplify \( \text{NARC} \) to \( \text{NARC} \).
Automaton

Figure 5.315 depicts the automaton associated with the \texttt{in} constraint. Let $\text{VAL}_i$ be the \texttt{val} attribute of the $i^{th}$ item of the $\text{VALUES}$ collection. To each pair $(\text{VAR}, \text{VAL}_i)$ corresponds a 0-1 signature variable $S_i$, as well as the following signature constraint: $\text{VAR} = \text{VAL}_i \Leftrightarrow S_i$.

![Automaton Diagram](image)

Figure 5.315: Automaton of the \texttt{in} constraint

![Hypergraph Diagram](image)

Figure 5.316: Hypergraph of the reformulation corresponding to the automaton of the \texttt{in} constraint
5.163 in_interval

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>in_interval(VAR, LOW, UP)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>dom, in.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VAR : dvar, LOW : int, UP : int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restriction</td>
<td>LOW ≤ UP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>Enforce the domain variable VAR to take a value within the interval [LOW, UP].</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>(3, 2, 5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Typical</td>
<td>LOW &lt; UP, VAR &gt; LOW, VAR &lt; UP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symmetries</td>
<td>LOW can be decreased.</td>
<td>UP can be increased.</td>
<td>An occurrence of a value of VAR can be replaced by any other value in [LOW, UP].</td>
</tr>
<tr>
<td>Remark</td>
<td>Entailment occurs immediately after posting this constraint. The in_interval constraint is referenced under the name dom in Gecode.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Systems</td>
<td>member in Choco, dom in Gecode, in in JaCoP, in in SICStus.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>See also</td>
<td>common keyword: domain, in (domain definition).</td>
<td>generalisation: in_interval_reified (reified version), in_intervals (single interval replaced by a set of intervals), in_set (interval replaced by set variable).</td>
<td></td>
</tr>
<tr>
<td>Keywords</td>
<td>characteristic of a constraint: automaton, automaton without counters, reified automaton constraint, derived collection.</td>
<td>constraint arguments: unary constraint.</td>
<td></td>
</tr>
</tbody>
</table>
constraint network structure: Berge-acyclic constraint network.
constraint type: value constraint.
filtering: arc-consistency.
modelling: interval, domain definition.
Derived Collections

\[
col(VARIABLE\rightarrow\text{collection}(\text{var}\rightarrow\text{dvar}), [\text{item(\text{var}\rightarrow\text{VAR})}])
\]

\[
col(\text{INTERVAL}\rightarrow\text{collection}([\text{low}\rightarrow\text{INT}, \text{up}\rightarrow\text{INT}]), [\text{item(\text{low}\rightarrow\text{LOW}, \text{up}\rightarrow\text{UP})}])
\]

Arc input(s) \hspace{1cm} \text{VARIABLE INTERVAL}

Arc generator \hspace{1cm} \text{PRODUCT} \rightarrow \text{collection}(\text{variable, interval})

Arc arity \hspace{1cm} 2

Arc constraint(s) \hspace{1cm}

\begin{itemize}
  \item \text{variable.var} \geq \text{interval.low}
  \item \text{variable.var} \leq \text{interval.up}
\end{itemize}

Graph property(ies) \hspace{1cm} \text{NARC}=1

Graph model

Parts (A) and (B) of Figure 5.317 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the unique arc of the final graph is stressed in bold.

![Graph Model](image)

Figure 5.317: Initial and final graph of the \text{in\_interval} constraint
Automaton

Figure 5.318 depicts the automaton associated with the `in_interval` constraint. We have one single 0-1 signature variable $S$ as well as the following signature constraint: $VAR \geq LOW \land VAR \leq UP \iff S$.

![Automaton Diagram](attachment:image.png)

Figure 5.318: Automaton of the `in_interval` constraint

![Hypergraph Diagram](attachment:image.png)

Figure 5.319: Hypergraph of the reformulation corresponding to the automaton of the `in_interval` constraint
## 5.164 in_interval_reified

<table>
<thead>
<tr>
<th>Description</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Reified version of <code>in_interval</code>.</td>
</tr>
<tr>
<td>Constraint</td>
<td><code>in_interval_reified(VAR, LOW, UP, B)</code></td>
</tr>
<tr>
<td>Synonyms</td>
<td><code>dom_reified, in_reified</code>.</td>
</tr>
<tr>
<td>Arguments</td>
<td>VAR : dvar</td>
</tr>
<tr>
<td>Restrictions</td>
<td>LOW (\leq) UP</td>
</tr>
<tr>
<td>Purpose</td>
<td>Enforce the following equivalence, VAR ∈ [LOW, UP] ⇔ B.</td>
</tr>
<tr>
<td>Example</td>
<td>(3, 2, 5, 1)</td>
</tr>
</tbody>
</table>

The `in_interval_reified` constraint holds since:

- Its first argument VAR = 3 is greater than or equal to its second argument LOW = 2 and less than or equal to its third argument UP = 5 (i.e., \(3 \in [2, 5]\)).
- The corresponding Boolean variable B is set to 1 since condition \(3 \in [2, 5]\) holds.

**Typical**

- VAR \(\not=\) LOW
- VAR \(\not=\) UP
- LOW \(<\) UP

**Symmetries**

- An occurrence of a value of VAR that belongs to [LOW, UP] (resp. does not belong to [LOW, UP]) can be replaced by any other value in [LOW, UP] (resp. not in [LOW, UP]).
- One and the same constant can be added to VAR, LOW and UP.

**Reformulation**

The `in_interval_reified` constraint can be reformulated in terms of linear constraints. For convenience, we rename VAR to \(x\), LOW to \(l\), UP to \(u\), and B to \(y\). The constraint is decomposed into the following conjunction of constraints:

\[
x \geq l \iff y_1, \\
x \leq u \iff y_2, \\
y_1 \land y_2 \iff y.
\]

We show how to encode these constraints with linear inequalities. The first constraint, i.e., \(x \geq l \iff y_1\) is encoded by posting one of the following three constraints:
\[
\begin{align*}
\begin{cases}
\text{a)} & \text{if } \overline{x} \geq l : & y_1 = 1, \\
\text{b)} & \text{if } \overline{x} < l : & y_1 = 0, \\
\text{c)} & \text{otherwise} : & x \geq (l - \overline{x}) \cdot y_1 + \overline{x} \land x \leq (\overline{x} - l + 1) \cdot y_1 + l - 1.
\end{cases}
\end{align*}
\]

On the one hand, cases a) and b) correspond to situations where one can fix \(y_1\), no matter what value will be assigned to \(x\). On the other hand, in case c), \(y_1\) can take both values 0 or 1 depending on the value assigned to \(x\). As shown by Figure 5.320, all possible solutions for the pair of variables \((x, y_1)\) satisfy the following two linear inequalities \(x \geq (l - \overline{x}) \cdot y_1 + \overline{x}\) and \(x \leq (\overline{x} - l + 1) \cdot y_1 + l - 1\). The first inequality discards all points that are above the line that goes through the two extreme solution points \((\overline{x}, 0)\) and \((l, 1)\), while the second one removes all points that are below the line that goes through the two extreme solution points \((l - 1, 0)\) and \((\overline{x}, 1)\).

![Figure 5.320: Illustration of the reformulation of the reified constraint \(x \geq l \iff y_1\) with two linear inequalities](image)

The second constraint, i.e., \(x \leq u \iff y_2\) is encoded by posting one of the following three constraints:

\[
\begin{align*}
\begin{cases}
\text{d)} & \text{if } \overline{x} \leq u : & y_2 = 1, \\
\text{e)} & \text{if } \overline{x} > u : & y_2 = 0, \\
\text{f)} & \text{otherwise} : & x \leq (u - \overline{x}) \cdot y_2 + \overline{x} \land x \geq (\overline{x} - u + 1) \cdot y_2 + u + 1.
\end{cases}
\end{align*}
\]

On the one hand, cases d) and e) correspond to situations where one can fix \(y_2\), no matter what value will be assigned to \(x\). On the other hand, in case f), \(y_2\) can take both value 0 or 1 depending on the value assigned to \(x\). As shown by Figure 5.321, all possible solutions for the pair of variables \((x, y_2)\) satisfy the following two linear inequalities \(x \leq (u - \overline{x}) \cdot y_2 + \overline{x}\) and \(x \geq (\overline{x} - u + 1) \cdot y_2 + u + 1\). The first inequality discards all points that are above the line that goes through the two extreme solution points \((\overline{x}, 0)\) and \((u, 1)\), while the second one removes all points that are below the line that goes through the two extreme solution points \((u + 1, 0)\) and \((\overline{x}, 1)\).
The third constraint, i.e., $y_1 \land y_2 \leftrightarrow y$ is encoded as:

$$\begin{cases}
g) & y \geq y_1 + y_2 - 1, \\
h) & y \leq y_1, \\
i) & y \leq y_2.
\end{cases}$$

Case g) handles the implication $y_1 \land y_2 \Rightarrow y$, while cases h) and i) take care of the other side $y \Rightarrow y_1 \land y_2$.

See also specialisation: in_interval.

uses in its reformulation: alldifferent (bound consistency preserving reformulation).

Keywords characteristic of a constraint: reified constraint.

constraint arguments: binary constraint.

constraint type: predefined constraint, value constraint.

filtering: arc-consistency.
Figure 5.321: Illustration of the reformulation of the reified constraint $x \leq u \iff y_2$ with two linear inequalities
5.165 in_intervalls

**DESCRIPTION**

**Origin**  
Domain definition.

**Constraint**  
in_intervalls(VAR, INTERVALS)

**Synonym**  
in.

**Arguments**  
VAR : dvar  
INTERVALS : collection(low-int, up-int)

**Restrictions**  
required(INTERVALS, [low, up])  
INTERVALS.low ≤ INTERVALS.up  
|INTERVALS| > 0

**Purpose**  
Enforce the domain variable VAR to take a value within one of the intervals specified by the collection of intervals INTERVALS.

**Example**  
\[
\begin{pmatrix}
5, (low - 1, up - 1),
low - 3, up - 5,
low - 8, up - 8
\end{pmatrix}
\]

The in_intervalls constraint holds since its first argument VAR = 5 belongs to the second intervals of the collection of intervals INTERVALS.

**Typical**  
|INTERVALS| > 1

**Symmetries**  
- Items of INTERVALS are permutable.  
- INTERVALS.low can be decreased.  
- INTERVALS.up can be increased.  
- One and the same constant can be added to VAR as well as to the low and up attributes of all items of INTERVALS.

**Arg. properties**  
Extensible wrt. INTERVALS.

**Remark**  
Entailment occurs immediately after posting this constraint.

**Systems**  
dom in Gecode, in in JaCoP, in in SICStus.

**See also**  
specialisation: in_interval (set of intervals replaced by single interval).

**Keywords**  
constraint arguments: unary constraint.  
constraint type: value constraint, predefined constraint.  
filtering: arc-consistency.  
modelling: interval, domain definition.
### 5.166 in_relation

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint</td>
<td>in_relation(VARIABLES, TUPLES_OF_VALS)</td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>case, extension, extensional, extensional_support, extensional_supportv, extensional_supportmdd, extensional_supportstr, feastupleac, table.</td>
<td></td>
</tr>
<tr>
<td>Types</td>
<td>TUPLE_OF_VARS : collection(var−dvar)</td>
<td>TUPLE_OF_VALS : collection(val−int)</td>
</tr>
<tr>
<td>Arguments</td>
<td>VARIABLES : TUPLE_OF_VARS</td>
<td>TUPLES_OF_VALS : collection(tuple − TUPLE_OF_VALS)</td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(TUPLE_OF_VARS, var)</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>Enforce the tuple of variables VARIABLES to take its value out of a set of tuples of values TUPLES_OF_VALS. The value of a tuple of variables (\langle V_1, V_2, \ldots, V_n \rangle) is a tuple of values (\langle U_1, U_2, \ldots, U_n \rangle) if and only if (V_1 = U_1 \land V_2 = U_2 \land \cdots \land V_n = U_n).</td>
<td></td>
</tr>
</tbody>
</table>
| Example     | \[
\begin{pmatrix}
(5,3,3), \\
tuple = (5,2,3), \\
tuple = (5,2,6), \\
tuple = (5,3,3)
\end{pmatrix}
\] | The in relation constraint holds since its first argument \((5,3,3)\) corresponds to the third item of the collection of tuples TUPLES_OF_VALS. |
| Typical     | \(|\text{TUPLE_OF_VARS}| > 1\) |       |
| Symmetries  | • Items of TUPLES_OF_VALS are permutable. | • Items of VARIABLES and TUPLES_OF_VALS.tuple are permutable (same permutation used). |
|             | • All occurrences of two distinct tuples of values in VARIABLES or TUPLES_OF_VALS.tuple can be swapped; all occurrences of a tuple of values in VARIABLES or TUPLES_OF_VALS.tuple can be renamed to any unused tuple of values. |       |
Arg. properties

Extensible wrt. TUPLES_OF_VALS.

Usage

Quite often some constraints cannot be easily expressed, neither by a formula, nor by a regular pattern. In this case one has to define the constraint by specifying in extension the combinations of allowed values.

Remark

The in_relation constraint is called extensional_support in JaCoP (http://www.jacop.eu/). Within SICStus Prolog the constraint can be applied to more than one single tuple of variables and is called table. Within [78] this constraint is called extension.

The in_relation constraint is called table in MiniZinc (http://www.g12.cs.mu.oz.au/minizinc/).

Systems


Used in

cond_lex_cost, cond_lex_greater, cond_lex_greatereq, cond_lex_less, cond_lex_leqeq.

See also

common keyword: element (data constraint).

See also: cond_lex_cost (COST parameter added).

See also: used in graph description: vec_eq_tuple.

Keywords

characteristic of a constraint: tuple, derived collection.

combinatorial object: relation.

constraint type: data constraint, extension.

filtering: arc-consistency.
Derived Collection

\[
\text{col}\left(\text{TUPLES_OF_VARS} - \text{collection}(\text{vec} - \text{TUPLE_OF_VARS}),
\begin{array}{l}
\text{item}(	ext{vec} - \text{VARIABLES})
\end{array}\right)
\]

Arc input(s)  TUPLES_OF_VARS TUPLES_OF_VALS

Arc generator  \(PRODUCT \rightarrow \text{collection}(\text{tuples_of_vars}, \text{tuples_of_vals})\)

Arc arity  2

Arc constraint(s)  \text{vec.eq.tuple}(\text{tuples_of_vars.vec}, \text{tuples_of_vals.tuple})

Graph property(ies)  \(\text{NARC} \geq 1\)

Graph model

Parts (A) and (B) of Figure 5.322 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the unique arc of the final graph is stressed in bold.

Figure 5.322: Initial and final graph of the in_relation constraint
5.167 in_same_partition

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Used for defining several entries of this catalog.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>in_same_partition(VAR1, VAR2, PARTITIONS)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>VALUES : collection(val–int)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VAR1 : dvar</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VAR2 : dvar</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PARTITIONS : collection(p – VALUES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>Enforce VAR1 and VAR2 to be respectively assigned to values (v_1) and (v_2) that both belong to a same partition of the collection PARTITIONS.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Example     | \[
|             | \(\begin{pmatrix}
|             | 6, 2, /p \in (1,3),
|             | p \in (4),
|             | p \in (2,6)\end{pmatrix}\) |
|             | The in_same_partition constraint holds since its first and second arguments \(VAR1 = 6\) and \(VAR2 = 2\) both belong to the third partition \((2,6)\) of its third argument PARTITIONS. |
| Typical     | \(VAR1 \neq VAR2\) |
| Symmetries  | • Arguments are permutable w.r.t. permutation \((VAR1, VAR2)\) (PARTITIONS). |
|             | • Items of PARTITIONS are permutable. |
|             | • Items of \(\text{PARTITIONS}.p\) are permutable. |
| Arg. properties | Extensible wrt. PARTITIONS. |
| Used in     | alldifferent_partition, balance_partition, change_partition, common_partition, nclass, same_partition, soft_same_partition_var, soft_used_by_partition_var, used_by_partition. |
| See also    | common keyword: alldifferent_partition (partition), in (value constraint). |
|             | used in graph description: in. |
Keywords

characteristic of a constraint: partition, automaton, automaton without counters, reified automaton constraint, derived collection.
constraint arguments: binary constraint.
constraint network structure: centered cyclic(2) constraint network(1).
constraint type: value constraint.
filtering: arc-consistency.
**Derived Collection**

\[
\text{col}\left(\text{VARIABLES−collection(var−dvar)}\right)\text{,}\n\text{item}(\text{var−VAR1}, \text{item}(\text{var−VAR2}))\text{.}
\]

**Arc input(s)**

VARIABLES PARTITIONS

**Arc generator**

\text{PRODUCT} \rightarrow \text{collection}(\text{variables.partitions})

**Arc arity**

2

**Arc constraint(s)**

\text{in}(\text{variables.var.partitions.p})

**Graph property(ies)**

- \text{NSOURCE} = 2
- \text{NSINK} = 1

**Graph model**

VAR1 and VAR2 are put together in the derived collection VARIABLES. Since both VAR1 and VAR2 should take their value in one of the partition depicted by the PARTITIONS collection, the final graph should have two sources corresponding respectively to VAR1 and VAR2. Since two, possibly distinct, values should be assigned to VAR1 and VAR2 and since these values belong to the same partition \(p\) the final graph should only have one sink. This sink corresponds in fact to partition \(p\).

Parts (A) and (B) of Figure 5.323 respectively show the initial and final graph associated with the Example slot. Since we both use the NSOURCE and NSINK graph properties, the source and sink vertices of the final graph are shown with a double circle.

![Figure 5.323: Initial and final graph of the in_same_partition constraint](image)

**Signature**

Note that the sinks of the initial graph cannot become sources of the final graph since isolated vertices are eliminated from the final graph. Since the final graph contains two sources it also includes one arc between a source and a sink. Therefore the minimum number of sinks of the final graph is equal to one. So we can rewrite \(\text{NSINK} = 1\) to \(\text{NSINK} \geq 1\) and simplify \(\text{NSINK}\) to \(\text{NSINK}\).
Automaton

Figure 5.324 depicts the automaton associated with the `in_same_partition` constraint. Let $\text{VALUES}_i$ be the $p$ attribute of the $i^{th}$ item of the `PARTITIONS` collection. To each triple $(\text{VAR1}, \text{VAR2}, \text{VALUES}_i)$ corresponds a 0-1 signature variable $S_i$ as well as the following signature constraint: $((\text{VAR1} \in \text{VALUES}_i) \land (\text{VAR2} \in \text{VALUES}_i)) \leftrightarrow S_i$.

![Automaton Diagram](image)

Figure 5.324: Automaton of the `in_same_partition` constraint

![Hypergraph Diagram](image)

Figure 5.325: Hypergraph of the reformulation corresponding to the automaton of the `in_same_partition` constraint
### 5.168 in_set

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Used for defining constraints with set variables.</td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td><code>in_set(VAL, SET)</code></td>
</tr>
<tr>
<td><strong>Synonyms</strong></td>
<td>dom, member.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td></td>
</tr>
<tr>
<td>VAL : dvar</td>
<td>SET : svar</td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>Constraint variable VAL to belong to set SET.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>(3, {1, 3})</td>
</tr>
</tbody>
</table>

**Remark**
When SET is fixed the `in_set` constraint is referenced under the name dom in Gecode.

**Systems**
member in Choco, rel in Gecode, dom in Gecode.

**Used in**
bipartite, clique, connected, cutset, dag, discrepancy, disj, inverse_set, k_cut, link_set_toBOOLEANS, open_alldifferent, open_among, open_atleast, open_atmost, open_global_cardinality, open_global_cardinality_low_up, path_from_to, proper_forest, roots, strongly_connected, sum, sum_set, symmetric, symmetric_cardinality, symmetric_gcc, tour.

**See also**
common keyword: in (value constraint).
specialisation: in_interval (set variable replaced by fixed interval).

**Keywords**
constraint arguments: constraint involving set variables.
constraint type: predefined constraint, value constraint.
modelling: included.
5.169 incomparable

**DESCRIPTION**

**Origin**
Inspired by incomparable rectangles.

**Constraint**
incomparable(VECTOR1, VECTOR2)

**Synonym**
incomparables.

**Arguments**
| VECTOR1 : collection(var–dvar) |
| VECTOR2 : collection(var–dvar) |

**Restrictions**
| required(VECTOR1.var) |
| required(VECTOR2.var) |
| |VECTOR1| ≥ 1 |
| |VECTOR2| ≥ 1 |
| |VECTOR1| = |VECTOR2|

**Purpose**
Enforce that when the components of VECTOR1 and VECTOR2 are ordered, and respectively denoted by SVECTOR1[1],SVECTOR1[2],...,SVECTOR1[|VECTOR1|] and SVECTOR2[1],SVECTOR2[2],...,SVECTOR2[|VECTOR2|], we neither have SVECTOR1[i].var ≤ SVECTOR2[i].var (for all i ∈ [1,|VECTOR1|]) nor have SVECTOR2[i].var ≤ SVECTOR1[i].var (for all i ∈ [1,|VECTOR1|]).

**Example**

```
((16, 2), (4, 11))
```

The incomparable constraint holds since 16 > 4 and 2 < 11.

**Typical**

| |VECTOR1| > 1 |

**Used in**
all_incomparable.

**See also**

system of constraints: all_incomparable.

**Keywords**
characteristic of a constraint: vector.
constraint type: predefined constraint.
## 5.170 increasing

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>KOALOG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>increasing(VARIABLES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>VARIABLES : collection(var−dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restriction</td>
<td>required(VARIABLES, var)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>The variables of the collection VARIABLES are increasing.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example

\[ ((1, 1, 4, 8)) \]

The **increasing** constraint holds since \( 1 \leq 1 \leq 4 \leq 8 \).

Typical

\[
\begin{align*}
|\text{VARIABLES}| & > 1 \\
\text{range} & > 1
\end{align*}
\]

Symmetry

One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

Contractible wrt. VARIABLES.

Systems

increasingNValue in Choco, rel in Gecode, increasing in MiniZinc.

Used in

increasing_global_cardinality, increasing_nvalue, increasing_sum.

See also

common keyword: precedence, strictly.decreasing (*order constraint*).

comparison swapped: decreasing.

implied by:

- all.equal,
- increasing_global_cardinality,
- increasing_nvalue (remove NVAL parameter from increasing_nvalue),
- increasing_sum (remove SUM parameter from increasing_sum),
- strictly.increasing.

implies: no_peak, no_valley.

uses in its reformulation: sort.permutation.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint network structure: sliding cyclic(1) constraint network(1).

constraint type: decomposition, order constraint.

filtering: arc-consistency.
Arc input(s) VARIABLES
Arc generator $PATH \rightarrow \text{collection}(\text{variables1}, \text{variables2})$
Arc arity 2
Arc constraint(s) $\text{variables1}.\text{var} \leq \text{variables2}.\text{var}$
Graph property(ies) $\text{NARC}= |\text{VARIABLES}| - 1$

Graph model Parts (A) and (B) of Figure 5.326 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

Figure 5.326: Initial and final graph of the increasing constraint
Automaton

Figure 5.327 depicts the automaton associated with the increasing constraint. To each pair of consecutive variables \((\text{VAR}_i, \text{VAR}_{i+1})\) of the collection \(\text{VARIABLES}\) corresponds a 0-1 signature variable \(S_i\). The following signature constraint links \(\text{VAR}_i, \text{VAR}_{i+1}\) and \(S_i\):

\[
\text{VAR}_i \leq \text{VAR}_{i+1} \Leftrightarrow S_i.
\]

\[
Q_0 = s \quad \text{VAR}_1 \quad\quad\quad\quad s \quad\quad\quad\quad \text{VAR}_n = s
\]

Figure 5.327: Automaton of the increasing constraint

\[
Q_0 = s \quad Q_1 \quad\quad\quad\quad Q_2 \quad\quad\quad\quad Q_{n-1} = s
\]

Figure 5.328: Hypergraph of the reformulation corresponding to the automaton of the increasing constraint
## 5.171 increasing_global_cardinality

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Conjoin <code>global_cardinality_low_up</code> and <code>increasing</code>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td><code>increasing_global_cardinality(VARIABLES, VALUES)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td><code>increasing_global_cardinality_low_up</code>, <code>increasing_gcc</code>, <code>increasing_gcc_low_up</code>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td><code>VARIABLES</code> : <code>collection(var−dvar)</code></td>
<td><code>VALUES</code> : <code>collection(val−int, omin−int, omax−int)</code></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td><code>required(VARIABLES.var)</code>&lt;br&gt;<code>increasing(VARIABLES)</code>&lt;br&gt;`</td>
<td>VALUES</td>
<td>&gt; 0<code>&lt;br&gt;</code>required(VALUES, [val, omin, omax])<code>&lt;br&gt;</code>distinct(VALUES, val)<code>&lt;br&gt;</code>VALUES.omin ≥ 0<code>&lt;br&gt;</code>VALUES.omax ≤</td>
</tr>
<tr>
<td>Purpose</td>
<td>The variables of the collection <code>VARIABLES</code> are increasing. In addition, each value <code>VALUES[i].val</code> (1 ≤ i ≤</td>
<td>VALUES</td>
<td>) should be taken by at least <code>VALUES[i].omin</code> and at most <code>VALUES[i].omax</code> variables of the <code>VARIABLES</code> collection.</td>
</tr>
</tbody>
</table>

### Example

\[
\begin{pmatrix}
\langle 3, 3, 6, 8 \rangle.
\langle \text{val} − 3, \text{omin} − 2, \text{omax} − 3, \rangle
\langle \text{val} − 5, \text{omin} − 0, \text{omax} − 1, \rangle
\langle \text{val} − 6, \text{omin} − 1, \text{omax} − 2 \rangle
\end{pmatrix}
\]

The `increasing_global_cardinality` constraint holds since:

- The values of the collection `⟨3, 3, 6, 8⟩` are sorted in increasing order.
- Values 3, 5 and 6 are respectively used 2 (2 ≤ 2 ≤ 3), 0 (0 ≤ 0 ≤ 1) and 1 (1 ≤ 1 ≤ 2) times within the collection `⟨3, 3, 6, 8⟩` and since no constraint was specified for value 8.

### Typical

- `|VARIABLES| > 1`<br>`range(VARIABLES.var) > 1`<br>`|VALUES| > 1`<br>`VALUES.omin ≤ |VARIABLES|`<br>`VALUES.omax > 0`<br>`VALUES.omax ≤ |VARIABLES|`<br>`|VARIABLES| > |VALUES|`
Symmetry

Items of VALUES are permutable.

Usage

This constraint can be used in order to break symmetry in the context of the following pattern. We have a matrix $M$ of variables with the same constraint on each row and a $\text{global_cardinality}_\text{low_up}$ constraint on each column. Beside lexicographically ordering the rows of $M$ with a $\text{lex_chain_less}$ constraint, one can also state an $\text{increasing_global_cardinality}$ on the first column of $M$ in order to improve propagation on the corresponding variables.

Reformulation

The $\text{increasing_global_cardinality}$ constraint can be expressed in term of a conjunction of a $\text{global_cardinality}_\text{low_up}$ and an $\text{increasing}$ constraints. Even if we achieve arc-consistency on these two constraints this hinders propagation as shown by the following small example.

We have two variables $X$ and $Y$ ($X \leq Y$), which both take their values in the set $\{2, 3\}$. In addition, assume that the minimum number of occurrences of values 0, 1 and 2 are respectively equal to 0, 1 and 1. Similarly assume that, the maximum number of occurrences of values 0, 1 and 2 are respectively equal to 1, 1 and 2. The reformulation does not reduce the domain of variables $X$, $Y$ in any way, while the automaton described in the Automaton slot fixes $X$ to 2 and $Y$ to 3.

See also

$\text{implies}$: $\text{global_cardinality}_\text{low_up}$, $\text{increasing}$.
$\text{related}$: $\text{ordered_global_cardinality}$.

Keywords

application area: assignment.
characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.
constraint network structure: Berge-acyclic constraint network.
constraint type: value constraint, order constraint.
filtering: arc-consistency.
symmetry: symmetry, matrix symmetry.
For all items of VALUES:

<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>SELF↦collection(variables)</td>
</tr>
<tr>
<td>Arc arity</td>
<td>1</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>variables.var = VALUES.val</td>
</tr>
</tbody>
</table>
| Graph property(ies) | • NVERTEX ≥ VALUES.omin  
                       • NVERTEX ≤ VALUES.omax |

**Graph model**

Since we want to express one unary constraint for each value we use the “For all items of VALUES” iterator. Part (A) of Figure 5.329 shows the initial graphs associated with each value 3, 5 and 6 of the VALUES collection of the Example slot. Part (B) of Figure 5.329 shows the two corresponding final graphs respectively associated with values 3 and 6 that are both assigned to the variables of the VARIABLES collection (since value 5 is not assigned to any variable of the VARIABLES collection the final graph associated with value 5 is empty). Since we use the NVERTEX graph property, the vertices of the final graphs are stressed in bold.

![Initial and final graphs](image)

Figure 5.329: Initial and final graph of the increasing global cardinality constraint
Automaton

A first systematic approach for creating an automaton that only recognises the solutions of the increasing\_global\_cardinality constraint could be to:

- First, create an automaton that recognises the solutions of the increasing constraint.
- Second, create an automaton that recognises the solutions of the global\_cardinality\_low\_up constraint.
- Third, make the product of the two previous automata and minimise the resulting automaton.

However this approach is not going to scale well in practice since the automaton associated with the global\_cardinality\_low\_up constraint may have a too big size. Therefore we propose an approach where we directly construct in one single step the automaton that only recognises the solutions of the increasing\_global\_cardinality constraint. Note that we do not have any formal proof that the resulting automaton is always minimum.

Without loss of generality, we assume that:

- All items of the VALUES collection are sorted in increasing value on the attribute val.
- All the potential values of the variables of the VARIABLES collection are included within the set of values of the collection VALUES (i.e., the val attribute).\(^6\)

Before defining the states of the automaton, we first need to introduce the following notion. A value VALUES\([v].\)val is constrained by its maximum number of occurrences if and only if VALUES\([v].\)omax \(\leq 1 \lor\) VALUES\([v].\)omax \(< |\text{VARIABLES}| - \sum_{u=1, u \neq v}^{|\text{VALUES}|} \text{VALUES}\([u].\)omin.\(^7\)

Let \(V\) denote the set of constrained values (i.e., their indexes within the collection VALUES) by their respective maximum number of occurrences.

After determining the set \(V\), the omax attribute of each potential value is normalised in the following way:

- For an unconstrained value VALUES\([v].\)val we reset VALUES\([v].\)omax to \(\max(1, \text{VALUES}[v].\)omin).
- For a constrained value VALUES\([v].\)val we reset VALUES\([v].\)omax to 1 if its current value is smaller than 1.

We are now in position to introduce the states of the automaton.

The \(1 + \sum_{v=1, v \in V}^{|\text{VALUES}|} \text{VALUES}[v].\)omax + \(\sum_{u=1, u \notin V}^{|\text{VALUES}|} \text{VALUES}[u].\)omin states of the automaton that only accepts solutions of the increasing\_global\_cardinality constraint are defined in the following way:

- For the \(v^{th}\) item of the collection VALUES we have:
  - If \(v \in V\), VALUES\([v].\)omax states labelled by \(s_{voc} (1 \leq o \leq \text{VALUES}[v].\)omax).
  - If \(v \notin V\), VALUES\([v].\)omin states labelled by \(s_{voc} (1 \leq o \leq \text{VALUES}[v].\)omin).
- We have an initial state labelled by \(s_{00}\).

\(^6\)If this is not the case, we can include these values within the VALUES collection and set their minimum and maximum number of occurrences to 0 and \(|\text{VARIABLES}| - \sum_{u=1}^{|\text{VALUES}|} \text{VALUES}[u].\)omin.

\(^7\)When VALUES\([v].\)omax \(\leq 1\) we cannot reduce the number of states related to value VALUES\([v].\)val and we therefore consider that we are in the constrained case.
Terminal states correspond to those states $s_{v,0}$ such that, both (1) $o$ is greater than or equal to $\text{VALUES}[v].\text{min}$, and (2) there is no value item $\text{VALUES}[w]$ ($w > v$) such that $\text{VALUES}[w].\text{min} > 0$. Transitions are defined in the following way:

- There is an arc, labelled by $\text{VALUES}[v].\text{val}$, from the initial state $s_{0,0}$ to every state $s_{0,1}$ where $\text{VALUES}[v]$ is an item for which all values $\text{VALUES}[u].\text{val}$ strictly less than $\text{VALUES}[v].\text{val}$ verify the condition $\text{VALUES}[u].\text{min} = 0$.

- For each value $\text{VALUES}[v].\text{val}$ constrained by its maximum number of occurrences (i.e., $v \in \mathcal{V}$), there is an arc, labelled by $\text{VALUES}[v].\text{val}$, from the state $s_{0,k}$ to the state $s_{0,k+1}$ for all $k$ in $[1, \text{VALUES}[v].\text{max} - 1]$.

- For each value $\text{VALUES}[v].\text{val}$ unconstrained by its maximum number of occurrences (i.e., $v \notin \mathcal{V}$), there is an arc, labelled by $\text{VALUES}[v].\text{val}$, from the state $s_{0,k}$ to the state $s_{0,k}$ for all $k$ in $[1, \text{VALUES}[v].\text{min} - 1]$. There is also a loop, labelled by $\text{VALUES}[v].\text{val}$, from state $s_{0,k}$ to the state $s_{0,k}$ for $k = \text{VALUES}[v].\text{min}$.

- For each value $\text{VALUES}[v].\text{val}$ constrained by its maximum number of occurrences (i.e., $v \in \mathcal{V}$), there is an arc, labelled by $\text{VALUES}[w].\text{val}$, from state $s_{v,k}$ to state $s_{v,1}$ ($v < w$) for all $k$ in $[\text{VALUES}[v].\text{min}, \text{VALUES}[v].\text{max}]$ and for all $w$ such that $\forall u \in [v + 1, w - 1]: \text{VALUES}[u].\text{min} = 0$.

- For each value $\text{VALUES}[v].\text{val}$ unconstrained by its maximum number of occurrences (i.e., $v \notin \mathcal{V}$), there is an arc, labelled by $\text{VALUES}[w].\text{val}$, from state $s_{v,k}$ to state $s_{w,1}$ ($v < w$) for $k = \text{VALUES}[v].\text{min}$ and for all $w$ such that $\forall u \in [v + 1, w - 1]: \text{VALUES}[u].\text{min} = 0$.

Figure 5.330 depicts the automaton associated with the increasing global cardinality constraint of the Example slot. For this purpose we assume without loss of generality that we have four decision variables that all take their potential values within interval $[3, 8]$. Consequently, values 4, 7 and 8 are first added to the items of the VALUES collection. Both values 3 and 6 are unconstrained by their respective maximum number of occurrences. Therefore their max attributes are respectively reduced to 2 and 1. All other values, namely values 4, 5, 7 and 8, are constrained values. The increasing global cardinality constraint holds since the corresponding sequence of visited states, $s_{0,0}$ $s_{1,1}$ $s_{1,2}$ $s_{0,1}$ $s_{0,0}$, ends up in a terminal state (i.e., terminal states are depicted by thick circles in the figure). Note that non initial states are first indexed by the position of an item within the VALUES collection, and not by the value itself (e.g., within $s_{1,2}$ the 1 designates value 3). For instance state $s_{1,1}$ depicts the fact that the automaton has already recognised one single occurrence of value 3, while $s_{1,2}$ corresponds to the fact that the automaton has already seen at least two occurrences of value 3.

\footnote{The at least comes from the loop on state $s_{1,2}$.}
Figure 5.330: Automaton of the increasing global cardinality constraint of the Example slot: the path corresponding to the solution $(3, 3, 6, 8)$ is depicted by thick arcs.
### 5.172 increasing_nvalue

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Conjoin nvalue and increasing.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>increasing_nvalue(NVAL, VARIABLES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>NVAL : dvar&lt;br&gt; VARIABLES : collection(var−dvar)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Restrictions| NVAL ≥ min(1, |VARIABLES|)
              | NVAL ≤ |VARIABLES|
              | required(VARIABLES, var)
              | increasing(VARIABLES) |       |           |
| Purpose     | The variables of the collection VARIABLES are increasing. In addition, NVAL is the number of distinct values taken by the variables of the collection VARIABLES. |       |           |
| Example     | (2, (6, 6, 8, 8, 8)) |       |           |

The increasing_nvalue constraint (see Figure 5.331 for a graphical representation) holds since:

- The values of the collection ⟨6, 6, 8, 8, 8⟩ are sorted in increasing order.
- NVAL = 2 is set to the number of distinct values occurring within the collection ⟨6, 6, 8, 8, 8⟩.

![Diagram](image)

**Figure 5.331:** The solution associated with the example

### Typical

| VARIABLES| > 1
| range(VARIABLES.var) | > 1 |

### Symmetry

One and the same constant can be added to the var attribute of all items of VARIABLES.
Functional dependency: NVAL determined by VARIABLES.

A complete filtering algorithm in a linear time complexity over the sum of the domain sizes is described in [43].

The increasing_nvalue constraint can be expressed in term of a conjunction of a nvalue and an increasing constraints (i.e., a chain of non strict inequality constraints on adjacent variables of the collection VARIABLES). But as shown by the following example, $V_1 \in [1, 2], V_2 \in [1, 2], V_1 \leq V_2, \text{nvalue}(2, (V_1, V_2))$, this hinders propagation (i.e., the unique solution $V_1 = 1, V_2 = 2$ is not directly obtained after stating all the previous constraints).

A better reformulation achieving arc-consistency uses the \texttt{seq_bin} constraint [290] that we now introduce. Given $N$ a domain variable, $X$ a sequence of domain variables, and $C$ and $B$ two binary constraints, \texttt{seq_bin}(N, X, C, B) holds if (1) $N$ is equal to the number of $C$-stretches in the sequence $X$, and (2) $B$ holds on any pair of consecutive variables in $X$. A $C$-stretch is a generalisation of the notion of stretch introduced by G. Pesant [285], where the equality constraint is made explicit by replacing it by a binary constraint $C$, i.e., a $C$-stretch is a maximal length subsequence of $X$ for which the binary constraint $C$ is satisfied on consecutive variables. increasing_nvalue(NVAL, VARIABLES) can be reformulated as \texttt{seq_bin}(NVAL, VARIABLES, =, $\leq$).

increasingNValue in Choco.

See also: increasing (remove NVAL parameter from increasing_nvalue), nvalue.

related: increasing_nvalue_chain.

shift of concept: ordered_nvector (variable replaced by vector and $\leq$ replaced by lex_lesseq).

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint network structure: Berge-acyclic constraint network.

constraint type: counting constraint, value partitioning constraint, order constraint.

filtering: arc-consistency.

final graph structure: strongly connected component, equivalence.

modelling: number of distinct equivalence classes, number of distinct values, functional dependency.

symmetry: symmetry.
Arc input(s)  VARIABLES
Arc generator  \textit{CLIQUE} \rightarrow \textit{collection}(\textit{variables1, variables2})
Arc arity 2
Arc constraint(s)  \textit{variables1.var} = \textit{variables2.var}
Graph property(ies)  \textit{NSCC} = \textit{NVAL}
Graph class \textit{EQUIVALENCE}

Graph model

Parts (A) and (B) of Figure 5.332 respectively show the initial and final graph associated with the Example slot. Since we use the NSCC graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a value that is assigned to some variables of the VARIABLES collection. The following values 6 and 8 are used by the variables of the VARIABLES collection.

Figure 5.332: Initial and final graph of the increasing \textit{nvalue} constraint
Automaton

A first systematic approach for creating an automaton that only recognises the solutions of the increasing_nvalue constraint could be to:

- First, create an automaton that recognises the solutions of the increasing constraint.
- Second, create an automaton that recognises the solutions of the nvalue constraint.
- Third, make the product of the two previous automata and minimise the resulting automaton.

However this approach is not going to scale well in practice since the automaton associated with the nvalue constraint has a too big size. Therefore we propose an approach where we directly construct in one single step the automaton that only recognises the solutions of the increasing_nvalue constraint. Note that we do not have any formal proof that the resulting automaton is always minimum.

Without loss of generality, assume that the collection of variables VARIABLES contains at least one variable (i.e., |VARIABLES| \geq 1). Let l, m, n, min and max respectively denote the minimum and maximum possible value of variable NVAL, the number of variables of the collection VARIABLES, the smallest value that can be assigned to the variables of VARIABLES, and the largest value that can be assigned to the variables of VARIABLES. Let s = max − min + 1 denote the total number of potential values. Clearly, the maximum number of distinct values that can be assigned to the variables of the collection VARIABLES cannot exceed the quantity d = min(m, n, s). The \frac{s(s+1)}{2} \geq \frac{(x-d)(x-d+1)}{2} + 1 states of the automaton that only accepts solutions of the increasing_nvalue constraint can be defined in the following way:

- We have an initial state labelled by \( s_{00} \).
- We have \frac{s(s+1)}{2} \geq \frac{(x-d)(x-d+1)}{2} + 1 states labelled by \( s_{ij} \). The first index \( i \) of a state \( s_{ij} \) corresponds to the number of distinct values already encountered, while the second index \( j \) denotes the the current value (i.e., more precisely the index of the current value, where the minimum value has index 1).

Terminal states depend on the possible values of variable NVAL and correspond to those states \( s_{ij} \) such that \( i \) is a possible value for variable NVAL. Note that we assume no further restriction on the domain of NVAL (otherwise the set of terminal states needs to be reduced in order to reflect the current set of possible values of NVAL). Three classes of transitions are respectively defined in the following way:

1. There is a transition, labelled by \( \text{min} + j - 1 \), from the initial state \( s_{00} \) to the state \( s_{ij} \) \((1 \leq j \leq s)\).
2. There is a loop, labelled by \( \text{min} + j - 1 \) for every state \( s_{ij} \) \((1 \leq i \leq d, i \leq j \leq s)\).
3. \( \forall i \in [1, d-1], \forall j \in [i, s], \forall k \in [j+1, s] \) there is a transition labelled by \( \text{min} + k - 1 \) from \( s_{ij} \) to \( s_{i+1,k} \).

We respectively have \( s \) transitions of class 1, \( \frac{(x+1)}{2} - \frac{(x-d)(x-d+1)}{2} \) transitions of class 2, and \( \frac{6}{2} \) transitions of class 3. Note that all states \( s_{ij} \) such that \( i + s - j < l \) can be discarded since they do not allow to reach the minimum number of distinct values required \( l \).

Part (A) of Figure 5.333 depicts the automaton associated with the increasing_nvalue constraint of the \textbf{Example} slot. For this purpose, we assume that variable NVAL...
is fixed to value 2 and that variables of the collection \textsc{variables} take their values within interval \([6, 8]\). Part (B) of Figure 5.333 represents the simplified automaton where all states that do not allow to reach a terminal state were removed. The \texttt{increasing\_global\_cardinality} constraint holds since the corresponding sequence of visited states, \(s_0 s_11 s_3 s_23 s_23 s_23\), ends up in a terminal state (i.e., terminal states are depicted by thick circles in the figure).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{automaton.png}
\caption{Automaton \texttt{A} – and simplified automaton – \texttt{B} – of the \texttt{increasing\_nvalue} constraint of the \texttt{Example} slot: the path corresponding to the solution \((6, 6, 8, 8, 8)\) is depicted by thick arcs.}
\end{figure}
### 5.173 increasing_nvalue_chain

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from <code>increasing_nvalue</code>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td><code>increasing_nvalue_chain(NVAL, VARIABLES)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>NVAL : <code>dvar</code></td>
<td>VARIABLES : <code>collection(b−dvar, var−dvar)</code></td>
<td></td>
</tr>
</tbody>
</table>
| Restrictions| NVAL \(\geq\) \(\min(1, |VARIABLES|)\)  
NVAL \(\leq\) | VARIABLES | required(VARIABLES, [b, var])  
VARIABLES.b \(\geq\) 0  
VARIABLES.b \(\leq\) 1 | | |

For each consecutive pair of items `VARIABLES[i]`, `VARIABLES[i + 1]` \((1 \leq i < |VARIABLES|)\) of the `VARIABLES` collection at least one of the following conditions hold:

1. `VARIABLES[i + 1].b = 0`,
2. `VARIABLES[i].var \leq VARIABLES[i + 1].var`.

In addition, `NVAL` is equal to number of pairs of variables `VARIABLES[i]`, `VARIABLES[i + 1]` \((1 \leq i < |VARIABLES|)\) plus one, which verify at least one of the following conditions:

1. `VARIABLES[i + 1].b = 0`,
2. `VARIABLES[i].var < VARIABLES[i + 1].var`.

Note that `VARIABLES[1].b` is not referenced at all in the previous definition (i.e., its value does not influence at all the values assigned to the other variables).

#### Example

\[
\begin{pmatrix}
  b & var - 2,
  b - 1 & var - 4,
  b - 1 & var - 4,
  b - 1 & var - 4,
  b - 1 & var - 4,
  b - 0 & var - 4,
  b - 1 & var - 8,
  b - 0 & var - 1,
  b - 0 & var - 7,
  b - 1 & var - 7
\end{pmatrix}
\]

The `increasing_nvalue_chain` constraint holds since:

1. The condition `VARIABLES[i + 1].b = 0 \lor VARIABLES[i].var \leq VARIABLES[i + 1].var` holds for every pair of adjacent items of the `VARIABLES` collection:
• For the pair \((\text{VARIABLES}_1\.\text{var}, \text{VARIABLES}_2\.\text{var})\) we have 
\[
\text{VARIABLES}_1\.\text{var} \leq \text{VARIABLES}_2\.\text{var} \ (2 \leq 4).
\]
• For the pair \((\text{VARIABLES}_2\.\text{var}, \text{VARIABLES}_3\.\text{var})\) we have 
\[
\text{VARIABLES}_2\.\text{var} \leq \text{VARIABLES}_3\.\text{var} \ (4 \leq 4).
\]
• For the pair \((\text{VARIABLES}_3\.\text{var}, \text{VARIABLES}_4\.\text{var})\) we have 
\[
\text{VARIABLES}_3\.\text{var} \leq \text{VARIABLES}_4\.\text{var} \ (4 \leq 4).
\]
• For the pair \((\text{VARIABLES}_4\.\text{var}, \text{VARIABLES}_5\.\text{var})\) we have 
\[
\text{VARIABLES}_5\.\text{b} = 0.
\]
• For the pair \((\text{VARIABLES}_5\.\text{var}, \text{VARIABLES}_6\.\text{var})\) we have 
\[
\text{VARIABLES}_5\.\text{var} \leq \text{VARIABLES}_6\.\text{var} \ (4 \leq 8).
\]
• For the pair \((\text{VARIABLES}_6\.\text{var}, \text{VARIABLES}_7\.\text{var})\) we have 
\[
\text{VARIABLES}_7\.\text{b} = 0.
\]
• For the pair \((\text{VARIABLES}_7\.\text{var}, \text{VARIABLES}_8\.\text{var})\) we have 
\[
\text{VARIABLES}_8\.\text{b} = 0.
\]
• For the pair \((\text{VARIABLES}_8\.\text{var}, \text{VARIABLES}_9\.\text{var})\) we have 
\[
\text{VARIABLES}_8\.\text{var} \leq \text{VARIABLES}_9\.\text{var} \ (7 \leq 7).
\]

2. \text{NVAL} is equal to number of pairs of variables \text{VARIABLES}_i,.\text{VARIABLES}_{i+1} \ (1 \leq i < |\text{VARIABLES}|) plus one which verify at least \text{VARIABLES}_i+1\.\text{b} = 0 \lor \text{VARIABLES}_i\.\text{var} < \text{VARIABLES}_{i+1}.\text{var}.\) Beside the plus one, the following five pairs contribute for 1 in \text{NVAL}:

• For the pair \((\text{VARIABLES}_1\.\var, \text{VARIABLES}_2\.\var)\) we have 
\[
\text{VARIABLES}_1\.\var \leq \text{VARIABLES}_2\.\var \ (2 < 4).
\]
• For the pair \((\text{VARIABLES}_4\.\var, \text{VARIABLES}_5\.\var)\) we have 
\[
\text{VARIABLES}_5\.\text{b} = 0.
\]
• For the pair \((\text{VARIABLES}_5\.\var, \text{VARIABLES}_6\.\var)\) we have 
\[
\text{VARIABLES}_5\.\var \leq \text{VARIABLES}_6\.\var \ (4 < 8).
\]
• For the pair \((\text{VARIABLES}_6\.\var, \text{VARIABLES}_7\.\var)\) we have 
\[
\text{VARIABLES}_7\.\text{b} = 0.
\]
• For the pair \((\text{VARIABLES}_7\.\var, \text{VARIABLES}_8\.\var)\) we have 
\[
\text{VARIABLES}_8\.\text{b} = 0.
\]

Typical

\[|\text{VARIABLES}| > 1\]
\[\text{range}(\text{VARIABLES.b}) > 1\]
\[\text{range}(\text{VARIABLES.var}) > 1\]

See also

related: increasing_nvalue, nvalue, ordered_nvector.

Keywords

constraint type: counting constraint, order constraint.

modelling: number of distinct values.
Graph model

Parts (A) and (B) of Figure 5.334 respectively show the initial and final graph associated with the second graph constraint of the Example slot. Since we use the NARC graph property the arcs of the final graph are stressed in bold.

Figure 5.334: Initial and final graph of the increasing_nvalue_chain constraint
Automaton

Without loss of generality, assume that the collection $\text{VARIABLES}$ contains at least one variable (i.e., $|\text{VARIABLES}| \geq 1$). Let $l, m, n, \min$ and $\max$ respectively denote the minimum and maximum possible value of variable $\text{NVAL}$, the number of items of the collection $\text{VARIABLES}$, the smallest value that can be assigned to $\text{VARIABLES}[i].\text{var}$ $(1 \leq i \leq n)$, and the largest value that can be assigned to $\text{VARIABLES}[i].\text{var}$ $(1 \leq i \leq n)$. Let $s = \max - \min + 1$ denote the total number of potential values. Clearly, the maximum value of $\text{NVAL}$ cannot exceed the quantity $d = \min(m, n)$. The states of the automaton that only accepts solutions of the increasing\_nvalue\_chain constraint can be defined in the following way:

- We have an initial state labelled by $s_{00}$.
- We have $d \cdot s$ states labelled by $s_{ij}$ $(1 \leq i \leq d, 1 \leq j \leq s)$.

Terminal states depend on the possible values of variable $\text{NVAL}$ and correspond to those states $s_{ij}$ such that $i$ is a possible value for variable $\text{NVAL}$. Note that we assume no further restriction on the domain of $\text{NVAL}$ (otherwise the set of terminal states needs to be reduced in order to reflect the current set of possible values of $\text{NVAL}$).

Transitions of the automaton are labelled by a pair of values $(\alpha, \beta)$ and correspond to a condition of the form $\text{VARIABLES}[i].b = \alpha \land \text{VARIABLES}[i].\text{var} = \beta$, $(1 \leq i \leq n)$. Characters $\ast$ and $+$ respectively represent all values in $\{0, 1\}$ and all values in $\{\min, \min + 1, \ldots, \max\}$. Four classes of transitions are respectively defined in the following way:

1. There is a transition, labelled by the pair $(\ast, \min + j - 1)$, from the initial state $s_{00}$ to the state $s_{1j}$ $(1 \leq j \leq s)$. We use the $\ast$ character since $\text{VARIABLES}[1].b$ is not use at all in the definition of the increasing\_nvalue\_chain constraint.
2. There is a loop, labelled by the pair $(1, \min + j - 1)$ for every state $s_{ij}$ $(1 \leq i \leq d, 1 \leq j \leq s)$.
3. $\forall i \in [1, d - 1], \forall j \in [1, s], \forall k \in [j + 1, s]$ there is a transition labelled by the pair $(1, \min + k - 1)$ from $s_{ij}$ to $s_{i+1,k}$.
4. $\forall i \in [1, d - 1], \forall j \in [1, s]$ there is a transition labelled by the pair $(0, +)$ from $s_{ij}$ to $s_{i+1,1}$. 
Figure 5.335: Automaton of the increasing_nvalue_chain constraint under the hypothesis that all variables are assigned a value in \{6, 7, 8\} and that NVAL is equal to 2. The character * on a transition corresponds to a 0 or to a 1 and the + corresponds to a 6, 7 or 8.
### 5.174 increasing_sum

**DESCRIPTION**

Conjoin increasing and sum_ctr.

**SYNONYMS**

increasing_sum_ctr, increasing_sum_eq.

**ARGUMENTS**

| VARIABLES : collection(var−dvar) |
| S : dvar |

**RESTRICTIONS**

required(VARIABLES.var)

increasing(VARIABLES)

**PURPOSE**

The variables of the collection VARIABLES are increasing. In addition, S is the sum of the variables of the collection VARIABLES.

**EXAMPLE**

\((3, 3, 6, 8), 20\)

The increasing_sum constraint holds since:

- The values of the collection \((3, 3, 6, 8)\) are sorted in increasing order.
- \(S = 20\) is set to the sum \(\langle 3 + 3 + 6 + 8 \rangle\).

**USAGE**

The increasing_sum constraint can be used for breaking some symmetries in bin packing problems. Given a set of \(n\) bins with the same maximum capacity, and a set of items each of them with a specific height, the problem is to pack all items in the bins. To break symmetry we order bins by increasing use. This is done by introducing a variable \(x_i\) \((0 \leq i < n)\) for each bin \(i\) giving its use, i.e., the sum of items heights assigned to bin \(i\), and by posting the following increasing_sum(\(\langle x_0, x_1, \ldots, x_{n-1} \rangle, s\)) where \(s\) denotes the sum of the heights of all the items to pack.

**ALGORITHM**

A linear time filtering algorithm achieving bound-consistency for the increasing_sum constraint is described in [293]. This algorithm was motivated by the fact that achieving bound-consistency on the inequality constraints and on the sum constraint independently hinders propagation, as illustrated by the following small example, where the maximum value of \(x_1\) is not reduced to 2: \(x_1 \in [1, 3], x_2 \in [2, 5], s \in [5, 6], x_1 < x_2, x_1 + x_2 = s\).

Given an increasing_sum(\(\langle x_0, x_1, \ldots, x_{n-1} \rangle, s\)) constraint, the bound-consistency algorithm consists of three phases:
1. A normalisation phase adjusts the minimum and maximum value of variables \(x_0, x_1, \ldots, x_{n-1}\) with respect to the chain of inequalities \(x_0 \leq x_1 \leq \ldots \leq x_{n-1}\). A forward phase adjusts the minimum value of \(x_1, x_2, \ldots, x_{n-1}\) (i.e., \(x_{i+1} \geq x_i\)), while a backward phase adjusts the maximum value of \(x_{n-2}, x_{n-1}, \ldots, x_0\) (i.e., \(x_{i-1} \leq x_i\)).

2. A phase restricts the minimum and maximum value of the sum variable \(s\) with respect to the chain of inequalities \(x_0 \leq x_1 \leq \ldots \leq x_{n-1}\) (i.e., \(s \geq \sum_{0 \leq i < n} x_i\) and \(\pi \leq \sum_{0 \leq i < n} x_i\)).

3. A final phase reduces the minimum and maximum value of variables \(x_0, x_1, \ldots, x_{n-1}\) both from the bounds of \(s\) and from the chain of inequalities. Without loss of generality we now focus on the pruning of the maximum value of variables \(x_0, x_1, \ldots, x_{n-1}\). For this purpose we first need to introduce the notion of the last intersecting index of a variable \(x_i\), denoted by \(last_i\). This corresponds to the greatest index in \([i + 1, n - 1]\) such that \(\pi_i > x_{last_i}\), or \(i\) if no such integer exists. Then the increase of the minimum value of \(s\) when \(x_i\) is equal to \(\pi_i\) is equal to \(\sum_{k \in [i, last_i]} (\pi_i - x_k)\). When this increase exceeds the available margin, i.e., \(\pi_i - \sum_{0 \leq i < n} x_i\), we update the maximum value of \(x_i\).

We illustrate a part of the final phase on the following example increasing sum \((x_0, x_1, x_2, x_3, x_4, x_5, s)\), where \(x_0 \in [2, 6], x_1 \in [4, 7], x_2 \in [4, 7], x_3 \in [5, 7], x_4 \in [6, 9], x_5 \in [7, 9]\) and \(s \in [28, 29]\). Observe that the domains are consistent with the first two phases of the algorithm, since,

1. the minimum (and maximum) values of variables \(x_0, x_1, x_2, x_3, x_4, x_5\) are increasing,
2. the sum of the minimum of the variables \(x_0, x_1, x_2, x_3, x_4, x_5\), i.e., 28 is less than or equal to the maximum value of \(s\),
3. the sum of the maximum of the variables \(x_0, x_1, x_2, x_3, x_4, x_5\), i.e., 45 is greater than or equal to the minimum value of \(s\).

Now, assume we want to know the increase of the minimum value of \(s\) when \(x_0\) is set to its maximum value 6. First we compute the last intersecting index of variable \(x_0\). Since \(x_4\) is the last variable for which the minimum value is less than or equal to maximum value of \(x_0\) we have \(last_0 = 4\). The increase is equal to \(\sum_{k \in [0, 4]} (\pi_0 - x_k) = (6 - 2) + (6 - 4) + (6 - 4) + (6 - 5) + (6 - 6) = 9\). Since it exceeds the margin \(29 - (2 + 4 + 5 + 6 + 7) = 1\), we have to reduce the maximum value of \(x_0\). How to do this incrementally is described in [293].

See also

common keyword: sum.ctr (sum).

implies: increasing.

Keywords

characteristic of a constraint: sum.

constraint type: predefined constraint, order constraint, arithmetic constraint.

filtering: bound-consistency.

symmetry: symmetry.
5.175  indexed_sum

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>N. Beldiceanu</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>indexed_sum(ITEMS, TABLE)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>ITEMS : (\text{collection}(\text{index-dvar}, \text{weight-dvar}))</td>
<td>TABLE : (\text{collection}(\text{index-int}, \text{summation-dvar}))</td>
</tr>
<tr>
<td>Restrictions</td>
<td>(</td>
<td>\text{ITEMS}</td>
</tr>
<tr>
<td></td>
<td>(\text{required}(\text{ITEMS}, [\text{index}, \text{weight}]))</td>
<td>(\text{required}(\text{TABLE}, [\text{index}, \text{summation}]))</td>
</tr>
<tr>
<td></td>
<td>(\text{ITEMS}.\text{index} \geq 1)</td>
<td>(\text{TABLE}.\text{index} \geq 1)</td>
</tr>
<tr>
<td></td>
<td>(\text{ITEMS}.\text{index} \leq</td>
<td>\text{TABLE}</td>
</tr>
<tr>
<td></td>
<td>(\text{increasing_seq}(\text{TABLE}, \text{index}))</td>
<td></td>
</tr>
</tbody>
</table>

**Purpose**

Given several items of the collection ITEMS (each of them having a specific fixed index as well as a weight that may be negative or positive), and a table TABLE (each entry of TABLE corresponding to a summation variable), assign each item to an entry of TABLE so that the sum of the weights of the items assigned to that entry is equal to the corresponding summation variable.

Example

\[
\begin{pmatrix}
\langle \text{index} - 3, \text{weight} - 4, \rangle \\
\langle \text{index} - 1, \text{weight} - 6, \rangle \\
\langle \text{index} - 3, \text{weight} - 1, \rangle \\
\langle \text{index} - 1, \text{summation} - 6, \rangle \\
\langle \text{index} - 2, \text{summation} - 0, \rangle \\
\langle \text{index} - 3, \text{summation} - 3, \rangle \\
\end{pmatrix}
\]

The indexed_sum constraint holds since the summation variables associated with each entry of TABLE are equal to the sum of the weights of the items assigned to the corresponding entry:

- \(\text{TABLE}[1].\text{summation} = \text{ITEMS}[2].\text{weight} = 6\) (since \(\text{TABLE}[1].\text{index} = \text{ITEMS}[2].\text{index} = 1\)),
- \(\text{TABLE}[2].\text{summation} = 0\) (since \(\text{TABLE}[2].\text{index} = 2\) does not occur as a value of the index attribute of an item of ITEMS),
- \(\text{TABLE}[3].\text{summation} = \text{ITEMS}[1].\text{weight} + \text{ITEMS}[3].\text{weight} = -4 + 1 = -3\) (since \(\text{TABLE}[3].\text{index} = \text{ITEMS}[1].\text{index} = \text{ITEMS}[3].\text{index} = 3\)).

**Typical**

\(|\text{ITEMS}| > 1\)
\(|\text{range(ITEMS.index)}| > 1\)
\(|\text{TABLE}| > 1\)
\(|\text{range(TABLE.summation)}| > 1\)
Symmetries

- Items of ITEMS are permutable.
- Items of TABLE are permutable.

Reformulation

The indexed_sum(ITEMS, TABLE) constraint can be expressed in term of a set of reified constraints and of $|TABLE|$ arithmetic constraints (i.e., scalar_product constraints).

1. For each item $ITEMS[i]$ ($i \in [1,|ITEMS|]$) and for each table entry $j$ ($j \in [1,|TABLE|]$) of TABLE we create a 0-1 variable $B_{ij}$ that will be set to 1 if and only if $ITEMS[i].index$ is fixed to $j$ (i.e., $B_{ij} \leftrightarrow ITEMS[i].index = j$).

2. For each entry $j$ of the table TABLE, we impose the sum $ITEMS[1].weight \cdot B_{1j} + ITEMS[2].weight \cdot B_{2j} + \ldots + ITEMS[|ITEMS|].weight \cdot B_{|ITEMS|j}$ to be equal to $TABLE[j].summation$.

See also

specialisation: bin_packing (negative contribution not allowed, effective use variable for each bin replaced by an overall fixed capacity), bin_packing_capa (negative contribution not allowed, effective use variable for each bin replaced by a fixed capacity for each bin).

used in graph description: sum_ctr.

Keywords

application area: assignment.
modelling: variable indexing, variable subscript.
For all items of TABLE:

<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>ITEMS TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>( PRODUCT \mapsto \text{collection}(\text{items}, \text{table}) )</td>
</tr>
<tr>
<td>Arc arity</td>
<td>2</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>items.index = table.index</td>
</tr>
</tbody>
</table>
| Sets          | SUCC \mapsto \begin{bmatrix}
|               | \text{source},
|               | \text{variables} \rightarrow \text{col}(\text{VARIABLES\text{-collection}(\text{var\text{-dvar})}}, \text{item}([\text{var} \rightarrow \text{ITEMS\text{-weight}}]) \end{bmatrix} |
| Constraint(s) on sets | \text{sum}_\text{ctr}(\text{variables}, =, \text{TABLE\text{-summation}}) |

**Graph model**

We enforce the \text{sum}_\text{ctr} constraint on the weight of the items that are assigned to the same entry. Within the context of the Example slot, part (A) of Figure 5.336 shows the initial graphs associated with entries 1, 2 and 3 (i.e., one initial graph for each item of the TABLE collection). Part (B) of Figure 5.336 shows the corresponding final graphs associated with entries 1 and 3. Each source vertex of the final graph can be interpreted as an item assigned to a specific entry of TABLE.

![Initial and final graph](image)
## 5.176 inflexion

### DESCRIPTION

**Origin**
N. Beldiceanu

**Constraint**
\[ \text{inflexion}(N, \text{VARIABLES}) \]

**Arguments**
- \( N : \text{dvar} \)
- \( \text{VARIABLES} : \text{collection}(\text{var} - \text{dvar}) \)

**Restrictions**
- \( N \geq 0 \)
- \( N \leq \max(0, |\text{VARIABLES}|-2) \)
- \( \text{required}(\text{VARIABLES}, \text{var}) \)

The number \( N \) is equal to the number of times that the following conjunctions of constraints hold:

- \( X_i \text{CTR} X_{i+1} \land X_i \neq X_{i+1} \),
- \( X_{i+1} = X_{i+2} \land \cdots \land X_{j-2} = X_{j-1} \),
- \( X_{j-1} \neq X_j \land X_{j-1} \text{CTR} X_j \),

where \( X_k \) is the \( k \)th item of the \( \text{VARIABLES} \) collection and \( 1 \leq i, i + 2 \leq j, j \leq n \) and \( \text{CTR} \) is \(< \text{ or } > \).

### LINKS

### AUTOMATON

### Purpose

The inflexion constraint holds since the sequence 1 1 4 8 8 2 7 1 contains three inflexions peaks that respectively correspond to values 8, 2 and 7.

### Example

\[
\begin{pmatrix}
\text{var} - 1, \\
\text{var} - 1, \\
\text{var} - 4, \\
\text{var} - 8, \\
\text{var} - 8, \\
\text{var} - 2, \\
\text{var} - 7, \\
\text{var} - 1
\end{pmatrix}
\]

### Typical

- \( N > 0 \)
- \(|\text{VARIABLES}| > 2\)
- \( \text{range}(\text{VARIABLES}.\text{var}) > 1 \)

### Symmetries

- Items of \( \text{VARIABLES} \) can be reversed.
- One and the same constant can be added to the \text{var} attribute of all items of \( \text{VARIABLES} \).

### Usage

Useful for constraining the number of inflexions of a sequence of domain variables.

### Remark

Since the arity of the arc constraint is not fixed, the inflexion constraint cannot be currently described. However, this would not hold anymore if we were introducing a slot that specifies how to merge adjacent vertices of the final graph.
See also

common keyword: global_contiguity, peak, valley (sequence).

Keywords

characteristic of a constraint: automaton, automaton with counters.
combinatorial object: sequence.
constraint network structure: sliding cyclic(1) constraint network(2).

Figure 5.337: The sequence 1 1 4 8 8 2 7 1 and its three inflexions
Figure 5.338 depicts the automaton associated with the inflexion constraint. To each pair of consecutive variables \( (\text{VAR}_i, \text{VAR}_{i+1}) \) of the collection \( \text{VARIABLES} \) corresponds a signature variable \( S_i \). The following signature constraint links \( \text{VAR}_i, \text{VAR}_{i+1} \) and \( S_i \):

\[
\begin{align*}
(\text{VAR}_i > \text{VAR}_{i+1} \Leftrightarrow S_i = 0) \land (\text{VAR}_i = \text{VAR}_{i+1} \Leftrightarrow S_i = 1) \land (\text{VAR}_i < \text{VAR}_{i+1} \Leftrightarrow S_i = 2).
\end{align*}
\]

Figure 5.338: Automaton of the inflexion constraint

Figure 5.339: Hypergraph of the reformulation corresponding to the automaton of the inflexion constraint
### 5.177 inside_sboxes

<table>
<thead>
<tr>
<th>Description</th>
<th>Links</th>
<th>Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometry, derived from [318]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Constraint**

\(\text{inside}\_\text{sboxes}(K, \text{DIMS}, \text{OBJECTS}, \text{SBOXES})\)

**Synonym**

inside.

**Types**

- VARIABLES : collection(v\text{-\textit{dvar}})
- INTEGERS : collection(v\text{-\textit{int}})
- POSITIVES : collection(v\text{-\textit{int}})

**Arguments**

- \(K\) : int
- DIMS : sint
- OBJECTS : collection(oid\text{-\textit{int}}, sid\text{-\textit{int}}, x \text{ VARIABLE\}}
- SBOXES : collection(sid\text{-\textit{int}}, t \text{ INTEGERS, } l \text{ POSITIVES})

**Restrictions**

- \(|\text{VARIABLES}| \geq 1\)
- \(|\text{INTEGERS}| \geq 1\)
- \(|\text{POSITIVES}| \geq 1\)
- \(\text{required(VARIABLES, }v)\)
- \(\text{required(INTEGERS, }v)\)
- \(\text{required(POSITIVES, }v)\)
- \(\text{POSITIVES} > 0\)
- \(K > 0\)
- \(\text{DIMS} \geq 0\)
- \(\text{DIMS} < K\)
- \(\text{increasing\_seq(OBJECTS, }\text{oid})\)
- \(\text{required(OBJECTS, }\text{oid}, \text{sid}, \text{x})\)
- \(\text{OBJECTS.oid} \geq 1\)
- \(\text{OBJECTS.oid} \leq |\text{OBJECTS}|\)
- \(\text{OBJECTS.sid} \geq 1\)
- \(\text{OBJECTS.sid} \leq |\text{SBOXES}|\)
- \(\text{SBOXES} \geq 1\)
- \(\text{required(SBOXES, }\text{sid}, t, l)\)
- \(\text{SBOXES.sid} \geq 1\)
- \(\text{SBOXES.sid} \leq |\text{SBOXES}|\)
- \(\text{do\_not\_overlap(SBOXES)}\)
Holds if, for each pair of objects \((O_i, O_j), i < j\), \(O_i\) is inside \(O_j\) with respect to a set of dimensions depicted by \(\text{DIMS}\). \(O_i\) and \(O_j\) are objects that take a shape among a set of shapes. Each shape is defined as a finite set of shifted boxes, where each shifted box is described by a box in a \(K\)-dimensional space at a given offset (from the origin of the shape) with given sizes. More precisely, a shifted box is an entity defined by its shape id \(\text{sid}\), shift offset \(\text{t}\), and sizes \(l\). Then, a shape is defined as the union of shifted boxes sharing the same shape id. An object is an entity defined by its unique object identifier \(\text{oid}\), shape id \(\text{sid}\) and origin \(x\).

An object \(O_i\) is inside an object \(O_j\) with respect to a set of dimensions depicted by \(\text{DIMS}\) if and only if, for all shifted boxes \(s_i\) associated with \(O_i\), there exists a shifted box \(s_j\) of \(O_j\) such that \(s_j\) is inside \(s_i\). A shifted box \(s_j\) is inside a shifted box \(s_i\) if and only if, for all dimensions \(d \in \text{DIMS}\), (1) the start of \(s_j\) in dimension \(d\) is strictly less than the start of \(s_i\) in dimension \(d\), and (2) the end of \(s_i\) in dimension \(d\) is strictly less than the end of \(s_j\) in dimension \(d\).

Figure 5.340 shows the objects of the example. Since \(O_1\) is inside \(O_2\) and \(O_3\), and since \(O_2\) is also inside \(O_3\), the inside_sboxes constraint holds.
Symmetries

- Items of SBOXES are permutable.
- Items of OBJECTS.x, SBOXES.t and SBOXES.1 are permutable (same permutation used).

Arg. properties

Suffix-contractible wrt. OBJECTS.

Remark

One of the eight relations of the Region Connection Calculus [318]. The constraint inside_sboxes is a restriction of the original relation since it requires that each box of an object is contained by one box of the other object.

See also

common keyword: contains_sboxes, coveredby_sboxes, covers_sboxes, disjoint_sboxes, equal_sboxes, meet_sboxes (rcc8), non_overlap_sboxes (geometrical constraint, logic), overlap_sboxes (rcc8).

Keywords

constraint type: logic.

gradient: geometrical constraint, rcc8.
Logic

- \text{origin}(O_1, S_1, D) \overset{\text{def}}{=} O_1.x(D) + S_1.t(D)
- \text{end}(O_1, S_1, D) \overset{\text{def}}{=} O_1.x(D) + S_1.t(D) + S_1.1(D)
- \text{inside}_sboxes(Dims, O_1, S_1, O_2, S_2) \overset{\text{def}}{=} 
  \forall D \in Dims 
  \left( \begin{array}{c} \text{origin}(O_2, S_2, D) < \\
   \text{origin}(O_1, S_1, D) \\
   \text{end}(O_1, S_1, D) < \\
   \text{end}(O_2, S_2, D) \end{array} \right)
- \text{inside}_sboxes(Dims, O_1, O_2) \overset{\text{def}}{=} 
  \forall S_1 \in \text{sboxes}([O_1.\text{sid}]) 
  \exists S_2 \in \text{sboxes}([O_2.\text{sid}]) 
  \left( \begin{array}{c}
   \text{inside}_sboxes(Dims, O_1, S_1, O_2, S_2) \\
   \text{inside}_sboxes(S_1, O_1, S_2, O_2) \end{array} \right)
- \text{all}_sboxes(Dims, OIDS) \overset{\text{def}}{=} 
  \forall O_1 \in \text{objects}(OIDS) 
  \forall O_2 \in \text{objects}(OIDS) 
  O_1.\text{oid} < \Rightarrow 
  O_2.\text{oid}
- \text{all}_sboxes(DIMENSIONS, OIDS)
5.178 int_value_precede

**Description**

**Origin**

[240]

**Constraint**

int_value.precede(S, T, VARIABLES)

**Synonyms**

precede, precedence, value.precede.

**Arguments**

S : int
T : int
VARIABLES : collection(var−dvar)

**Restrictions**

S ≠ T
required(VARIABLES, var)

**Purpose**

If value T occurs in the collection of variables VARIABLES then its first occurrence should be preceded by an occurrence of value S.

**Example**

(0, 1, ⟨4, 0, 6, 1, 0⟩)

The int_value.precede constraint holds since the first occurrence of value 0 precedes the first occurrence of value 1.

**Typical**

S < T
|VARIABLES| > 1
atleast(1, VARIABLES, S)
atleast(1, VARIABLES, T)

**Symmetries**

- An occurrence of a value of VARIABLES.var that is different from S and T can be replaced by any other value that is also different from S and T.
- All occurrences of values S and T can be swapped in S, T and VARIABLES.var.

**Arg. properties**

- Suffix-contractible wrt. VARIABLES.
- Aggregate: S(id), T(id), VARIABLES(union).

**Algorithm**

A filtering algorithm for maintaining value precedence is presented in [240]. Its complexity is linear to the number of variables of the collection VARIABLES.

**Systems**

precede in Gecode, value.precede in MiniZinc.

**See also**

generalisation: int_value_precede_chain(sequence of 2 values replaced by sequence of at least 2 values), set_value_precede (sequence of domain variables replaced by sequence of set variables).
Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint,

constraint network structure: Berge-acyclic constraint network.

constraint type: order constraint.

filtering: arc-consistency.

symmetry: symmetry, indistinguishable values, value precedence.
Automaton

Figure 5.341 depicts the automaton associated with the int_value_precede constraint. Let $\text{VAR}_i$ be the $i^{th}$ variable of the VARIABLES collection. To each triple $(S, T, \text{VAR}_i)$ corresponds a signature variable $S_i$ as well as the following signature constraint: $(\text{VAR}_i = S \iff S_i = 1) \land (\text{VAR}_i = T \iff S_i = 2) \land (\text{VAR}_i \neq S \land \text{VAR}_i \neq T \iff S_i = 3)$.

Figure 5.341: Automaton of the int_value_precede constraint

Figure 5.342: Hypergraph of the reformulation corresponding to the automaton of the int_value_precede constraint
5.179  \textbf{int\_value\_precede\_chain}  

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[240]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>\texttt{int_value_precede_chain(VALUES, VARIABLES)}</td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>\texttt{precede, precedence, value_precede_chain}</td>
<td></td>
</tr>
</tbody>
</table>
| Arguments   | \begin{align*}
\text{VALUES} & : \textit{collection(\text{var\_int})} \\
\text{VARIABLES} & : \textit{collection(\text{var\_dvar})}
\end{align*} |           |
| Restrictions| \begin{align*}
\text{required(VALUES, var)} \\
\text{distinct(VALUES, var)} \\
\text{required(VARIABLES, var)}
\end{align*} |           |

Purpose

Assuming \( n \) denotes the number of items of the \texttt{VALUES} collection, the following condition holds for every \( i \in [1, n - 1] \): When it is defined, the first occurrence of the \((i + 1)^{th}\) value of the \texttt{VALUES} collection should be preceded by the first occurrence of the \(i^{th}\) value of the \texttt{VALUES} collection.

Example

The \texttt{int\_value\_precede\_chain} constraint holds since within the sequence \(4, 0, 6, 1, 0\):

- The first occurrence of value 4 occurs before the first occurrence of value 0.
- The first occurrence of value 0 occurs before the first occurrence of value 1.

Typical

\begin{align*}
|VALUES| > 1 \\
\textit{strictly\_increasing}(VALUES) \\
|VARIABLES| > |VALUES| \\
\textit{range}(VARIABLES.var) > 1 \\
\textit{used\_by}(VARIABLES, VALUES)
\end{align*}

Symmetry

An occurrence of a value of \texttt{VARIABLES.var} that does not occur in \texttt{VALUES.var} can be replaced by any other value that also does not occur in \texttt{VALUES.var}.

Arg. properties

- \textbf{Contractible} wrt. \texttt{VALUES}.
- \textbf{Suffix-contractible} wrt. \texttt{VARIABLES}.
- \textbf{Aggregate}: \texttt{VALUES(id), VARIABLES(union)}. 
Usage

The `int_value_precede_chain` constraint is useful for breaking symmetries in graph colouring problems. We set a `int_value_precede_chain` constraint on all variables $V_1, V_2, \ldots, V_n$ associated with the vertices of the graph to colour, where we state that the first occurrence of colour $i$ should be located before the first occurrence of colour $i+1$ within the sequence $V_1, V_2, \ldots, V_n$.

Figure 5.343 illustrates the problem of colouring earth and mars from Thom Sulanke. Part (A) of Figure 5.343 provides a solution where the first occurrence of each value of $i$, $(i \in \{1, 2, \ldots, 8\})$ is located before the first occurrence of value $i+1$. This is obtained by using the following constraints:

\[
\begin{aligned}
& A \neq B, A \neq E, A \neq F, A \neq G, A \neq H, A \neq I, A \neq J, A \neq K, \\
& B \neq A, B \neq C, B \neq F, B \neq G, B \neq H, B \neq I, B \neq J, B \neq K, \\
& C \neq B, C \neq D, C \neq F, C \neq G, C \neq H, C \neq I, C \neq J, C \neq K, \\
& D \neq C, D \neq E, D \neq F, D \neq G, D \neq H, D \neq I, D \neq J, D \neq K, \\
& E \neq A, E \neq D, E \neq F, E \neq G, E \neq H, E \neq I, E \neq J, E \neq K, \\
& F \neq A, F \neq B, F \neq C, F \neq D, F \neq E, F \neq G, F \neq H, F \neq I, F \neq J, F \neq K, \\
& G \neq A, G \neq B, G \neq C, G \neq D, G \neq E, G \neq F, G \neq H, G \neq I, G \neq J, G \neq K, \\
& H \neq A, H \neq B, H \neq C, H \neq D, H \neq E, H \neq F, H \neq G, H \neq I, H \neq J, H \neq K, \\
& I \neq A, I \neq B, I \neq C, I \neq D, I \neq E, I \neq F, I \neq G, I \neq H, I \neq J, I \neq K, \\
& J \neq A, J \neq B, J \neq C, J \neq D, J \neq E, J \neq F, J \neq G, J \neq H, J \neq I, J \neq K, \\
& K \neq A, K \neq B, K \neq C, K \neq D, K \neq E, K \neq F, K \neq G, K \neq H, K \neq I, K \neq J, \\
\end{aligned}
\]

Part (B) provides a symmetric solution where the value precedence constraints between the pairs of values $(1, 2), (2, 3), (4, 5), (7, 8)$ and $(8, 9)$ are all violated (each violation is depicted by a dashed curve).

Remark

When we have more than one class of interchangeable values (i.e., a partition of interchangeable values) we can use one `int_value_precede_chain` constraint for breaking value symmetry in each class of interchangeable values. However it was shown in [414] that enforcing arc-consistency for such a conjunction of `int_value_precede_chain` constraints is NP-hard.

Algorithm

The 2004 reformulation [27] associated with the automaton of the Automaton slot achieves arc-consistency since the corresponding constraint network is a Berge-acyclic constraint network. Later on, another formulation into a sequence of ternary sliding constraints was proposed by [413]. It also achieves arc-consistency for the same reason.

Systems

`precede` in Gecode, `value_precede_chain` in MiniZinc.

See also

specialisation: `int_value_precede` (sequence of at least 2 values replaced by sequence of 2 values).

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint network structure: Berge-acyclic constraint network.

constraint type: order constraint.
Figure 5.343: Using the int_value_precede_chain constraint for breaking symmetries in graph colouring problems; there is a curve between the first occurrence of value $v$ ($1 \leq v \leq 8$) in the sequence of variables $A, B, C, D, E, F, G, H, I, J, K$, and the first occurrence of value $v + 1$ (a plain curve if the corresponding value precedence constraint holds, a dashed curve otherwise)
filtering: arc-consistency.
problems: graph colouring.
symmetry: symmetry, indistinguishable values, value precedence.
Automaton

Figure 5.344 depicts the automaton associated with the int_value_precede_chain constraint. Let $n$ and $m$ respectively denote the number of variables of the VARIABLES collection and the number of values of the VALUES collection. Let $\text{VAR}_i$ be the $i^{th}$ variable of the VARIABLES collection. Let $\text{VAL}_v$ ($1 \leq v \leq m$) denote the $v^{th}$ value of the VALUES collection.

![Diagram of automaton]

Figure 5.344: Automaton of the int_value_precede_chain constraint

We now show how to construct such an automaton systematically. For this purpose let us first introduce some notations:

- Without loss of generality we assume that we have at least two values (i.e., $m \geq 2$).
- Let $\mathcal{C}$ be the set of values that can be potentially assigned to a variable of the VARIABLES collection, but which do not belong to the values of the VALUES collection (i.e., $\mathcal{C} = (\text{dom}(\text{VAR}_1) \cup \text{dom}(\text{VAR}_2) \cup \ldots \cup \text{dom}(\text{VAR}_n)) - \{\text{VAL}_1, \text{VAL}_2, \ldots, \text{VAL}_m\} = \{w_1, w_2, \ldots, w_{|\mathcal{C}|}\}$.
The states and transitions of the automaton are respectively defined in the following way:

- We have \( m + 1 \) states labelled \( s_0, s_1, \ldots, s_m \) from which \( s_0 \) is the initial state. All states are terminal states.

- We have the following three sets of transitions:
  1. For all \( v \in [0, m - 1] \), a transition from \( s_v \) to \( s_{v+1} \) labelled by value \( \text{val}_{v+1} \). Each transition of this type will be triggered on the first occurrence of value \( \text{val}_{v+1} \) within the variables of the VARIABLES collection.
  2. For all \( v \in [1, m] \) and for all \( w \in [1, v] \), a self loop on \( s_v \) labelled by value \( \text{val}_w \). Such transitions encode the fact that we stay in the same state as long as we have a value that was already encountered.
  3. If the set \( C \) is not empty, then for all \( v \in [0, m] \) a self loop on \( s_v \) labelled by the fact that we take a value not in VALUES (i.e., a value in \( C \)). This models the fact that, encountering a value that does not belong to the set of values of the VALUES collection, leaves us in the same state.
5.180 interval_and_count

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[119]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>interval_and_count(ATMOST, COLOURS, TASKS, SIZE_INTERVAL)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>ATMOST : int, COLOURS : collection(val=int), TASKS : collection(origin=dvar, colour=dvar), SIZE_INTERVAL : int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>ATMOST $\geq 0$, required(COLOURS, val), distinct(COLOURS, val), required(TASKS, [origin, colour]), TASKS.origin $\geq 0$, SIZE_INTERVAL $&gt; 0$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

First consider the set of tasks of the TASKS collection, where each task has a specific colour that may not be initially fixed. Then consider the intervals of the form $[k \cdot \text{SIZE\_INTERVAL}, k \cdot \text{SIZE\_INTERVAL} + \text{SIZE\_INTERVAL} - 1]$, where $k$ is an integer. The interval_and_count constraint enforces that, for each interval $I_k$ previously defined, the total number of tasks, which both are assigned to $I_k$ and take their colour in COLOURS, does not exceed the limit ATMOST.

Example

First consider the set of tasks of the TASKS collection, where each task has a specific colour that may not be initially fixed. Then consider the intervals of the form $[k \cdot \text{SIZE\_INTERVAL}, k \cdot \text{SIZE\_INTERVAL} + \text{SIZE\_INTERVAL} - 1]$, where $k$ is an integer. The interval_and_count constraint enforces that, for each interval $I_k$ previously defined, the total number of tasks, which both are assigned to $I_k$ and take their colour in COLOURS, does not exceed the limit ATMOST.

Figure 5.346 shows the solution associated with the example. The constraint interval_and_count holds since, for each interval, the number of tasks taking colour 4 does not exceed the limit 2.

Figure 5.346: Solution with the use of each interval
Typical

\[
\begin{align*}
\text{ATMOST} & > 0 \\
\text{ATMOST} & < |\text{TASKS}| \\
|\text{COLOURS}| & > 0 \\
|\text{TASKS}| & > 1 \\
\text{range}(\text{TASKS}.\text{origin}) & > 1 \\
\text{range}(\text{TASKS}.\text{colour}) & > 1 \\
\text{SIZE}_\text{INTERVAL} & > 1
\end{align*}
\]

Symmetries

- ATMOST can be increased.
- Items of COLOURS are permutable.
- Items of TASKS are permutable.
- One and the same constant can be added to the origin attribute of all items of TASKS.
- An occurrence of a value of TASKS.origin that belongs to the \( k \)-th interval, of size \( \text{SIZE}_\text{INTERVAL} \), can be replaced by any other value of the same interval.
- An occurrence of a value of TASKS.colour that belongs to COLOURS.val (resp. does not belong to COLOURS.val) can be replaced by any other value in COLOURS.val (resp. not in COLOURS.val).

Arg. properties

- Contractible wrt. COLOURS.
- Contractible wrt. TASKS.

Usage

This constraint was originally proposed for dealing with timetabling problems. In this context the different intervals are interpreted as morning and afternoon periods of different consecutive days. Each colour corresponds to a type of course (i.e., French, mathematics). There is a restriction on the maximum number of courses of a given type each morning as well as each afternoon.

Remark

If we want to only consider intervals that correspond to the morning or to the afternoon we could extend the interval and count constraint in the following way:

- We introduce two extra parameters \( \text{REST} \) and \( \text{QUOTIENT} \) that correspond to non-negative integers such that \( \text{REST} \) is strictly less than \( \text{QUOTIENT} \).
- We add the following condition to the arc constraint:
  \[
  (\text{tasks1}.\text{origin}/\text{SIZE}_\text{INTERVAL}) \equiv \text{REST}(\text{mod } \text{QUOTIENT})
  \]

Now, if we want to express a constraint on the morning intervals, we set \( \text{REST} \) to 0 and \( \text{QUOTIENT} \) to 2.

Reformulation

Let \( K \) denote the index of the last possible interval where the tasks can be assigned: 
\[
K = \left\lfloor \max_{i \in [1, |\text{TASKS}|]} (\text{TASKS}[i].\text{origin})/\text{SIZE}_\text{INTERVAL} - 1 \right\rfloor.
\]

The interval and count (ATMOST, COLOURS, TASKS, SIZE_INTERVAL) constraint can be expressed in term of a set of reified constraints and of \( K \) arithmetic constraints (i.e., sum_ctr constraints).

1. For each task \( \text{TASKS}[i] \) \( (i \in [1, |\text{TASKS}|]) \) of the \( \text{TASKS} \) collection we create a 0-1 variable \( B_i \) that will be set to 1 if and only if task \( \text{TASKS}[i] \) takes a colour within the
set of colours COLOURS:
$$B_i \leftrightarrow \text{TASKS}[i].\text{colour} = \text{COLOURS}[i].\text{val} \lor$$
$$\text{TASKS}[i].\text{colour} = \text{COLOURS}[2].\text{val} \lor$$
$$\ldots$$
$$\text{TASKS}[i].\text{colour} = \text{COLOURS}[\|\text{COLOURS}\|].\text{val}.$$  

2. For each task $\text{TASKS}[i] \ (i \in [1, \|\text{TASKS}\|])$ and for each interval $[k \cdot \text{SIZE\_INTERVAL}, k \cdot \text{SIZE\_INTERVAL} + \text{SIZE\_INTERVAL} - 1] \ (k \in [0, K])$ we create a 0-1 variable $B_{ik}$ that will be set to 1 if and only if, both task $\text{TASKS}[i]$ takes a colour within the set of colours COLOURS, and the origin of task $\text{TASKS}[i]$ is assigned within interval $[k \cdot \text{SIZE\_INTERVAL}, k \cdot \text{SIZE\_INTERVAL} + \text{SIZE\_INTERVAL} - 1]$:

$$B_{ik} \leftrightarrow B_i \land$$
$$\text{TASKS}[i].\text{origin} \geq k \cdot \text{SIZE\_INTERVAL} \land$$
$$\text{TASKS}[i].\text{origin} \leq k \cdot \text{SIZE\_INTERVAL} + \text{SIZE\_INTERVAL} - 1$$

3. Finally, for each interval $[k \cdot \text{SIZE\_INTERVAL}, k \cdot \text{SIZE\_INTERVAL} + \text{SIZE\_INTERVAL} - 1] \ (k \in [0, K])$, we impose the sum $B_{1k} + B_{2k} + \ldots + B_{|\text{TASKS}|k}$ to not exceed the maximum allowed capacity $\text{ATMOST}$.

See also

- assignment dimension removed: among_low_up (assignment dimension corresponding to intervals is removed).
- related: interval_and_sum (among_low_up constraint replaced by sum_ctr).
- used in graph description: among_low_up.

Keywords

- application area: assignment.
- characteristic of a constraint: coloured, automaton, automaton with array of counters.
- constraint type: timetabling constraint, resource constraint, temporal constraint.
- modelling: assignment dimension, interval.
Arc input(s)  
TASKS

Arc generator  
$\text{PRODUCT} \rightarrow \text{collection}(\text{tasks1, tasks2})$

Arc arity  
2

Arc constraint(s)  
$\text{tasks1.origin/\text{SIZE\_INTERVAL}} = \text{tasks2.origin/\text{SIZE\_INTERVAL}}$

Sets  
$\text{SUCC} \mapsto$

\[
\text{variables} - \text{col}\left(\text{VARIABLES}\rightarrow\text{collection}(\text{var-dvar}), \text{item}[\text{var} \rightarrow \text{TASKS.colour}]\right)
\]

Constraint(s) on sets  
$\text{among\_low\_up}(0, \text{ATMOST}, \text{variables}, \text{COLOURS})$

Graph model  

We use a bipartite graph where each class of vertices corresponds to the different tasks of the TASKS collection. There is an arc between two tasks if their origins belong to the same interval. Finally we enforce an among_low_up constraint on each set $\mathcal{S}$ of successors of the different vertices of the final graph. This puts a restriction on the maximum number of tasks of $\mathcal{S}$ for which the colour attribute takes its value in COLOURS.

Parts (A) and (B) of Figure 5.347 respectively show the initial and final graph associated with the Example slot. Each connected component of the final graph corresponds to items that are all assigned to the same interval.

![Initial and final graph](image)

Figure 5.347: Initial and final graph of the interval_and_count constraint
Automaton

Figure 5.348 depicts the automaton associated with the *interval_and_count* constraint. Let $\text{COLOUR}_i$ be the colour attribute of the $i$th item of the TASKS collection. To each pair $(\text{COLOURS}, \text{COLOUR}_i)$ corresponds a signature variable $S_i$ as well as the following signature constraint: $\text{COLOUR}_i \in \text{COLOURS} \iff S_i$.

![Automaton Diagram]

Figure 5.348: Automaton of the *interval_and_count* constraint
5.181  interval_and_sum

**Description**

Derived from cumulative.

**Constraint**

interval_and_sum(SIZE_INTERVAL, TASKS, LIMIT)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIZE_INTERVAL</td>
<td>int</td>
</tr>
<tr>
<td>TASKS</td>
<td>collection(origin-dvar, height-dvar)</td>
</tr>
<tr>
<td>LIMIT</td>
<td>int</td>
</tr>
</tbody>
</table>

**Restrictions**

- SIZE_INTERVAL > 0
- TASKS(origin ≥ 0)
- TASKS.height ≥ 0
- LIMIT ≥ 0

A maximum resource capacity constraint: We have to fix the origins of a collection of tasks in such a way that, for all the tasks that are allocated to the same interval, the sum of the heights does not exceed a given capacity. All the intervals we consider have the following form: $[k \cdot \text{SIZE\_INTERVAL}, k \cdot \text{SIZE\_INTERVAL} + \text{SIZE\_INTERVAL} - 1]$, where $k$ is an integer.

**Purpose**

Figure 5.349 shows the solution associated with the example. The constraint interval_and_sum holds since the sum of the heights of the tasks that are located in the same interval does not exceed the limit 5. Each task $t$ is depicted by a rectangle $r$ associated with the interval to which the task $t$ is assigned. The rectangle $r$ is labelled with the position of $t$ within the items of the TASKS collection. The origin of task $t$ is represented by a small black square located within its corresponding rectangle $r$. Finally, the height of a rectangle $r$ is equal to the height of the task $t$ to which it corresponds.

![Figure 5.349: Solution showing for each interval the corresponding tasks](image)

Figure 5.349: Solution showing for each interval the corresponding tasks
Typical

\[ \text{SIZE\_INTERVAL} > 1 \]
\[ |\text{TASKS}| > 1 \]
\[ \text{range}(\text{TASKS}.\text{origin}) > 1 \]
\[ \text{range}(\text{TASKS}.\text{height}) > 1 \]
\[ \text{LIMIT} < \text{sum}(\text{TASKS}.\text{height}) \]

Symmetries

- Items of TASKS are **permutable**.
- One and the same constant can be **added** to the origin attribute of all items of TASKS.
- An occurrence of a value of TASKS.origin that belongs to the \( k \)-th interval, of size \( \text{SIZE\_INTERVAL} \), can be **replaced** by any other value of the same interval.
- TASKS.height can be **decreased** to any value \( \geq 0 \).
- LIMIT can be **increased**.

Arg. properties

**Contractible** wrt. TASKS.

Usage

This constraint can be used for timetabling problems. In this context the different intervals are interpreted as morning and afternoon periods of different consecutive days. We have a capacity constraint for all tasks that are assigned to the same morning or afternoon of a given day.

Reformulation

Let \( K \) denote the index of the last possible interval where the tasks can be assigned:
\[
K = \left\lfloor \frac{\text{max}_{i \in \{1,|\text{TASKS}|\}} (\text{TASKS}[i].\text{origin} \times \text{SIZE\_INTERVAL}) + \text{SIZE\_INTERVAL} - 1}{\text{SIZE\_INTERVAL}} \right\rfloor.
\]

The interval and sum (\( \text{SIZE\_INTERVAL}, \text{TASKS}, \text{LIMIT} \)) constraint can be expressed in term of a set of reified constraints and of \( K \) arithmetic constraints (i.e., **scalar product** constraints).

1. For each task \( \text{TASKS}[i] \) \( (i \in [1,|\text{TASKS}|]) \) and for each interval \( [k \times \text{SIZE\_INTERVAL}, k \times \text{SIZE\_INTERVAL} + \text{SIZE\_INTERVAL} - 1] \) \( (k \in [0, K]) \), we create a 0-1 variable \( B_{ik} \) that will be set to 1 if and only if the origin of task \( \text{TASKS}[i] \) is assigned within interval \( [k \times \text{SIZE\_INTERVAL}, k \times \text{SIZE\_INTERVAL} + \text{SIZE\_INTERVAL} - 1] \):
\[
B_{ik} \leftrightarrow \text{TASKS}[i].\text{origin} \geq k \times \text{SIZE\_INTERVAL} \land \text{TASKS}[i].\text{origin} \leq k \times \text{SIZE\_INTERVAL} + \text{SIZE\_INTERVAL} - 1
\]
2. Finally, for each interval \( [k \times \text{SIZE\_INTERVAL}, k \times \text{SIZE\_INTERVAL} + \text{SIZE\_INTERVAL} - 1] \) \( (k \in [0, K]) \), we impose the sum \( \text{TASKS}[1].\text{height} \cdot B_{1k} + \text{TASKS}[2].\text{height} \cdot B_{2k} + \ldots + \text{TASKS}[|\text{TASKS}|].\text{height} \cdot B_{|\text{TASKS}|,k} \) to not exceed the maximum allowed capacity \( \text{LIMIT} \).

See also

**assignment dimension removed**: \( \text{sum\_ctr} \) (assignment dimension corresponding to intervals is removed).

**related**: interval and count (\( \text{sum\_ctr} \) constraint replaced by among low up).

**used in graph description**: \( \text{sum\_ctr} \).

Keywords

**application area**: assignment.

**characteristic of a constraint**: automaton, automaton with array of counters.

**constraint type**: timetabling constraint, resource constraint, temporal constraint.

**modelling**: assignment dimension, interval.
Arc input(s) \( \text{TASKS} \)

Arc generator \( \text{PRODUCT} \mapsto \text{collection}(\text{tasks1}, \text{tasks2}) \)

Arc arity 2

Arc constraint(s) \( \text{tasks1.\text{origin/SIZE\_INTERVAL}} = \text{tasks2.\text{origin/SIZE\_INTERVAL}} \)

Sets \( \text{SUCC} \mapsto \)
\[
\begin{aligned}
\text{source,} \\
\text{variables} - \text{col}(\text{VARIABLES\_collection}(\text{var\_dvar}),) \\
\text{[item} \text{var} - \text{TASKS.height})
\end{aligned}
\]

Constraint(s) on sets \( \text{sum} \_\text{ctr}(\text{variables}, \leq, \text{LIMIT}) \)

Graph model

We use a bipartite graph where each class of vertices corresponds to the different tasks of the TASKS collection. There is an arc between two tasks if their origins belong to the same interval. Finally we enforce a sum_ctr constraint on each set \( S \) of successors of the different vertices of the final graph. This put a restriction on the maximum value of the sum of the height attributes of the tasks of \( S \).

Parts (A) and (B) of Figure 5.350 respectively show the initial and final graph associated with the Example slot. Each connected component of the final graph corresponds to items that are all assigned to the same interval.

Figure 5.350: Initial and final graph of the interval and sum constraint
Automaton

Figure 5.351 depicts the automaton associated with the \texttt{interval} \texttt{and sum} constraint. To each item of the collection \texttt{TASKS} corresponds a signature variable $S_i$ that is equal to 1.

\[
\begin{align*}
\text{Figur}e~5.351:~\text{Automaton of the } \texttt{interval} \texttt{and sum} \text{ constraint}
\end{align*}
\]
5.182 inverse

**Origin**  
CHIP

**Constraint**  
`inverse(NODES)`

**Synonyms**  
assignment, channel, inverse_channeling.

**Argument**  
`NODES : collection(index=int, succ=dvar, pred=dvar)`

**Restrictions**  
`required(NODES,[index, succ, pred])`
- `NODES.index ≥ 1`
- `NODES.index ≤ |NODES|`
- `distinct(NODES,index)`
- `NODES.succ ≥ 1`
- `NODES.succ ≤ |NODES|`
- `NODES.pred ≥ 1`
- `NODES.pred ≤ |NODES|`

**Purpose**  
Enforce each vertex of a digraph to have exactly one predecessor and one successor. In addition the following two statements are equivalent:

1. The successor of the $i^{th}$ node is the $j^{th}$ node.
2. The predecessor of the $j^{th}$ node is the $i^{th}$ node.

**Example**  
\[
\begin{pmatrix}
  index - 1 & succ - 2 & pred - 2, \\
  index - 2 & succ - 1 & pred - 1, \\
  index - 3 & succ - 5 & pred - 4, \\
  index - 4 & succ - 3 & pred - 5, \\
  index - 5 & succ - 4 & pred - 3
\end{pmatrix}
\]

The inverse constraint holds since:

- `NODES[1].succ = 2 ↔ NODES[2].pred = 1`,
- `NODES[2].succ = 1 ↔ NODES[1].pred = 2`,
- `NODES[3].succ = 5 ↔ NODES[5].pred = 3`,
- `NODES[4].succ = 3 ↔ NODES[3].pred = 4`,

**Typical**  
$|NODES| > 1$

**Symmetries**  
- Items of `NODES` are permutable.
- Attributes of `NODES` are permutable w.r.t. permutation `(index) (succ, pred)` (permutation applied to all items).
Arg. properties

- Functional dependency: NODES.succ determined by NODES.index and NODES.pred.
- Functional dependency: NODES.pred determined by NODES.index and NODES.succ.

Usage

This constraint is used in order to make the link between the successor and the predecessor variables. This is sometimes required by specific heuristics that use both predecessor and successor variables. In some problems, the successor and predecessor variables are respectively interpreted as column and row variables (i.e., we have a bijection between the successor variables and their values). This is for instance the case in the \( n \)-queens problem (i.e., place \( n \) queens on a \( n \) by \( n \) chessboard in such a way that no two queens are on the same row, the same column or the same diagonal) when we use the following model: to each column of the chessboard we associate a variable that gives the row where the corresponding queen is located. Symmetrically, to each row of the chessboard we create a variable that indicates the column where the associated queen is placed. Having these two sets of variables, we can now write a heuristics that selects the column or the row for which we have the fewest number of alternatives for placing a queen.

Remark

In the original inverse constraint of CHIP the index attribute was not explicitly present. It was implicitly defined as the position of a variable in a list, the first position being 1. This is also the case for SICStus Prolog, JaCoP and Gecode where the variables are respectively indexed from 1, 0 and 0. Within SICStus Prolog and JaCoP (http://www.jacop.eu/), the inverse constraint is called assignment. Within Gecode, it is called channel (http://www.gecode.org/).

Algorithm

We can reuse the filtering algorithm associated with the alldifferent constraint, both for the successor and the predecessor variables. In addition, each time value \( j \) is removed from the \( i \)th successor variable, we have to remove value \( i \) from the \( j \)th predecessor variable. Similarly, each time value \( i \) is removed from the \( j \)th successor variable, we have also to remove value \( j \) from the \( i \)th predecessor variable.

Systems

inverseChanneling in Choco, channel in Gecode, inverse in MiniZinc, assignment in SICStus.

See also

- common keyword: cycle, symmetric alldifferent (permutation).
- generalisation: inverse_offset (do not assume anymore that the smallest value of the pred or succ attributes is equal to 1), inverse_set (domain variable replaced by set variable), inverse_within_range (partial mapping between two collections of distinct size).
- implies (items to collection): lex alldifferent.

Keywords

- characteristic of a constraint: automaton, automaton with array of counters.
- combinatorial object: permutation.
- constraint arguments: pure functional dependency.
- constraint type: graph constraint.
- filtering: arc-consistency.
- heuristics: heuristics.
modelling: channelling constraint, permutation channel, dual model, functional dependency.

modelling exercises: n-Amazon, zebra puzzle.

puzzles: n-Amazon, n-queen, zebra puzzle.
Arc input(s)  NODES
Arc generator  $CLIQUE\mapsto collection(nodes1, nodes2)$
Arc arity  2
Arc constraint(s)  
  • nodes1.succ = nodes2.index
  • nodes2.pred = nodes1.index

Graph property(ies)  \[\text{NARC} = |\text{NODES}|\]

Graph model  
In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices. This is why the inverse constraint considers objects that have three attributes:

  • One fixed attribute $\text{index}$ that is the identifier of the vertex,
  • One variable attribute $\text{succ}$ that is the successor of the vertex,
  • One variable attribute $\text{pred}$ that is the predecessor of the vertex.

Parts (A) and (B) of Figure 5.352 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

![Graph model](image)

Figure 5.352: Initial and final graph of the inverse constraint

Signature  
Since all the $\text{index}$ attributes of the NODES collection are distinct and because of the first condition $\text{nodes1.succ} = \text{nodes2.index}$ of the arc constraint all the vertices of the final graph have at most one predecessor.

Since all the $\text{index}$ attributes of the NODES collection are distinct and because of the second condition $\text{nodes2.pred} = \text{nodes1.index}$ of the arc constraint all the vertices of the final graph have at most one successor.

From the two previous remarks it follows that the final graph is made up from disjoint paths and disjoint circuits. Therefore the maximum number of arcs of the final graph is
equal to its maximum number of vertices $\text{NODES}$. So we can rewrite the graph property $\text{NARC} = |\text{NODES}|$ to $\text{NARC} \geq |\text{NODES}|$ and simplify $\text{NARC}$ to $\text{NARC}$. 
Figure 5.353 depicts the automaton associated with the inverse constraint. To each item of the collection NODES corresponds a signature variable $S_i$ that is equal to 1.

\[
S_i \quad \text{arith}(C_i, =, 0)
\]

\[
\begin{aligned}
&\{C[Succ_i] = C[Succ_i] + \text{INDEX}_i, \\
&C[\text{INDEX}_i] = C[\text{INDEX}_i] = \text{PRED}_i
\end{aligned}
\]

Figure 5.353: Automaton of the inverse constraint
5.183  \textit{inverse\_offset}

\begin{tabular}{|l|l|l|}
\hline
\textbf{Origin} & Gecode & \\
\hline
\textbf{Constraint} & \textit{inverse\_offset}(\textsc{soffset}, \textsc{poffset}, \textsc{nodes}) & \\
\hline
\textbf{Synonym} & channel. & \\
\hline
\textbf{Arguments} & \begin{tabular}{l}
\textsc{soffset} : int \\
\textsc{poffset} : int \\
\textsc{nodes} : collection(index\_int, succ\_dvar, pred\_dvar)
\end{tabular} & \\
\hline
\textbf{Restrictions} & \begin{tabular}{l}
\textit{required}(\textsc{nodes}, \{index, succ, pred\}) \\
\textsc{nodes}.\text{index} \geq 1 \\
\textsc{nodes}.\text{index} \leq |\textsc{nodes}| \\
\textit{distinct}(\textsc{nodes}, \text{index}) \\
\textsc{nodes}.\text{succ} \geq 1 + \textsc{soffset} \\
\textsc{nodes}.\text{succ} \leq |\textsc{nodes}| + \textsc{soffset} \\
\textsc{nodes}.\text{pred} \geq 1 + \textsc{poffset} \\
\textsc{nodes}.\text{pred} \leq |\textsc{nodes}| + \textsc{poffset}
\end{tabular} & \\
\hline
\textbf{Purpose} & \begin{tabular}{l}
Enforce each vertex of a digraph to have exactly one predecessor and one successor. In addition the following two statements are equivalent: \\
1. The successor of the $i^{th}$ node minus \textsc{soffset} is equal to $j$. \\
2. The predecessor of the $j^{th}$ node minus \textsc{poffset} is equal to $i$. \\
\end{tabular} & \\
\hline
\textbf{Example} & \begin{tabular}{l}
\begin{pmatrix}
-1, 0, \\
index - 1 & succ - 4 & pred - 3, \\
index - 2 & succ - 2 & pred - 5, \\
index - 3 & succ - 0 & pred - 2, \\
index - 4 & succ - 6 & pred - 8, \\
index - 5 & succ - 1 & pred - 1, \\
index - 6 & succ - 7 & pred - 7, \\
index - 7 & succ - 5 & pred - 4, \\
index - 8 & succ - 3 & pred - 6
\end{pmatrix}
\end{tabular} & \\
\hline
\end{tabular}

The \textit{inverse\_offset} constraint holds since:
\begin{itemize}
\item \textsc{nodes}[1].\text{succ} - (-1) = 5 \leftrightarrow \textsc{nodes}[5].\text{pred} - 0 = 1,
\item \textsc{nodes}[2].\text{succ} - (-1) = 3 \leftrightarrow \textsc{nodes}[3].\text{pred} - 0 = 2,
\item \textsc{nodes}[3].\text{succ} - (-1) = 1 \leftrightarrow \textsc{nodes}[1].\text{pred} - 0 = 3,
\item \textsc{nodes}[4].\text{succ} - (-1) = 7 \leftrightarrow \textsc{nodes}[7].\text{pred} - 0 = 4,
\item \textsc{nodes}[5].\text{succ} - (-1) = 2 \leftrightarrow \textsc{nodes}[2].\text{pred} - 0 = 5.
\end{itemize}
\[ \text{NODES}[6].\text{succ} - (-1) = 8 \Leftrightarrow \text{NODES}[8].\text{pred} - 0 = 6. \]
\[ \text{NODES}[7].\text{succ} - (-1) = 6 \Leftrightarrow \text{NODES}[6].\text{pred} - 0 = 7. \]
\[ \text{NODES}[8].\text{succ} - (-1) = 4 \Leftrightarrow \text{NODES}[4].\text{pred} - 0 = 8. \]

Figure 5.354 shows the board that can be associated with this example.

Figure 5.354: Board associated with the example of the **Example** slot

**Typical**

\[
\begin{align*}
\text{SOFFSET} &\geq -1 \\
\text{SOFFSET} &\leq 1 \\
\text{POFFSET} &\geq -1 \\
\text{POFFSET} &\leq 1 \\
|\text{NODES}| &> 1
\end{align*}
\]

**Symmetry**

Items of \text{NODES} are permutable.

**Arg. properties**

- Functional dependency: \text{NODES}.\text{succ} determined by \text{SOFFSET}, \text{POFFSET}, \text{NODES}.\text{index} and \text{NODES}.\text{pred}.
- Functional dependency: \text{NODES}.\text{pred} determined by \text{SOFFSET}, \text{POFFSET}, \text{NODES}.\text{index} and \text{NODES}.\text{succ}.

**Remark**

The inverse_offset constraint is called channel in \textbf{Gecode} (http://www.gecode.org/). Having two offsets was motivated by the fact that it is possible to declare arrays at any position in the MiniZinc modelling language.

**Systems**

inverseChanneling in \textbf{Choco}, channel in \textbf{Gecode}.

**See also**

specialisation: inverse (assume that \text{SOFFSET} and \text{POFFSET} are both equal to 0).

**Keywords**

constraint arguments: pure functional dependency.
constraint type: graph constraint.
filtering: arc-consistency.
heuristics: heuristics.
Arc input(s) | NODES
---|---
Arc generator | \( CLIQUE \rightarrow \text{collection}(\text{nodes1}, \text{nodes2}) \)
Arc arity | 2
Arc constraint(s) | • \text{nodes1}.\text{succ} - \text{SOFFSET} = \text{nodes2}.\text{index}
• \text{nodes2}.\text{pred} - \text{POFFSET} = \text{nodes1}.\text{index}

Graph property(ies) | \( \text{NARC} = |\text{NODES}| \)

Graph model

In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices. This is why the \text{inverse_offset} constraint considers objects that have three attributes:

- One fixed attribute \text{index} that is the identifier of the vertex,
- One variable attribute \text{succ} that is the successor of the vertex,
- One variable attribute \text{pred} that is the predecessor of the vertex.

Parts (A) and (B) of Figure 5.355 respectively show the initial and final graph associated with the \textbf{Example} slot. Since we use the \textbf{NARC} graph property, the arcs of the final graph are stressed in bold.

Figure 5.355: Initial and final graph of the \text{inverse_offset} constraint
5.184 inverse_set

**DESCRIPTION**

- **Origin**: Derived from inverse.
- **Constraint**: inverse_set(X, Y)
- **Arguments**: 
  - X : collection(index=int, set=svar)
  - Y : collection(index=int, set=svar)
- **Restrictions**: 
  - required(X, [index, set])
  - required(Y, [index, set])
  - increasing_seq(X, index)
  - increasing_seq(Y, index)
  - X.index ≥ 1
  - X.index ≤ |X|
  - Y.index ≥ 1
  - Y.index ≤ |Y|
  - X.set ≥ 1
  - X.set ≤ |Y|
  - Y.set ≥ 1
  - Y.set ≤ |X|

**Purpose**

The following two statements are equivalent:

1. Value $j$ belongs to the set variable of the $i^{th}$ item of the $X$ collection.
2. Value $i$ belongs to the set variable of the $j^{th}$ item of the $Y$ collection.

I.e., \( j \in X[i] \iff i \in Y[j] \).

**Example**

\[
\begin{pmatrix}
\text{index} - 1 & \text{set} - \{2, 4\}, \\
\text{index} - 2 & \text{set} - \{4\}, \\
\text{index} - 3 & \text{set} - \{1\}, \\
\text{index} - 4 & \text{set} - \{4\}, \\
\text{index} - 1 & \text{set} - \{3\}, \\
\text{index} - 2 & \text{set} - \{1\}, \\
\text{index} - 3 & \text{set} - \emptyset, \\
\text{index} - 4 & \text{set} - \{1, 2, 4\}, \\
\text{index} - 5 & \text{set} - \emptyset
\end{pmatrix}
\]

The inverse_set constraint holds since:

\[
\begin{align*}
2 \in X[1].set & \iff 1 \in Y[2].set, \\
4 \in X[1].set & \iff 1 \in Y[4].set, \\
4 \in X[2].set & \iff 2 \in Y[4].set, \\
1 \in X[3].set & \iff 3 \in Y[1].set, \\
4 \in X[4].set & \iff 4 \in Y[4].set.
\end{align*}
\]

**Typical**

- \(|X| > 1\)
- \(|Y| > 1\)
Symmetries

- Arguments are permutable w.r.t. permutation $(X, Y)$.
- Items of $X$ are permutable.
- Items of $Y$ are permutable.

Usage

The `inverse_set` constraint can for instance be used in order to model problems where one has to place items on a rectangular board in such a way that a column or a row can have more than one item. We have one set variable for each row of the board; Its values are the column indexes corresponding to the positions where an item is placed. Similarly we have also one set variable for each column of the board; Its values are the row indexes corresponding to the positions where an item is placed. The `inverse_set` constraint maintains the link between the rows and the columns variables. Figure 5.356 shows the board that can be associated with the example of the Example slot.

Figure 5.356: Board associated with the example of the Example slot

Systems

- `inverseSet` in Choco, `inverse_set` in MiniZinc.

See also

- common keyword: `inverse_within_range` (channelling constraint).
- specialisation: `inverse` (set variable replaced by domain variable).
- used in graph description: `in_set`.

Keywords

- constraint arguments: constraint involving set variables.
- modelling: channelling constraint, set channel, dual model.
Arc input(s) | X Y
---|---
Arc generator | \( \text{PRODUCT} \rightarrow \text{collection}(x, y) \)
Arc arity | 2
Arc constraint(s) | in_set(y.index, x.set) ↔ in_set(x.index, y.set)
Graph property(ies) | \( \text{NARC} = |X| \times |Y| \)

Graph model

Parts (A) and (B) of Figure 5.357 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

Figure 5.357: Initial and final graph of the inverse set constraint
5.185  inverse_within_range

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from inverse.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>inverse_within_range(X, Y)</td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>inverse_in_range, inverse_range.</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>X : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Y : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(X, var)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>required(Y, var)</td>
<td></td>
</tr>
</tbody>
</table>

If the $i^{th}$ variable of the collection $X$ is assigned to $j$ and if $j$ is greater than or equal to 1 and less than or equal to the number of items of the collection $Y$ then the $j^{th}$ variable of the collection $Y$ is assigned to $i$.

Conversely, if the $j^{th}$ variable of the collection $Y$ is assigned to $i$ and if $i$ is greater than or equal to 1 and less than or equal to the number of items of the collection $X$ then the $i^{th}$ variable of the collection $X$ is assigned to $j$.

Example

$$\left(\langle 9, 4, 2 \rangle, \langle 9, 3, 9, 2 \rangle\right)$$

Since the second item of $X$ is assigned to 4, the fourth item of $Y$ is assigned to 2. Similarly, since the third item of $X$ is assigned to 2, the second item of $Y$ is assigned to 3. Figure 5.358 illustrates the correspondence between $X$ and $Y$.

Figure 5.358: Correspondence between the items of $X = \langle 9, 4, 2 \rangle$ and the items of $Y = \langle 9, 3, 9, 2 \rangle$

Typical

$$\vert X \vert > 1$$

$$\text{range}(X, \text{var}) > 1$$

$$\vert Y \vert > 1$$

$$\text{range}(Y, \text{var}) > 1$$
**Symmetry**

Arguments are permutable w.r.t. permutation \((X, Y)\).

**Usage**

Consider an integer value \(m\) and a sequence of \(n\) variables \(S\) from which you have to select a subsequence \(S'\) such that:

- All variables of \(S'\) have to be assigned to distinct values from \([1, m]\),
- All variables not in \(S'\) have to be assigned a value, not necessarily distinct, outside \([1, m]\).

As for the **inverse** constraint we may want to create explicitly a value variable for each value in \([1, m]\) in order to state some specific constraints on the value variables or to use a heuristics involving the original variables of \(S\) as well as the value variables. The purpose of the **inverse**.within.range constraint is to link the variables of \(S\) with the value variables.

**See also**

- **common keyword**: inverse.set (**channelling constraint**).
- **specialisation**: inverse (**the 2 collections have not necessarily the same number of items**).

**Keywords**

- **constraint type**: graph constraint.
- **final graph structure**: bipartite, no loop, symmetric.
- **heuristics**: heuristics.
- **modelling**: channelling constraint, dual model.
<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>X Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td><code>SYMmetricPRODUCT</code> $\rightarrow$ \textit{collection}(s1, s2)</td>
</tr>
<tr>
<td>Arc arity</td>
<td>2</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>s1.var = s2.key</td>
</tr>
<tr>
<td>Graph class</td>
<td>- \textit{BIPARTITE}</td>
</tr>
<tr>
<td></td>
<td>- \textit{NO LOOP}</td>
</tr>
<tr>
<td></td>
<td>- \textit{SYMmetric}</td>
</tr>
</tbody>
</table>
### ith_pos_different_from_0

**Description**

The `ith_pos_different_from_0` constraint holds since 4 corresponds to the position of the 2\(^{nd}\) non-zero item of the sequence 3 0 0 8 6.

**Purpose**

POS is the position of the ITH\(^{th}\) non-zero item of the sequence of variables VARIABLES.

**Example**

```plaintext
(2, 4, ⟨3, 0, 0, 8, 6⟩)
```

**Typical**

- |VARIABLES| > 1
- range(VARIABLES.var) > 1
- atleast(1, VARIABLES.var)

**Symmetry**

An occurrence of a value of VARIABLES.var that is different from 0 can be replaced by any other value that is also different from 0.

**Arg. properties**

Suffix-extensible wrt. VARIABLES.

**Keywords**

- characteristic of a constraint: joker value, automaton, automaton with counters.
- constraint network structure: alpha-acyclic constraint network(3).
- constraint type: data constraint.
- modelling: table.
Automaton

Figure 5.359 depicts the automaton associated with the \texttt{ith pos different from 0} constraint. To each variable \texttt{VAR}_i of the collection \texttt{VARIABLES} corresponds a 0-1 signature variable \texttt{S}_i. The following signature constraint links \texttt{VAR}_i and \texttt{S}_i: \texttt{VAR}_i = 0 \leftrightarrow \texttt{S}_i.

\[
\begin{align*}
(C &= 0, D = 0) \\
\text{VAR}_1 &= 0, \\
\text{VAR}_1 &< 0, \\
\text{VAR}_1 &> 0,
\end{align*}
\]

\[
\begin{align*}
\text{if } C &< \text{ITH} \text{ then } D = D + 1 \\
\text{if } C &< \text{ITH} \text{ then } C = C + 1, D = D + 1
\end{align*}
\]

Figure 5.359: Automaton of the \texttt{ith pos different from 0} constraint

![Automaton Diagram]

Figure 5.360: Hypergraph of the reformulation corresponding to the automaton of the \texttt{ith pos different from 0} constraint
## 5.187 k_alldifferent

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>[138]</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>$k_{alldifferent}(\text{VARS})$</td>
<td></td>
</tr>
<tr>
<td><strong>Synonyms</strong></td>
<td>$k_{alldiff}, k_{alldistinct}, \text{some_different}.$</td>
<td></td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>$X : \text{collection}(x_dvar)$</td>
<td></td>
</tr>
<tr>
<td><strong>Argument</strong></td>
<td>$\text{VARS} : \text{collection}(\text{vars} - X)$</td>
<td></td>
</tr>
</tbody>
</table>
| **Restrictions** | $|X| \geq 1$
$\text{required}(X, x)$
$\text{required}(\text{VARS}, \text{vars})$
$|\text{VARS}| \geq 1$ |       |
| **Purpose** | For each collection of variables depicted by an item of $\text{VARS}$, enforce their corresponding variables to take distinct values. Usually some variables occur in several collections. |       |
| **Example** | $\begin{pmatrix}
  x - 5,
  x - 6,
  \langle \text{vars} - \langle x - 0, 
  x - 9, 
  x - 3 \rangle, 
  \text{vars} - \langle 5, 6, 1, 2 \rangle \rangle
  \end{pmatrix}$ |       |
|             | The $k_{alldifferent}$ constraint holds since all the values 5, 6, 0, 9 and 3 are distinct and since all the values 5, 6, 1 and 2 are distinct as well. |       |
| **Typical** | $|X| > 1$
$|\text{VARS}| > 1$ |       |
| **Symmetries** | • Items of $\text{VARS}$ are permutable.
• Items of $\text{VARS}$.vars are permutable.
• All occurrences of two distinct values of $\text{VARS}$.vars.x can be swapped; all occurrences of a value of $\text{VARS}$.vars.x can be renamed to any unused value. |       |
| **Arg. properties** | Contractible wrt. $\text{VARS}$. |       |
| **Usage**   | Systems of $\text{alldifferent}$ constraints sharing variables occurs frequently in practice. We give 4 typical problems that can be modelled by a combination of $\text{alldifferent}$ constraints as well as one problem where a system of $\text{alldifferent}$ constraints provides a necessary condition. |       |
• The graph colouring problem is to colour with a restricted number of colours the vertices of a given undirected graph in such a way that adjacent vertices are coloured with distinct colours. The problem can be modelled by a system of \textit{alldifferent} constraints. All the next problems can been seen as graph colouring problems where the graphs have some specific structure.

• A Latin square of order \(n\) is an \(n \times n\) array in which \(n\) distinct numbers in \([1, n]\) are arranged so that each number occurs once in each row and column. The problem is to complete a partially filled Latin square. Part (A) of Figure 5.361 gives a partially filled Latin square, while part (B) provides a possible completion.

-  \begin{array}{ccc}
1 & & 3 \\
&  &  \\
3 & & 1 \\
\end{array}

-  \begin{array}{ccc}
1 & 2 & 3 & 4 \\
2 & 1 & 4 & 3 \\
3 & 4 & 1 & 2 \\
4 & 3 & 2 & 1 \\
\end{array}

Figure 5.361: A partially filled Latin square and a possible completion

• A Sudoku is a Latin square of order \(9 \times 9\) such that the numbers in each major \(3 \times 3\) block are distinct. As for the Latin square problem, the problem is to complete a partially filled board. Part (A) of Figure 5.362 gives a partially filled Sudoku board, while part (B) provides a possible completion. A constraint programming approach for solving Sudoku puzzles is depicted in [363]. It shows how to generate redundant constraints as well as shaving [257] in order to find a solution without guessing.

-  \begin{array}{ccccc}
2 & 6 & 8 & 1 & 2 \\
3 & 7 & 8 & 6 & 3 \\
4 & 5 & 7 & 9 & 4 \\
5 & 1 & 7 & 9 & 5 \\
3 & 9 & 5 & 1 & 6 \\
4 & 3 & 2 & 5 & 4 \\
1 & 3 & 2 & 5 & 1 \\
5 & 2 & 4 & 9 & 2 \\
3 & 8 & 4 & 6 & 3 \\
\end{array}

-  \begin{array}{ccccc}
7 & 2 & 6 & 4 & 9 \\
3 & 1 & 5 & 7 & 2 \\
4 & 8 & 9 & 6 & 5 \\
8 & 5 & 2 & 1 & 4 \\
6 & 7 & 3 & 9 & 8 \\
9 & 4 & 1 & 3 & 6 \\
1 & 9 & 4 & 8 & 3 \\
5 & 6 & 7 & 2 & 1 \\
2 & 3 & 8 & 5 & 7 \\
\end{array}

Figure 5.362: A partially Sudoku square and a possible completion

• A task assignment problem consists to assign a given set of non-preemptive tasks, which are fixed in time (i.e., the origin, duration and end of each task are fixed), to a set of resources so that, tasks that are assigned to the same resource do not overlap in time. Each task can be assigned to a predefined set of resources. Problems like aircraft stand allocation [129], [362] or air traffic flow management [19] correspond to an example of a real-life task assignment problem. Assignment of service professionals [12] is yet another industrial example where professionals have to be
assigned positions in such a way that positions assigned to a given professional do not overlap in time.

Part (A) of Figure 5.363 gives an example of task assignment problem. For each task we indicate the set of resources where it can potentially be assigned (i.e., the domain of its assignment variable). For instance, task T1 can be assigned to resources 1 or 2. Part (B) of Figure 5.363 gives the corresponding interval graph: We have one vertex for each task and an edge between two tasks that overlap in time. We have a system of alldifferent constraints corresponding to the maximum cliques of the interval graph (i.e., \{T1,T5,T8\}, \{T2,T6\}, \{T3,T6,T9\}, \{T3,T7,T9\}, \{T4,T7,T9\}). Finally, part (C) of Figure 5.363 provides a possible solution to the task assignment problem where tasks T1, T2, T9 are assigned to resource 1, tasks T3, T4, T8 are assigned to resource 2, and tasks T5, T6, T7 are assigned to resource 3.

**Figure 5.363:** A task assignment problem, its corresponding interval graph and a possible solution

- The tree partitioning with precedences problem is to compute a vertex-partitioning of a given digraph \(G\) in disjoint trees (i.e., a forest), so that a given set of precedences holds. The problem can be modelled with a treeprecedence\(\text{\text{NTREE, VERTICES}}\) constraint, where \text{NTREE} is a domain variable specifying the numbers of trees in the forest and \text{VERTICES} is a collection of the digraph’s \(n\) vertices. Each item \(v \in \text{VERTICES}\) has the following attributes, which complete the description of the digraph:
  - \text{index} is an integer in \([1, n]\) that can be interpreted as the label of \(v\).
  - \text{father} is a domain variable whose domain consists of elements (vertex label) of \([1, n]\). It can be interpreted as the unique successor of \(v\).
  - \text{preds} is a possibly empty set of integers, its elements (vertex label) being in \([1, n]\). It can be interpreted as the mandatory ancestors of \(v\).

We model the treeprecedence constraint by the digraph \(G = (V, E)\) in which the vertices represent the elements of \text{VERTICES} and the arcs represent the successors relations between them. Formally, \(G\) is defined as follows:
  - To the \(i^{th}\) vertex \((1 \leq i \leq n)\), \text{VERTICES}[i], of the \text{VERTICES} collection corresponds a vertex of \(V\) denoted by \(v_i\).
  - For every pair of vertices (\text{VERTICES}[i],\text{VERTICES}[j]), where \(i\) and \(j\) are not necessarily distinct, there is an arc from \(v_i\) to \(v_j\) in \(E\).
The \texttt{tree\_precedence} constraint specifies that its associated digraph $\mathcal{G}$ should be a forest that fulfills the precedence constraints. Formally, a ground instance of a $\texttt{tree\_precedence}(\text{NTREE}, \text{VERTICES})$ constraint is satisfied if and only if the following conditions hold:

1. $\forall i \in [1, n] : \text{VERTICES}[i].\text{index} = i$,
2. Its associated digraph $\mathcal{G}$ consists of \text{NTREE} connected components,
3. Each connected component of $\mathcal{G}$ does not contain any circuit involving more than one vertex,
4. For every vertex $\text{VERTICES}[i]$ such that $j \in \text{VERTICES}[i].\text{preds}$ there must be an elementary path in $\mathcal{G}$ from $\text{VERTICES}[j]$ to $\text{VERTICES}[i]$.

We can build the following system of \texttt{alldifferent} constraints that corresponds to a necessary condition for the \texttt{tree\_precedence} constraint: To each vertex $v$ of $\mathcal{G}$, which both has no predecessors and cannot be the root of a tree, we generate an \texttt{alldifferent} constraint involving the father variables of those descendants of $v$ in $\mathcal{G}$ that cannot be the root of a tree.

![Figure 5.364: A set of precedences and a corresponding feasible tree](image)

For the set of precedences depicted by part (A) of Figure 5.364, where we assume that $\text{VERTICES}[12]$ is the only vertex that can be a root and where $F_i$ denotes the father variable associated with $\text{VERTICES}[i]$, we get the following system of \texttt{alldifferent} constraints:

- \texttt{alldifferent}($\langle F_1, F_3, F_5, F_6, F_7, F_{10}, F_{11} \rangle$),
- \texttt{alldifferent}($\langle F_2, F_4, F_7, F_8, F_9, F_{10}, F_{11} \rangle$).

The variables of these two \texttt{alldifferent} constraints respectively correspond to the descendants of the two source vertices (i.e., $F_1$ and $F_2$) of the precedence graph depicted by part (A) of Figure 5.364. On part (A) of Figure 5.364 the descendants of $F_1$ and $F_2$ are respectively depicted with a thick line and a grey circle. Their intersection, $\{F_7, F_{10}, F_{11}, F_{12}\}$, from which we remove

\footnote{The number in a vertex gives the value of the \texttt{index} attribute of the corresponding item.}
$F_{12}$ belong to the two \textit{alldifferent} constraints. In fact, $F_{12}$ is not mentioned in the two \textit{alldifferent} constraints since its corresponding vertex is the root of a tree. Part (B) of Figure 3.364 gives a possible tree satisfying all the precedences constraints expressed by part (A), where precedences are depicted with a dotted line. It corresponds to the following ground solution:

\begin{verbatim}
tree_precedence((
  index - 1 father - 3 preds - {}),
  index - 2 father - 4 preds - {}),
  index - 3 father - 5 preds - {1},
  index - 4 father - 8 preds - {2},
  index - 5 father - 6 preds - {1},
  index - 6 father - 7 preds - {3},
  index - 7 father - 10 preds - {3, 4},
  index - 8 father - 9 preds - {4},
  index - 9 father - 7 preds - {2},
  index - 10 father - 11 preds - {5, 6, 7},
  index - 11 father - 12 preds - {7, 8, 9},
  index - 12 father - 12 preds - {10, 11}))
\end{verbatim}

**Remark**

It was shown in [139] that, finding out whether a system of two \textit{alldifferent} constraints sharing some variables has a solution or not is \textit{NP}-hard. This was achieved by reduction from set packing.

A slight variation in the way of describing the arguments of the $k$ \textit{alldifferent} constraint appears in [337] under the name of \textit{some different}: the set of disequalities is described by a set of pairs of variables, where each pair corresponds to a disequality constraint between two given variables.

Within the context of linear programming, a relaxation of the $k$ \textit{alldifferent} constraint is provided in [7]. The special case where $k = 2$ is discussed in [8].

**Algorithm**

Even if there is no filtering algorithm for the $k$ \textit{alldifferent} constraint, one can enforce redundant constraints for the following patterns:

- Within the context of graph colouring, one can state an \textit{nvalue} constraint for every cycle of odd length of the graph to colour enforcing that the corresponding variables have to be assigned to at least three distinct values.

- Within the context of Latin squares, one can state a \textit{colored_matrix} constraint enforcing that each value is used exactly once in each row and column.

- Within the context of two \textit{alldifferent} constraints \textit{alldifferent}($U_1, \ldots, U_n, V_1, \ldots, V_m$) and \textit{alldifferent}($U_1, \ldots, U_n, W_1, \ldots, W_m$) where the domain of all variables $U_1, \ldots, U_n, V_1, \ldots, V_m, W_1, \ldots, W_m$ is included in the interval $[1, n + m]$, one can state a \textit{same and global cardinality} constraint stating that the variables $V_1, \ldots, V_m$ should correspond to a permutation of the variables $W_1, \ldots, W_m$ and that the variables $V_1, \ldots, V_m$ should be assigned to distinct values.

- In the general case of two \textit{alldifferent} constraints \textit{alldifferent}($U_1, \ldots, U_n, V_1, \ldots, V_m$) and \textit{alldifferent}($U_1, \ldots, U_n, W_1, \ldots, W_o$), one can state an \textit{nvalue} constraint involving the variables $V_1, \ldots, V_m$ and $W_1, \ldots, W_o$ enforcing that these variables should not use more than $s - n$ distinct values, where $s$ denotes the cardinality of the union of the domains of the variables $U_1, \ldots, U_n, V_1, \ldots, V_m, W_1, \ldots, W_o$. 


Several propagation rules for the $k_{\text{alldifferent}}$ constraint are also described in [235].

**Reformulation**
Given two $\text{alldifferent}$ constraints that share some variables, a reformulation preserving bound-consistency was introduced in [69]. This reformulation is based on an extension of Hall’s theorem that is presented in the same paper.

**See also**

*common keyword:* colored_matrix (*system of constraints*).
*generalisation:* diffn, geost (*tasks for which the start attribute is not fixed*).
*part of system of constraints:* alldifferent.
*related:* nvalue (*implied by two overlapping alldifferent*), same_and_global_cardinality (*implied by two overlapping alldifferent and restriction on values*).

**Keywords**

*application area:* air traffic management, assignment.
*characteristic of a constraint:* all different, disequality.
*combinatorial object:* permutation, Latin square.
*complexity:* set packing.
*constraint type:* system of constraints, overlapping alldifferent, value constraint, decomposition.
*filtering:* bound-consistency, duplicated variables.
*problems:* graph colouring.
*puzzles:* Sudoku.
For all items of $VARS$:

<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>$VARS.vars$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>$CLIQUE \mapsto \text{collection}(x_1, x_2)$</td>
</tr>
<tr>
<td>Arc arity</td>
<td>2</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>$x_1.x = x_2.x$</td>
</tr>
<tr>
<td>Graph property(ies)</td>
<td>$\text{MAX_NSCC} \leq 1$</td>
</tr>
</tbody>
</table>

**Graph model**

For each collection of variables depicted by an item of $VARS$ we generate a *clique* with an *equality* constraint between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed one.
### 5.188 k_cut

<table>
<thead>
<tr>
<th>Origin</th>
<th>E. Althaus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint</td>
<td>( \text{k_cut}(K, \text{NODES}) )</td>
</tr>
<tr>
<td>Arguments</td>
<td>( K : \text{int} )  \</td>
</tr>
<tr>
<td>Restrictions</td>
<td>( K \geq 1 )  \</td>
</tr>
<tr>
<td>Purpose</td>
<td>Select some arcs of a digraph in order to have at least ( K ) connected components (an isolated vertex, i.e. a vertex without any ingoing or outgoing arc, is counted as one connected component).</td>
</tr>
</tbody>
</table>
| Example | \[
\begin{pmatrix}
\text{index} - 1 & \text{succ} = \emptyset, \\
\text{index} - 2 & \text{succ} = \{3, 5\}, \\
3, & \text{succ} = \{5\}, \\
\text{index} - 4 & \text{succ} = \emptyset, \\
\text{index} - 5 & \text{succ} = \{2, 3\}
\end{pmatrix}
\]

The \( k \text{\_cut} \) constraint holds since the graph corresponding to the \( \text{NODES} \) collection contains 3 connected components (i.e., two connected components respectively involving vertices 1 and 4 and a third connected component containing the remaining vertices 2, 3 and 5), and since the first argument \( K \) enforces to have at least 3 connected components.

| Typical | \( | \text{NODES}| > 1 \) |
| Symmetries | • \( K \) can be decreased to any value \( \geq 1 \).  \\ | • Items of \( \text{NODES} \) are permutable. |
| See also | common keyword: \text{link\_set\_to\_booleans} (\text{constraint involving set variables}).  \\ | used in graph description: \text{in\_set}. |
| Keywords | constraint arguments: constraint involving set variables.  \\ | constraint type: graph constraint.  \\ | filtering: linear programming.  \\ | final graph structure: connected component. |
Arc input(s)  NODES
Arc generator  $CLIQUE\rightarrow collection(nodes1, nodes2)$
Arc arity  2
Arc constraint(s)  $nodes1.index = nodes2.index \lor in\ set(nodes2.index, nodes1.succ)$
Graph property(ies)  $\text{NCC} \geq K$

Graph model

$nodes1.index = nodes2.index$ holds if $nodes1$ and $nodes2$ correspond to the same vertex. It is used in order to enforce keeping all the vertices of the initial graph. This is because an isolated vertex counts always as one connected component. Within the context of the Example slot, part (A) of Figure 5.365 shows the initial graph from which we have chosen to start. It is derived from the set associated with each vertex. Each set describes the potential values of the $\text{succ}$ attribute of a given vertex. Part (B) of Figure 5.365 gives the final graph associated with the example of the Example slot. The $k$-cut constraint holds since we have at least $K = 3$ connected components in the final graph.

Figure 5.365: Initial and final graph of the $k$-cut set constraint
### 5.189 k_disjoint

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from disjoint</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>$k_{disjoint}(SETS)$</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>VARIABLES : collection(var–dvar)</td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>SETS : collection(set – VARIABLES)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>$required$(VARIABLES, var) [VARIABLES] ≥ 1 [SETS] &gt; 1</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>Given</td>
<td>[SETS] of domain variables, the $k_{disjoint}$ constraint enforces that no value is assigned to more than one set.</td>
</tr>
</tbody>
</table>

#### Example

The $k_{disjoint}$ constraint holds since:

- The set of values \{1, 5, 9\} and \{0, 2, 6, 7, 8\} respectively assigned to the variables of the first and second collections have an empty intersection.
- The set of values \{1, 5, 9\} and \{3, 4\} respectively assigned to the variables of the first and third collections have an empty intersection.
- The set of values \{0, 2, 6, 7, 8\} and \{3, 4\} respectively assigned to the variables of the second and third collections have an empty intersection.

#### Typical

\[|VARIABLES| > 1\]

#### Symmetries

- Items of SETS are **permutable**.
- Items of SETS.set are **permutable**.
- An occurrence of a value of VARIABLES.var can be **replaced** by any value of VARIABLES.var.
- All occurrences of two distinct values of SETS.set.var can be **swapped**; all occurrences of a value of SETS.set.var can be **renamed** to any unused value.
Arg. properties

Contractible wrt. SETS.

See also

part of system of constraints: disjoint.
used in graph description: disjoint.

Keywords

characteristic of a constraint: disequality.
constraint type: system of constraints, decomposition, value constraint.
modelling: empty intersection.
Arc input(s)  
Arc generator  \( CLIQUE(<) \rightarrow \text{collection}(\text{set1,set2}) \)
Arc arity  2
Arc constraint(s)  \( \text{disjoint}(\text{set1.set, set2.set}) \)
Graph property(ies)  \( \text{NARC} = |\text{SETS}| \times (|\text{SETS}| - 1)/2 \)

Graph model  
Parts (A) and (B) of Figure 5.366 respectively show the initial and final graph associated with the Example slot. To each vertex corresponds a collection of variables, while to each arc corresponds a disjoint constraint.

Figure 5.366: Initial and final graph of the k_disjoint constraint
## 5.190 \texttt{k_same}

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[138]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>\texttt{k_same(SETS)}</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>VARIABLES : \texttt{collection(var-dvar)}</td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>SETS : \texttt{collection(set - VARIABLES)}</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>\texttt{required(VARIABLES, var)} \texttt{</td>
<td>VARIABLES</td>
</tr>
</tbody>
</table>

**Purpose**

Given \(|SETS|\) sets, each containing the same number of domain variables, the \texttt{k_same} constraint enforces that the multisets of values assigned to each set are all identical.

### Example

\[
\begin{align*}
\text{set} &= \langle \text{var} - 1, \\
& \quad \text{var} - 9, \\
& \quad \text{var} - 1, \\
& \quad \text{var} - 5, \\
& \quad \text{var} - 2, \\
& \quad \text{var} - 1, \\
& \quad \text{var} - 9, \\
& \quad \text{var} - 1, \\
& \quad \text{var} - 5, \\
& \quad \text{var} - 2, \\
& \quad \text{var} - 1, \\
& \quad \text{var} - 9, \\
& \quad \text{var} - 1 \rangle
\end{align*}
\]

The \texttt{k_same} constraint holds since:

- The first and second collections of variables are assigned to the same multiset.
- The second and third collections of variables are also assigned to the same multiset.

**Typical**

\(|VARIABLES| > 1\)
Symmetries

- Items of SETS are permutable.
- Items of SETS.set are permutable.
- All occurrences of two distinct values of SETS.set.var can be swapped; all occurrences of a value of SETS.set.var can be renamed to any unused value.

Arg. properties

Contractible wrt. SETS.

Remark

It was shown in [138] that, finding out whether the \textit{k}\_same constraint has a solution or not is NP-hard when we have more than one \textit{same} constraint. This was achieved by reduction from 3-dimensional-matching in the context where we have 2 \textit{same} constraints.

See also

- common keyword: \textit{k}\_same\_interval, \textit{k}\_same\_modulo, \textit{k}\_same\_partition (system of constraints).
- implies: \textit{k}\_used\_by.
- part of system of constraints: \textit{same}.
- used in graph description: \textit{same}.

Keywords

- characteristic of a constraint: sort based reformulation.
- combinatorial object: permutation, multiset.
- complexity: 3-dimensional-matching.
- constraint type: system of constraints, decomposition.
- modelling: equality between multisets.
Arc input(s) | SETS
---|---
Arc generator | \( \text{PATH} \rightarrow \text{collection(set1, set2)} \)
Arc arity | 2
Arc constraint(s) | \( \text{same(set1.set, set2.set)} \)
Graph property(ies) | \( \text{NARC} = |\text{SETS}| - 1 \)

**Graph model**

Parts (A) and (B) of Figure 5.367 respectively show the initial and final graph associated with the **Example** slot. To each vertex corresponds a collection of variables, while to each arc corresponds a **same** constraint.

![Graph Model Diagram]

Figure 5.367: Initial and final graph of the \( k \text{.same} \) constraint
5.191  k_same_interval

<table>
<thead>
<tr>
<th>Description</th>
<th>Links</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from same_interval and from k_same.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>k_same_interval(SETS, SIZE_INTERVAL)</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>VARIABLES : collection(var–dvar)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>SETS : collection(set – VARIABLES)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SIZE_INTERVAL : int</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(VARIABLES, var)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[VARIABLES] ≥ 1</td>
</tr>
<tr>
<td></td>
<td>required(SETS, set)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[SETS] &gt; 1</td>
</tr>
<tr>
<td></td>
<td>same_size(SETS, set)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SIZE_INTERVAL &gt; 0</td>
</tr>
<tr>
<td>Purpose</td>
<td>Given a collection of [SETS] sets, each containing the same number of domain variables, the k_same_interval constraint enforces a same_interval constraint between each pair of consecutive sets.</td>
<td></td>
</tr>
</tbody>
</table>

Example

In the example, the second argument SIZE_INTERVAL = 3 of the k_same_interval constraint defines the following family of intervals $[3 \cdot k, 3 \cdot k + 2]$, where $k$ is an integer. The k_same_interval constraint holds since:

In the example, the second argument SIZE_INTERVAL = 3 of the k_same_interval constraint defines the following family of intervals $[3 \cdot k, 3 \cdot k + 2]$, where $k$ is an integer. The k_same_interval constraint holds since:
• The first and second collections of variables are assigned 4 values in the interval [0, 2] as well as 2 values in the interval [6, 8].
• The second and third collections of variables are also assigned 4 values in the interval [0, 2] as well as 2 values in the interval [6, 8].

Typical

\[
\begin{align*}
\text{|VARIABLES|} & > 1 \\
\text{SIZE_INTERVAL} & > 1
\end{align*}
\]

Symmetries

• Items of SETS are permutable.
• Items of SETS.set are permutable.
• An occurrence of a value of SETS.set.var that belongs to the \(k\)-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.

Arg. properties

Contractible wrt. SETS.

See also

- common keyword: \texttt{k\_same (system of constraints)}.
- implies: \texttt{k\_used\_by\_interval}.
- part of system of constraints: \texttt{same\_interval}.
- used in graph description: \texttt{same\_interval}.

Keywords

- characteristic of a constraint: sort based reformulation.
- combinatorial object: permutation.
- constraint type: system of constraints, decomposition.
- modelling: interval.
Arc input(s) | SETS
---|---
Arc generator | $PATH \rightarrow \text{collection}(\text{set1}, \text{set2})$
Arc arity | 2
Arc constraint(s) | $\text{same\_interval}(\text{set1.set}, \text{set2.set}, \text{SIZE\_INTERVAL})$
Graph property(ies) | $\text{NARC} = |\text{SETS}| - 1$

Graph model

Parts (A) and (B) of Figure 5.368 respectively show the initial and final graph associated with the Example slot. To each vertex corresponds a collection of variables, while to each arc corresponds a $\text{same\_interval}$ constraint.

Figure 5.368: Initial and final graph of the $k_{\text{same\_interval}}$ constraint
5.192  k_same_modulo

DESCRIPTION  LINKS  GRAPH

Origin  Derived from same_modulo and from k_same.

Constraint  k_same_modulo(SETS, M)

Type  VARIABLES : collection(var−dvar)

Arguments  SETS : collection(set − VARIABLES)
M : int

Restrictions  required(VARIABLES, var)
  |VARIABLES| ≥ 1
  required(SETS, set)
  |SETS| > 1
  same_size(SETS, set)
M > 0

Purpose  Given a collection of |SETS| sets, each containing the same number of domain variables, the k_same_modulo constraint enforces a same_modulo constraint between each pair of consecutive sets.

Example

\[
\begin{align*}
\text{set} & \left\langle \text{set} := \langle \text{var} - 1, \text{var} - 9, \text{var} - 1, \text{var} - 5, \text{var} - 2, \text{var} - 1, \text{var} - 6, \text{var} - 4, \text{var} - 1, \text{var} - 1, \text{var} - 5, \text{var} - 5, \text{var} - 8, \text{var} - 3, \text{var} - 3, \text{var} - 4, \text{var} - 2, \text{var} - 4, \text{var} - 8, \text{var} - 7 \rangle, 3 \rangle.
\end{align*}
\]

The k_same_modulo constraint holds since:

- The first and second collections of variables are assigned 1 value in \(\{0, 3, \ldots, 3 \cdot k\}\), 3 values in \(\{1, 4, \ldots, 1 + 3 \cdot k\}\) and 2 values in \(\{2, 5, \ldots, 2 + 3 \cdot k\}\).
The second and third collections of variables are also assigned 1 value in 
\{0, 3, \ldots, 3 \cdot k\}, 3 values in \{1, 4, \ldots, 1 + 3 \cdot k\} and 2 values in \{2, 5, \ldots, 2 + 3 \cdot k\}.

Typical

\[
\begin{array}{l}
|\text{VARIABLES}| > 1 \\
M > 1
\end{array}
\]

Symmetries

- Items of SETS are \textit{permutable}.
- Items of SETS.set are \textit{permutable}.
- An occurrence of a value \(u\) of SETS.set.var can be \textit{replaced} by any other value \(v\) such that \(v\) is congruent to \(u\) modulo \(M\).

Arg. properties

Contractible wrt. SETS.

See also

- \textit{common keyword}: \texttt{k\_same (system of constraints)}.
- \textit{implies}: \texttt{k\_used\_by\_modulo}.
- \textit{part of system of constraints}: \texttt{same\_modulo}.
- \textit{used in graph description}: \texttt{same\_modulo}.

Keywords

- \textit{characteristic of a constraint}: sort based reformulation, modulo.
- \textit{combinatorial object}: permutation.
- \textit{constraint type}: system of constraints, decomposition.
<table>
<thead>
<tr>
<th><strong>Arc input(s)</strong></th>
<th>SETS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arc generator</strong></td>
<td>$PATH \rightarrow \text{collection}(\text{set1}, \text{set2})$</td>
</tr>
<tr>
<td><strong>Arc arity</strong></td>
<td>2</td>
</tr>
<tr>
<td><strong>Arc constraint(s)</strong></td>
<td>same_modulo(\text{set1.set, set2.set, M})</td>
</tr>
<tr>
<td><strong>Graph property(ies)</strong></td>
<td>$\text{NARC} =</td>
</tr>
</tbody>
</table>

**Graph model**

Parts (A) and (B) of Figure 5.369 respectively show the initial and final graph associated with the Example slot. To each vertex corresponds a collection of variables, while to each arc corresponds a same_modulo constraint.

![Figure 5.369: Initial and final graph of the k_same_modulo constraint](image)
### 5.193  \texttt{k\_same\_partition}

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
</table>

**Origin**

Derived from \texttt{same\_partition} and from \texttt{k\_same}.

**Constraint**

\texttt{k\_same\_partition(SETS, PARTITIONS)}

**Types**

<table>
<thead>
<tr>
<th>VARIABLES :</th>
<th>VALUES :</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{collection(var-dvar)}</td>
<td>\texttt{collection(val-int)}</td>
</tr>
</tbody>
</table>

**Arguments**

<table>
<thead>
<tr>
<th>SETS :</th>
<th>PARTITIONS :</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{collection(set - VARIABLES)}</td>
<td>\texttt{collection(p - VALUES)}</td>
</tr>
</tbody>
</table>

**Restrictions**

\[
\begin{align*}
\text{required(VARIABLES, var)} \\
|\text{VARIABLES}| & \geq 1 \\
|\text{VALUES}| & \geq 1 \\
\text{required(VALUES, val)} \\
\text{distinct(VALUES, val)} \\
\text{required(SETS, set)} \\
|\text{SETS}| & > 1 \\
\text{same\_size(SETS, set)} \\
\text{required(PARTITIONS, p)} \\
|\text{PARTITIONS}| & \geq 2
\end{align*}
\]

**Purpose**

Given a collection of \texttt{|SETS|} sets, each containing the same number of domain variables, the \texttt{k\_same\_partition} constraint enforces a \texttt{same\_partition} constraint between each pair of consecutive sets.
The first argument SETS of the \texttt{k\_same\_partition} constraint corresponds to 3 collections of variables, while the second argument PARTITIONS defines the 3 sets of values \{1, 3\}, \{4\} and \{2, 6\}. The \texttt{k\_same\_partition} constraint holds since:

- The first and second collections of variables are assigned 3 values in the \{1, 3\} as well as 3 values in \{2, 6\}.
- The second and third collections of variables are also assigned 3 values in the \{1, 3\} as well as 3 values in \{2, 6\}.

### Typical
| VARIABLES | > 1 |

### Symmetries
- Items of \texttt{SETS} are \texttt{permutable}.
- Items of \texttt{SETS.set} are \texttt{permutable}.
- Items of \texttt{PARTITIONS} are \texttt{permutable}.
- Items of \texttt{PARTITIONS.p} are \texttt{permutable}.
- An occurrence of a value of \texttt{SETS.set.var} can be replaced by any other value that also belongs to the same partition of \texttt{PARTITIONS}.

### Arg. properties
- \texttt{Contractible wrt. SETS}.

### See also
- \texttt{common keyword: k\_same (system of constraints)}.
- \texttt{implies: k\_used\_by\_partition}.
- \texttt{part of system of constraints: same\_partition}.
- \texttt{used in graph description: same\_partition}.
Keywords

characteristic of a constraint: sort based reformulation, partition.
combinatorial object: permutation.
constraint type: system of constraints, decomposition.
Arc input(s)  
\text{SETS}

Arc generator  
\text{PATH} \mapsto \text{collection}(\text{set1}, \text{set2})

Arc arity  
2

Arc constraint(s)  
\text{same\_partition}(\text{set1}.\text{set2}.\text{set}.\text{PARTITIONS})

Graph property(ies)  
\text{NARC} = |\text{SETS}| - 1

Graph model  
Parts (A) and (B) of Figure 5.370 respectively show the initial and final graph associated with the \textbf{Example} slot. To each vertex corresponds a collection of variables, while to each arc corresponds a \texttt{same\_partition} constraint.

![Graphs A and B](image-url)
### 5.194  k_used_by

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from used_by</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>( k_{used_by}(SETS) )</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>VARIABLES : collection(var-dvar)</td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>SETS : collection(set (-) VARIABLES)</td>
<td></td>
</tr>
</tbody>
</table>
| Restrictions| \[ \text{required}(\text{VARIABLES}, \text{var}) \]
|            | \(|\text{VARIABLES}| \geq 1\) |
|            | \[ \text{required}(\text{SETS}, \text{set}) \]
|            | \(|\text{SETS}| > 1\) |
|            | \text{non\_increasing\_size}(\text{SETS}, \text{set}) |

**Purpose**

Given \(|\text{SETS}| \) sets of domain variables, the \( k_{used\_by} \) constraint enforces a used_by constraint between each pair of consecutive sets.

The \( k_{used\_by} \) constraint holds since:

- The multiset of values \( \{\{1, 1, 1, 5\}\} \) associated with the second collection of variables is included into the multiset \( \{\{1, 1, 1, 2, 5\}\} \) associated with the first collection of variables.

- The multiset of values \( \{\{1, 1, 2\}\} \) associated with the third collection of variables is included into the multiset \( \{\{1, 1, 1, 2, 5\}\} \) associated with the second collection of variables.

**Typical**

\(|\text{VARIABLES}| > 1\)
Symmetries

- Items of SETS are permutable.
- Items of SETS.set are permutable.
- All occurrences of two distinct values of SETS.set.var can be swapped; all occurrences of a value of SETS.set.var can be renamed to any unused value.

Arg. properties

Contractible wrt. SETS.

Remark

Similarly to the k_same constraint [138], finding out whether the k_used_by constraint has a solution or not is NP-hard when we have more than one used_by constraint.

See also

- common keyword: k_used_by_interval, k_used_by_modulo, k_used_by_partition (system of constraints).
- implied by: k_same.
- part of system of constraints: used_by.
- used in graph description: used_by.

Keywords

- characteristic of a constraint: sort based reformulation.
- combinatorial object: multiset.
- constraint type: system of constraints, decomposition.
- modelling: inclusion.
Arc input(s) \text{SETS}
Arc generator \text{PATH} \rightarrow \text{collection(set1, set2)}
Arc arity 2
Arc constraint(s) \text{used_by(set1.set, set2.set)}
Graph property(ies) \text{NARC} = |\text{SETS}| - 1

Graph model

Parts (A) and (B) of Figure 5.371 respectively show the initial and final graph associated with the Example slot. To each vertex corresponds a collection of variables, while to each arc corresponds a \text{used_by} constraint.

![Graph Diagram](image)

Figure 5.371: Initial and final graph of the \text{k_used_by} constraint
5.195  **k_used_by_interval**

**DESCRIPTION**  
Derived from `used_by_interval` and from `k_used_by`.

**CONSTRAINT**  
`k_used_by_interval(SETS, SIZE_INTERVAL)`

**TYPE**  
VARIABLES : `collection(var−dvar)`

**ARGUMENTS**  
SETS : `collection(set−VARIABLES)`  
SIZE_INTERVAL : `int`

**RESTRICTIONS**  
required(VARIABLES, var)  
|VARIABLES| ≥ 1  
required(SETS, set)  
|SETS| > 1  
non_increasing_size(SETS, set)  
SIZE_INTERVAL > 0

**PURPOSE**  
Given |SETS| sets of domain variables, the `k_used_by_interval` constraint enforces a `used_by_interval` constraint between each pair of consecutive sets.

**EXAMPLE**  
In the example, the second argument `SIZE_INTERVAL = 3` defines the following family of intervals `[3·k, 3·k + 2]`, where `k` is an integer. Consequently, the `k_used_by_interval` constraint holds since:

- The first collection of variables is assigned 4 values in the interval `[0, 2]` as well as 2 values in the interval `[6, 8]`, while the second collection of variables is assigned no more values in the previous two intervals.
- The second collection of variables is assigned 2 values in the interval `[0, 2]` as well as 2 values in the interval `[6, 8]`, while the third collection of variables is assigned no more values in the previous two intervals.

**TYPICAL**  
|VARIABLES| > 1  
SIZE_INTERVAL > 0
Symmetries

- Items of SETS are permutable.
- Items of SETS.set are permutable.
- An occurrence of a value of SETS.set.var that belongs to the k-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.

Arg. properties

Contractible wrt. SETS.

See also

common keyword: k_used_by (system of constraints).
implied by: k_same_interval.
part of system of constraints: used_by_interval.
used in graph description: used_by_interval.

Keywords

characteristic of a constraint: sort based reformulation.
constraint type: system of constraints, decomposition.
modelling: inclusion, interval.
Arc input(s)  SETS
Arc generator  $PATH \rightarrow \text{collection}(\text{set1}, \text{set2})$
Arc arity  2
Arc constraint(s)  $\text{used}_{\text{by}}_{\text{interval}}(\text{set1.set}, \text{set2.set}, \text{SIZE.INTERVAL})$
Graph property(ies)  $\text{NARC} = |\text{SETS}| - 1$

Graph model

Parts (A) and (B) of Figure 5.372 respectively show the initial and final graph associated with the Example slot. To each vertex corresponds a collection of variables, while to each arc corresponds a $\text{used}_{\text{by}}_{\text{interval}}$ constraint.

![Graph model](image)

(A)  (B)

Figure 5.372: Initial and final graph of the $k_{\text{used}_{\text{by}}_{\text{interval}}}$ constraint
5.196  \texttt{k\textunderscore used\textunderscore by\textunderscore modulo} \\

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Derived from \texttt{used\textunderscore by\textunderscore modulo} and from \texttt{k\textunderscore used\textunderscore by}.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>\texttt{k\textunderscore used\textunderscore by\textunderscore modulo}(SETS, M)</td>
<td></td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>\texttt{VARIABLES} : \texttt{collection(var\textminus dvar)}</td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>SETS : \texttt{collection(set \textminus VARIABLES)}</td>
<td></td>
</tr>
<tr>
<td>M : int</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>\texttt{required}(\texttt{VARIABLES, var})</td>
<td></td>
</tr>
<tr>
<td>(</td>
<td>\texttt{VARIABLES}</td>
<td>\geq 1)</td>
</tr>
<tr>
<td>\texttt{required}(\texttt{SETS, set})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(</td>
<td>\texttt{SETS}</td>
<td>&gt; 1)</td>
</tr>
<tr>
<td>\texttt{non\textunderscore increasing\textunderscore size}(\texttt{SETS, set})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M &gt; 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>Given (</td>
<td>\texttt{SETS}</td>
</tr>
</tbody>
</table>
| **Example** | \begin{pmatrix} \texttt{var} \textminus 1, \\
| & \texttt{set} = \langle \texttt{var} \textminus 9, \\
| & \texttt{var} \textminus 4, \\
| & \texttt{var} \textminus 5, \\
| & \texttt{var} \textminus 2, \\
| & \texttt{var} \textminus 1, \\
| & \texttt{set} = \langle \texttt{7, 1, 2, 5} , \\
| & \texttt{set} = \langle \texttt{1, 1} \rangle \end{pmatrix}, 3 |       |       |
| The \texttt{k\textunderscore used\textunderscore by\textunderscore modulo} \texttt{constraint holds since:} |       |       |
| • The first collection of variables is assigned 1 value in \(\{0, 3, \ldots, 3 \cdot k\}\), 3 values in \(\{1, 4, \ldots, 1 + 3 \cdot k\}\) and 2 values in \(\{2, 5, \ldots, 2 + 3 \cdot k\}\), while the second collection of variables is assigned no more values in the previous three sets of values. |       |       |
| • The second collection of variables is assigned 2 values in \(\{0, 3, \ldots, 3 \cdot k\}\) and 2 values in \(\{2, 5, \ldots, 2 + 3 \cdot k\}\), while the third collection of variables is assigned no more values in the previous three sets of values. |       |       |
| **Typical** | \(|\texttt{VARIABLES}| > 1\) |       |       |
| M > 1 |       |       |
| **Symmetries** |       |       |
| • Items of \texttt{SETS} are \texttt{permutable}. |       |       |
| • Items of \texttt{SETS.set} are \texttt{permutable}. |       |       |
| • An occurrence of a value \(u\) of \texttt{SETS.set.var} can be \texttt{replaced} by any other value \(v\) such that \(v\) is congruent to \(u\) modulo \(M\). |       |       |
Arg. properties  Contractible wrt. SETS.

See also  common keyword: k_used_by(system of constraints).
          implied by: k_same_modulo.
          part of system of constraints: used_by_modulo.
          used in graph description: used_by_modulo.

Keywords  characteristic of a constraint: modulo, sort based reformulation.
          constraint type: system of constraints, decomposition.
          modelling: inclusion.
Arc input(s)  

Arc generator  

Arc arity  

Arc constraint(s)  

Graph property(ies)  

Graph model  

Parts (A) and (B) of Figure 5.373 respectively show the initial and final graph associated with the Example slot. To each vertex corresponds a collection of variables, while to each arc corresponds a used by modulo constraint.

Figure 5.373: Initial and final graph of the k used by modulo constraint
## 5.197 k_used_by_partition

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Derived from <code>used_by_partition</code> and from <code>k_used_by_partition</code></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td><code>k_used_by_partition(SETS, PARTITIONS)</code></td>
<td></td>
</tr>
<tr>
<td><strong>Types</strong></td>
<td><code>VARIABLES : collection(var−dvar)</code></td>
<td><code>VALUES : collection(val−int)</code></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td><code>SETS : collection(set − VARIABLES)</code></td>
<td><code>PARTITIONS : collection(p − VALUES)</code></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td><code>required(VARIABLES, var)</code></td>
<td>`</td>
</tr>
<tr>
<td></td>
<td><code>VALUES ≥ 1</code></td>
<td><code>required(VVALUES, val)</code></td>
</tr>
<tr>
<td></td>
<td><code>distinct(VALUES, val)</code></td>
<td><code>required(SETS, set)</code></td>
</tr>
<tr>
<td></td>
<td>`</td>
<td>SETS</td>
</tr>
<tr>
<td></td>
<td><code>required(PARTITIONS, p)</code></td>
<td>`</td>
</tr>
</tbody>
</table>

### Purpose

Given `|SETS|` sets of domain variables, the `k_used_by_partition` constraint enforces a `used_by_partition` constraint between each pair of consecutive sets.

```
\[
\begin{pmatrix}
\text{var} - 1, \\
\text{var} - 9, \\
\text{set} - \{\text{var} - 1, \\
\text{var} - 6, \\
\text{var} - 2, \\
\text{var} - 3\}, \\
\text{set} - \{1, 3, 6, 6\}, \\
\text{set} - \{2, 2\}, \\
\text{p} - \{1, 3\}, \\
\text{p} - \{4\}, \\
\text{p} - \{2, 6\}
\end{pmatrix}
\]
```

The `k_used_by_partition` constraint holds since:

- The first collection of variables is assigned 3 values in `{1, 3}`, 0 value in `{4}` and 2 values in `{2, 6}`, while the second collection of variables is assigned no more values in the previous three sets of values.
- The second collection of variables is assigned 2 values in `{1, 3}`, 0 value in `{4}` and 2 values in `{2, 6}`, while the third collection of variables is assigned no more values in the previous three sets of values.
| Typical | $|\text{VARIBALES}| > 1$

| Symmetries |  
| --- | --- |
| • Items of $\text{SETS}$ are **permutable**. |  
| • Items of $\text{SETS.set}$ are **permutable**. |  
| • Items of $\text{PARTITIONS}$ are **permutable**. |  
| • Items of $\text{PARTITIONS.p}$ are **permutable**. |  
| • An occurrence of a value of $\text{SETS.set.var}$ can be replaced by any other value that also belongs to the same partition of $\text{PARTITIONS}$. |  

| Arg. properties | **Contractible** wrt. $\text{SETS}$. |

<table>
<thead>
<tr>
<th>See also</th>
<th><strong>common keyword</strong>: $\text{k_used_by}$ (<strong>system of constraints</strong>).</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>implied by</strong>: $\text{k_same_partition}$.</td>
<td></td>
</tr>
<tr>
<td><strong>part of system of constraints</strong>: $\text{used_by_partition}$.</td>
<td></td>
</tr>
<tr>
<td><strong>used in graph description</strong>: $\text{used_by_partition}$.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Keywords</th>
<th><strong>characteristic of a constraint</strong>: partition, sort based reformulation.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>constraint type</strong>: system of constraints, decomposition.</td>
<td></td>
</tr>
</tbody>
</table>

Arc input(s)  \( \text{SETS} \)
Arc generator  \( \text{PATH} \mapsto \text{collection}(\text{set1, set2}) \)
Arc arity  2
Arc constraint(s)  \text{used_by_partition}(\text{set1.set, set2.set, PARTITIONS})
Graph property(ies)  \( \text{NARC} = |\text{SETS}| - 1 \)

Graph model
Parts (A) and (B) of Figure 5.374 respectively show the initial and final graph associated with the Example slot. To each vertex corresponds a collection of variables, while to each arc corresponds a \text{used_by_partition} constraint.

(A) (B)

Figure 5.374: Initial and final graph of the \text{k_used_by_partition} constraint
### 5.198 length_first_sequence

**DESCRIPTION**

Inspired by stretch_path

**LINKS**

length_first_sequence(LEN, VARIABLES)

**ARGUMENTS**

- LEN : dvar
- VARIABLES : collection(var–dvar)

**RESTRICTIONS**

- LEN ≥ 0
- LEN ≤ |VARIABLES|
- required(VARIABLES.var)

**PURPOSE**

LEN is the length of the maximum sequence of variables that take the same value that contains the first variable of the collection VARIABLES (or 0 if the collection is empty).

**EXAMPLE**

\[
\begin{pmatrix}
\text{var} - 4, \\
\text{var} - 4, \\
\text{var} - 4, \\
3, \\
\text{var} - 5, \\
\text{var} - 5, \\
\text{var} - 4
\end{pmatrix}
\]

The length_first_sequence constraint holds since the sequence associated with the first value of the collection VARIABLES = ⟨4, 4, 4, 5, 5, 4⟩ spans over three consecutive variables.

**TYPICAL**

- LEN < |VARIABLES|
- |VARIABLES| > 1

**SYMMETRY**

All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

**REFORMULATION**

Without loss of generality let assume that the collection VARIABLES = ⟨V₁, V₂, . . . , Vₙ⟩ has more than one variable. By introducing 2 · n − 1 0-1 variables, the length_first_sequence(LEN, VARIABLES) constraint can be expressed in term of 2 · n − 1 reified constraints and one arithmetic constraint (i.e., a sum_ctr constraint). We first introduce n − 1 variables that are respectively set to 1 if and only if two given consecutive variables of the collection VARIABLES are equal:

\[
\begin{align*}
B_{1,2} &\iff V_1 = V_2, \\
B_{2,3} &\iff V_2 = V_3, \\
&\ldots \\
B_{n-1,n} &\iff V_{n-1} = V_n.
\end{align*}
\]

We then introduce n variables A₁, A₂, . . . , Aₙ that are respectively associated to the different sliding sequences starting on the first variable of the sequence V₁ V₂ . . . Vₙ. Variable
$A_i$ is set to 1 if and only if $V_1 = V_2 = \ldots = V_i$:

$A_1 = 1,$
$A_2 \iff B_{1,2} \land A_1,$
$A_3 \iff B_{2,3} \land A_2,$

.................

$A_n \iff B_{n-1,n} \land A_{n-1}.$

Finally we state the following arithmetic constraint:

$\text{LEN} = A_1 + A_2 + \ldots + A_n.$

**See also**

common keyword: length_last_sequence (counting constraint, sequence).

**Keywords**

characteristic of a constraint: automaton, automaton with counters.

combinatorial object: sequence.

constraint network structure: sliding cyclic(1) constraint network(2).

constraint type: value constraint, counting constraint.
Automaton

Figure 5.375 depicts the automaton associated with the length_first_sequence constraint. To each pair of consecutive variables $(\text{VAR}_i, \text{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable $S_i$. The following signature constraint links $\text{VAR}_i, \text{VAR}_{i+1}$ and $S_i$: $\text{VAR}_0 = \text{VAR}_{i+1} \Leftrightarrow S_i$.

Figure 5.375: Automaton of the length_first_sequence constraint when $|\text{VARIABLES}| \geq 2$

Figure 5.376: Hypergraph of the reformulation corresponding to the automaton of the length_first_sequence constraint
5.199 **length_last_sequence**

### DESCRIPTION

**Origin**
Inspired by `stretch_path`

**Constraint**
`length_last_sequence(LEN, VARIABLES)`

**Arguments**
- `LEN`: `dvar`
- `VARIABLES`: `collection(var−dvar)`

**Restrictions**
- `LEN ≥ 0`
- `LEN ≤ |VARIABLES|`
- `required(VARIABLES, var)`

**Purpose**
`LEN` is the length of the maximum sequence of variables that take the same value that contains the last variable of the collection `VARIABLES` (or 0 if the collection is empty).

### Example

<table>
<thead>
<tr>
<th>var − 4, var − 4, var − 4,</th>
<th>var − 5, var − 5, var − 4</th>
</tr>
</thead>
</table>

The `length_last_sequence` constraint holds since the sequence associated with the last value of the collection `VARIABLES = ⟨4, 4, 4, 5, 5⟩` spans over one single variable.

**Typical**
- `LEN < |VARIABLES|`
- `|VARIABLES| > 1`

**Symmetry**
All occurrences of two distinct values of `VARIABLES.var` can be swapped; all occurrences of a value of `VARIABLES.var` can be renamed to any unused value.

**Reformulation**
Without loss of generality let assume that the collection `VARIABLES = ⟨V₁, V₂, . . . , Vₙ⟩` has more than one variable. By introducing `2 · n − 1` 0-1 variables, the `length_last_sequence(LEN, VARIABLES)` constraint can be expressed in term of `2 · n − 1` reified constraints and one arithmetic constraint (i.e., a `sum_ctr` constraint). We first introduce `n − 1` variables that are respectively set to 1 if and only if two given consecutive variables of the collection `VARIABLES` are equal:
- `B_{n−1,n} ⇔ V_{n−1} = V_n`
- `B_{n−2,n−1} ⇔ V_{n−2} = V_{n−1}`

```
.................
```
- `B_{1,2} ⇔ V₁ = V₂`

We then introduce `n` variables `Aₙ, A_{n−1}, . . . , A₁` that are respectively associated to the different sliding sequences ending on the last variable of the sequence `V₁ V₂ . . . Vₙ`. Variable `A₁` is set to 1 if and only if `Vₙ = V_{n−1} = . . . = V₁`: 
\[ A_n = 1, \]
\[ A_{n-1} \leftrightarrow B_{n-1,n} \land A_n, \]
\[ A_{n-2} \leftrightarrow B_{n-2,n-1} \land A_{n-1}, \]
\[ \ldots \]
\[ A_1 \leftrightarrow B_{1,2} \land A_2. \]

Finally we state the following arithmetic constraint:
\[ \text{LEN} = A_n + A_{n-1} + \ldots + A_1. \]

**See also**

common keyword: length_first_sequence (counting constraint, sequence).

**Keywords**

characteristic of a constraint: automaton, automaton with counters.

combinatorial object: sequence.

constraint network structure: sliding cyclic(1) constraint network(2).

constraint type: value constraint, counting constraint.
Figure 5.377 depicts the automaton associated with the length_last_sequence constraint. To each pair of consecutive variables \((\text{VAR}_i, \text{VAR}_{i+1})\) of the collection VARIABLES corresponds a signature variable \(S_i\). The following signature constraint links \(\text{VAR}_i, \text{VAR}_{i+1}\) and \(S_i\): \(\text{VAR}_i = \text{VAR}_{i+1} \iff S_i\).

\[
\begin{align*}
\text{VAR}_1 & \not= \text{VAR}_{1+1} & (C=1) \\
\text{VAR}_1 & = \text{VAR}_{1+1} & (C=C+1)
\end{align*}
\]

Figure 5.377: Automaton of the length_last_sequence constraint when \(|\text{VARIABLES}| \geq 2\)

Figure 5.378: Hypergraph of the reformulation corresponding to the automaton of the length_last_sequence constraint
5.200 leq

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Arithmetic.</td>
</tr>
<tr>
<td>Constraint</td>
<td>leq(VAR1, VAR2)</td>
</tr>
<tr>
<td>Synonyms</td>
<td>rel, xlteqy.</td>
</tr>
<tr>
<td>Arguments</td>
<td>VAR1 : dvar</td>
</tr>
<tr>
<td></td>
<td>VAR2 : dvar</td>
</tr>
<tr>
<td>Purpose</td>
<td>Enforce the fact that the first variable is less than or equal to the second variable.</td>
</tr>
<tr>
<td>Example</td>
<td>(1, 8)</td>
</tr>
<tr>
<td></td>
<td>The leq constraint holds since 1 is greater than or equal to 8.</td>
</tr>
<tr>
<td>Typical</td>
<td>VAR1 &lt; VAR2</td>
</tr>
<tr>
<td>Symmetries</td>
<td>• VAR1 can be replaced by any value ≤ VAR2.</td>
</tr>
<tr>
<td></td>
<td>• VAR2 can be replaced by any value ≥ VAR1.</td>
</tr>
<tr>
<td>Systems</td>
<td>leq in Choco, rel in Gecode, xlteqy in JaCoP, #== in SICStus.</td>
</tr>
<tr>
<td>See also</td>
<td>common keyword: neq (binary constraint, arithmetic constraint).</td>
</tr>
<tr>
<td></td>
<td>generalisation: leq.cst (constant added).</td>
</tr>
<tr>
<td></td>
<td>implied by: eq, lt.</td>
</tr>
<tr>
<td></td>
<td>implies (if swap arguments): geq.</td>
</tr>
<tr>
<td></td>
<td>negation: gt.</td>
</tr>
<tr>
<td>Keywords</td>
<td>constraint arguments: binary constraint.</td>
</tr>
<tr>
<td></td>
<td>constraint type: predefined constraint, arithmetic constraint.</td>
</tr>
<tr>
<td></td>
<td>filtering: arc-consistency.</td>
</tr>
</tbody>
</table>
### 5.201 leq_cst

**DESCRIPTION**

<table>
<thead>
<tr>
<th>Origin</th>
<th>Arithmetic.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint</td>
<td>leq_cst(VAR1, VAR2, CST2)</td>
</tr>
<tr>
<td>Arguments</td>
<td>VAR1 : dvar</td>
</tr>
<tr>
<td>Purpose</td>
<td>Enforce the fact that the first variable is less than or equal to the sum of the second variable and the constant.</td>
</tr>
<tr>
<td>Example</td>
<td>(5, 2, 4)</td>
</tr>
<tr>
<td>Typical</td>
<td>CST2 ≠ 0</td>
</tr>
<tr>
<td>Symmetries</td>
<td>VAR1 can be replaced by any value ≤ VAR2 + CST2.</td>
</tr>
<tr>
<td>See also</td>
<td>common keyword: geq_cst (binary constraint, arithmetic constraint).</td>
</tr>
<tr>
<td>Keywords</td>
<td>constraint arguments: binary constraint.</td>
</tr>
</tbody>
</table>
## 5.202  lex2

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
</table>

| **Origin** | [155] |
| **Constraint** | lex2(MATRIX) |
| **Synonyms** | double_lex, row_and_column_lex. |
| **Type** | VECTOR : collection(var − dvar) |
| **Argument** | MATRIX : collection(vec − VECTOR) |
| **Restrictions** | |
| | | |
| | | |
| | | |
| | | |
| | | |
| **Purpose** | Given a matrix of domain variables, enforces that both adjacent rows, and adjacent columns are lexicographically ordered (adjacent rows and adjacent columns can be equal). |
| **Example** | The lex2 constraint holds since: |
| | • The first row \( \langle 2, 2, 3 \rangle \) is lexicographically less than or equal to the second row \( \langle 2, 3, 1 \rangle \). |
| | • The first column \( \langle 2, 2 \rangle \) is lexicographically less than or equal to the second column \( \langle 2, 3 \rangle \). |
| | • The second column \( \langle 2, 3 \rangle \) is lexicographically less than or equal to the third column \( \langle 3, 1 \rangle \). |
| **Typical** | |
| | | |
| | | |
| **Symmetry** | One and the same constant can be added to the var attribute of all items of MATRIX.vec. |
| **Usage** | A symmetry-breaking constraint. |
| **Remark** | The idea of this symmetry-breaking constraint can already be found in the following articles of A. Lubiw [248, 249]. In block designs you sometimes want repeated blocks, so using the non-strict order would be required in this case. |
Reformulation

The lex2 constraint can be expressed as a conjunction of two lex_chain_llesseq constraints: A first lex_chain_llesseq constraint on the MATRIX argument and a second lex_chain_llesseq constraint on the transpose of the MATRIX argument.

Systems

lex2 in MiniZinc.

See also

common keyword: allperm, lex_llesseq (matrix symmetry, lexicographic order).

implied by: strict_lex2.

implies: lex_chain_llesseq.

part of system of constraints: lex_chain_llesseq.

Keywords

constraint type: predefined constraint, system of constraints, order constraint.

modelling: matrix, matrix model.

symmetry: symmetry, matrix symmetry, lexicographic order.
5.203  lex_alldifferent

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>J. Pearson</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>lex_alldifferent(VECTORS)</td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>lex_alldiff, lex_alldistinct, alldiff_on_tuples, alldifferent_on_tuples, alldistinct_on_tuples.</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>VECTOR : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>VECTORS : collection(vec − VECTOR)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>All the vectors of the collection VECTORS are distinct. Two vectors ((u_1, u_2, \ldots, u_n)) and ((v_1, v_2, \ldots, v_n)) are distinct if and only if there exists (i \in [1, n]) such that (u_i \neq v_i).</td>
<td></td>
</tr>
</tbody>
</table>
| Example     | (\[
|             | \quad \begin{cases} 
|             | \quad \text{vec} - (5, 2, 3), 
|             | \quad \text{vec} - (5, 2, 6), 
|             | \quad \text{vec} - (5, 3, 3) \end{cases} \]
|             | ) |       |
|             | The lex_alldifferent constraint holds since: |       |
|             | \bullet The first vector \((5, 2, 3)\) and the second vector \((5, 2, 6)\) of the VECTORS collection differ in their third component (i.e., \(3 \neq 6\)). |       |
|             | \bullet The first vector \((5, 2, 3)\) and the third vector \((5, 3, 3)\) of the VECTORS collection differ in their second component (i.e., \(2 \neq 3\)). |       |
|             | \bullet The second vector \((5, 2, 6)\) and the third vector \((5, 3, 3)\) of the VECTORS collection differ in their second and third components (i.e., \(2 \neq 3\) and \(6 \neq 3\)). |       |
| Typical     | |       |
|             | |       |
| Symmetries  | |       |
|             | \bullet Items of VECTORS are permutable. |       |
|             | \bullet Items of VECTORS.vec are permutable (same permutation used). |       |
|             | \bullet All occurrences of two distinct tuples of values of VECTORS.vec can be swapped; all occurrences of a tuple of values of VECTORS.vec can be renamed to any unused tuple of values. |       |
Arg. properties

- Contractible wrt. VECTORS.
- Extensible wrt. VECTORS.\textit{vec} (add items at same position).

Usage

When the vectors have two components, the \textit{lexאללדיפרנטי} constraint allows to directly enforce difference constraints between pairs of variables. Such difference constraints occur for instance in block design problems (e.g., Steiner triples, Kirkman schoolgirls problem). However, in all these problems a same variable may occur in more than one pair of variables. Consequently, arc-consistency is not achieved any more by the filtering algorithm described in [315].

Algorithm

A filtering algorithm achieving arc-consistency for the \textit{lexאללדיפרנטי} constraint is proposed by C.-G. Quimper and T. Walsh in [315] and a longer version is available in [316] and in [317].

Reformulation

The \textit{lexאללדיפרנטי}(VECTORS) constraint can be expressed as a clique of \textit{lexאללדיפרנטי} constraints. By associating a \(n\)-dimensional box for which all sizes are equal to 1, one can also express the \textit{lex אללдיפרנטי}(VECTORS) constraint as a \textit{diffn} or a \textit{geost} constraint. Enforcing all the \(n\)-dimensional boxes to not overlap is equivalent as enforcing all the vectors to be distinct. In the context of the multidimensional sweep algorithm of the \textit{geost} constraint [36], it makes more sense to make a complete sweep over the domain of each variable in order not to only restrict the minimum and maximum value of each variable.

See also

generalisation: \textit{diffn} (vector replaced by orthotope), \textit{geost} (vector replaced by object).

implied by: \textit{lex_chain_less}.

part of system of constraints: \textit{lex אללדיפרנטי}.

specialisation: \textit{alldifferent} (vector replaced by variable).

used in graph description: \textit{lex אללדיפרנטי}.

Keywords

characteristic of a constraint: vector.

constraint type: system of constraints, decomposition.

filtering: bipartite matching, arc-consistency.

modelling: difference between pairs of variables.
Arc input(s) | VECTORS
---|---
Arc generator | $\text{CLIQUE}(\langle \rangle \rightarrow \text{collection}(\text{vectors1}, \text{vectors2})$
Arc arity | 2
Arc constraint(s) | $\text{lex.different}(\text{vectors1.vec}, \text{vectors2.vec})$
Graph property(ies) | $\text{NARC} = |\text{VECTORS}| \cdot (|\text{VECTORS}| - 1)/2$

Graph model

Parts (A) and (B) of Figure 5.379 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

![Graph Diagram](image)

Figure 5.379: Initial and final graph of the lex_alldifferent constraint

Signature

Since we use the $\text{CLIQUE}(\langle \rangle)$ arc generator on the VECTORS collection the number of arcs of the initial graph is equal to $|\text{VECTORS}| \cdot (|\text{VECTORS}| - 1)/2$. For this reason we can rewrite $\text{NARC} = |\text{VECTORS}| \cdot (|\text{VECTORS}| - 1)/2$ to $\text{NARC} \geq |\text{VECTORS}| \cdot (|\text{VECTORS}| - 1)/2$ and simplify $\text{NARC}$ to $\text{NARC}$. 
5.204  lex_between

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[90]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>lex_between(LOWER_BOUND, VECTOR, UPPER_BOUND)</td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td>between.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOWER_BOUND : collection(var−int)</td>
</tr>
<tr>
<td>VECTOR    : collection(var−dvar)</td>
</tr>
<tr>
<td>UPPER_BOUND : collection(var−int)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>required(LOWER_BOUND, var)</td>
</tr>
<tr>
<td>required(VECTOR, var)</td>
</tr>
<tr>
<td>required(UPPER_BOUND, var)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>The vector VECTOR is lexicographically greater than or equal to the fixed vector LOWER_BOUND and lexicographically smaller than or equal to the fixed vector UPPER_BOUND.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>(⟨5, 2, 3, 9⟩, ⟨5, 2, 6, 2⟩, ⟨5, 2, 6, 3⟩)</td>
</tr>
</tbody>
</table>

The lex_between constraint holds since:

- The vector VECTOR = ⟨5, 2, 6, 2⟩ is greater than or equal to the vector LOWER_BOUND = ⟨5, 2, 3, 9⟩.

- The vector VECTOR = ⟨5, 2, 6, 2⟩ is less than or equal to the vector UPPER_BOUND = ⟨5, 2, 6, 3⟩.

<table>
<thead>
<tr>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symmetries</th>
</tr>
</thead>
<tbody>
<tr>
<td>• LOWER_BOUND.var can be decreased.</td>
</tr>
<tr>
<td>• UPPER_BOUND.var can be increased.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arg. properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suffix-contractible wrt. LOWER_BOUND, VECTOR and UPPER_BOUND (remove items from same position).</td>
</tr>
</tbody>
</table>
This constraint does usually not occur explicitly in practice. However it shows up indirectly in the context of the \texttt{lex\_chain\_less} and the \texttt{lex\_chain\_lesseq} constraints: in order to have a complete filtering algorithm for the \texttt{lex\_chain\_less} and the \texttt{lex\_chain\_lesseq} constraints one has to come up with a complete filtering algorithm for the \texttt{lex\_between} constraint. The reason is that the \texttt{lex\_chain\_less} as well as the \texttt{lex\_chain\_lesseq} constraints both compute feasible lower and upper bounds for each vector they mention. Therefore one ends up with a \texttt{lex\_between} constraint for each vector of the \texttt{lex\_chain\_less} and \texttt{lex\_chain\_lesseq} constraints.

The \texttt{lex\_between(\texttt{LOWER\_BOUND}, \texttt{VECTORS}, \texttt{UPPER\_BOUND})} constraint can be expressed as the conjunction \texttt{lex\_lesseq(\texttt{LOWER\_BOUND}, \texttt{VECTORS})} \land \texttt{lex\_lesseq(\texttt{VECTORS}, \texttt{UPPER\_BOUND})}.

\texttt{lex\_Chain\_Eq} in \texttt{Choco}, \texttt{lex\_chain} in \texttt{SICStus}.

\texttt{common keyword: lex\_chain\_less, lex\_chain\_lesseq, lex\_greater, lex\_greatereq, lex\_less} (\textit{lexicographic order}).

\texttt{part of system of constraints: lex\_lesseq}.

\texttt{characteristic of a constraint: vector, automaton, automaton without counters, reified automaton constraint,}

\texttt{constraint network structure: Berge-acyclic constraint network.}

\texttt{constraint type: order constraint, system of constraints.}

\texttt{filtering: arc-consistency.}

\texttt{symmetry: symmetry, lexicographic order.
**Automaton**

Figure 5.380 depicts the automaton associated with the \texttt{lex\_between} constraint. Let $L_i$, $V_i$, and $U_i$ respectively be the var attributes of the $i^{th}$ items of the LOWER\_BOUND, the VECTOR and the UPPER\_BOUND collections. To each triple $(L_i, V_i, U_i)$ corresponds a signature variable $S_i$ as well as the following signature constraint:

\[
\begin{align*}
(L_i < V_i) \land (V_i < U_i) & \iff S_i = 0 \\
(L_i < V_i) \land (V_i = U_i) & \iff S_i = 1 \\
(L_i < V_i) \land (V_i > U_i) & \iff S_i = 2 \\
(L_i = V_i) \land (V_i < U_i) & \iff S_i = 3 \\
(L_i = V_i) \land (V_i = U_i) & \iff S_i = 4 \\
(L_i = V_i) \land (V_i > U_i) & \iff S_i = 5 \\
(L_i > V_i) \land (V_i < U_i) & \iff S_i = 6 \\
(L_i > V_i) \land (V_i = U_i) & \iff S_i = 7 \\
(L_i > V_i) \land (V_i > U_i) & \iff S_i = 8.
\end{align*}
\]

Figure 5.380: Automaton of the \texttt{lex\_between} constraint
Figure 5.381: Hypergraph of the reformulation corresponding to the automaton of the \texttt{lex_between} constraint
## 5.205 lex_chain_less

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[90]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>lex_chain_less(VECTORS)</td>
<td></td>
</tr>
<tr>
<td>Usual name</td>
<td>lex_chain</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>VECTOR : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>VECTORS : collection(vec − VECTOR)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>For each pair of consecutive vectors VECTOR_i and VECTOR_{i+1} of the VECTORS collection we have that VECTOR_i is lexicographically strictly less than VECTOR_{i+1}. Given two vectors, ( \vec{X} ) and ( \vec{Y} ) of ( n ) components, ( \langle X_0, \ldots, X_{n-1} \rangle ) and ( \langle Y_0, \ldots, Y_{n-1} \rangle ), ( \vec{X} ) is lexicographically strictly less than ( \vec{Y} ) if and only if ( X_0 &lt; Y_0 ) or ( X_0 = Y_0 ) and ( \langle X_1, \ldots, X_{n-1} \rangle ) is lexicographically strictly less than ( \langle Y_1, \ldots, Y_{n-1} \rangle ).</td>
<td></td>
</tr>
</tbody>
</table>
| Example     | \begin{pmatrix} vec − (5, 2, 3, 9), \\
|             | vec − (5, 2, 6, 2), \\
|             | vec − (5, 2, 6, 3) \end{pmatrix} |       |
| Typical     | | |
| Arg. properties | | |
| Usage       | This constraint was motivated for breaking symmetry: more precisely when one wants to lexicographically order the consecutive columns of a matrix of decision variables. A further motivation is that using a set of lexicographic ordering constraints between two vectors does usually not allows to come up with a complete pruning. |       |
Algorithm

A filtering algorithm achieving arc-consistency for a chain of lexicographical ordering constraints is presented in [90].

Six different ways of integrating a chain of lexicographical ordering constraints within non-overlapping constraints like diffn or geost and within their corresponding necessary condition like the cumulative constraint are shown in [2].

Systems

lexChain in Choco, lex_chain in SICStus.

See also

common keyword: geost (symmetry, lexicographic ordering on the origins of tasks, rectangles, ...), lex_between, lex_greater, lex_greatereq, lex_leesseq (lexicographic order).

implied by: strict_lex2.

implies: lex_alldifferent, lex_chain_leesseq.

part of system of constraints: lex_less.

related: cumulative, diffn (lexicographic ordering on the origins of tasks, rectangles, ...).

system of constraints: strict_lex2.

used in graph description: lex_less.

Keywords

application area: floor planning problem.

characteristic of a constraint: vector.

constraint type: decomposition, order constraint, system of constraints.

filtering: arc-consistency.

heuristics: heuristics and lexicographical ordering.

modelling: degree of diversity of a set of solutions.

modelling exercises: degree of diversity of a set of solutions.

symmetry: symmetry, matrix symmetry, lexicographic order.
### Arc input(s)
VECTORS

### Arc generator
$PATH \xrightarrow{\text{collection}} (\text{vectors}_1, \text{vectors}_2)$

### Arc arity
2

### Arc constraint(s)
$\text{lex.less}(\text{vectors}_1.\text{vec}, \text{vectors}_2.\text{vec})$

### Graph property(ies)
$\text{NARC} = |\text{VECTORS}| - 1$

#### Graph model
Parts (A) and (B) of Figure 5.382 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold. The $\text{lex.chain.less}$ constraint holds since all the arc constraints of the initial graph are satisfied.

![Graph Model](image)

Figure 5.382: Initial and final graph of the $\text{lex.chain.less}$ constraint

#### Signature
Since we use the $PATH$ arc generator on the VECTORS collection the number of arcs of the initial graph is equal to $|\text{VECTORS}| - 1$. For this reason we can rewrite $\text{NARC} = |\text{VECTORS}| - 1$ to $\text{NARC} \geq |\text{VECTORS}| - 1$ and simplify $\text{NARC}$ to $\text{NARC}$. 
5.206 lex_chain_lesseq

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[90]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>lex_chain_lesseq(VECTORS)</td>
<td></td>
</tr>
<tr>
<td><strong>Usual name</strong></td>
<td>lex_chain</td>
<td></td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>VECTOR : collection(var − dvar)</td>
<td></td>
</tr>
<tr>
<td><strong>Argument</strong></td>
<td>VECTORS : collection(vec − VECTOR)</td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>required(VECTORS, var)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>required(VECTORS, vec)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>same_size(VECTORS, vec)</td>
<td></td>
</tr>
</tbody>
</table>

For each pair of consecutive vectors VECTOR$_i$ and VECTOR$_{i+1}$ of the VECTORS collection we have that VECTOR$_i$ is lexicographically less than or equal to VECTOR$_{i+1}$. Given two vectors, $\vec{X}$ and $\vec{Y}$ of $n$ components, $(X_0, \ldots, X_{n-1})$ and $(Y_0, \ldots, Y_{n-1})$, $\vec{X}$ is lexicographically less than or equal to $\vec{Y}$ if and only if $n = 0$ or $X_0 < Y_0$ or $X_0 = Y_0$ and $(X_1, \ldots, X_{n-1})$ is lexicographically less than or equal to $(Y_1, \ldots, Y_{n-1})$.

**Purpose**

The lex_chain_lesseq constraint holds since:

- The first vector $(5, 2, 3, 9)$ of the VECTORS collection is lexicographically less than or equal to the second vector $(5, 2, 6, 2)$ of the VECTORS collection.
- The second vector $(5, 2, 6, 2)$ of the VECTORS collection is lexicographically less than or equal to the third vector $(5, 2, 6, 2)$ of the VECTORS collection.

**Typical**

<table>
<thead>
<tr>
<th>VECTOR</th>
<th>&gt; 1</th>
</tr>
</thead>
</table>

**Arg. properties**

- **Contractible** wrt. VECTORS.
- **Suffix-contractible** wrt. VECTORS.vec (remove items from same position).

**Usage**

This constraint was motivated for breaking symmetry: more precisely when one wants to lexicographically order the consecutive columns of a matrix of decision variables. A further motivation is that using a set of lexicographic ordering constraints between two vectors does usually not allows to come up with a complete pruning.
Algorithm

A filtering algorithm achieving arc-consistency for a chain of lexicographical ordering constraints is presented in [90].

Six different ways of integrating a chain of lexicographical ordering constraints within non-overlapping constraints like diffn or geost and within their corresponding necessary condition like the cumulative constraint are shown in [2].

Systems

lexChainEq in Choco, lex_chain in SICStus.

See also

common keyword: allperm (lexicographic order), geost (symmetry, lexicographic ordering on the origins of tasks, rectangles, ...), lex_between, lex_greater, lex_greatereq, lex_less (lexicographic order).

implied by: lex2 (columns lex ordering imposed by constraint lex2 removed), lex_chain_less (non-strict order implied by strict order), ordered_atleast_nvector (NVEC of constraint ordered_atleast_nvector removed), ordered_atmost_nvector (NVEC of constraint ordered_atmost_nvector removed), ordered_nvector (NVEC of constraint ordered_nvector removed).

part of system of constraints: lex_lesseq.

related: cumulative, diffn (lexicographic ordering on the origins of tasks, rectangles, ...).

system of constraints: lex2.

used in graph description: lex_lesseq.

Keywords

characteristic of a constraint: vector.

constraint type: system of constraints, decomposition, order constraint.

filtering: arc-consistency.

heuristics: heuristics and lexicographical ordering.

symmetry: symmetry, matrix symmetry, lexicographic order.
Arc input(s) VECTORS
Arc generator $PATH \rightarrow \text{collection}(\text{vectors1}, \text{vectors2})$
Arc arity 2
Arc constraint(s) $\text{lex.lesseq}(\text{vectors1.vec}, \text{vectors2.vec})$
Graph property(ies) $\text{NARC} = |\text{VECTORS}| - 1$

Graph model Parts (A) and (B) of Figure 5.383 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold. The $\text{lex.chain.lesseq}$ constraint holds since all the arc constraints of the initial graph are satisfied.

![Graph diagram](image)

Figure 5.383: Initial and final graph of the $\text{lex.chain.lesseq}$ constraint

Signature Since we use the $PATH$ arc generator on the VECTORS collection the number of arcs of the initial graph is equal to $|\text{VECTORS}| - 1$. For this reason we can rewrite $\text{NARC} = |\text{VECTORS}| - 1$ to $\text{NARC} \geq |\text{VECTORS}| - 1$ and simplify $\text{NARC}$ to NARC.
5.207  \texttt{lex\_different}

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Used for defining \texttt{lex_alldifferent}.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>\texttt{lex_different}(\texttt{VECTOR1}, \texttt{VECTOR2})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>different, diff.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>\texttt{VECTOR1} : \texttt{collection(var-dvar)}</td>
<td>\texttt{VECTOR2} : \texttt{collection(var-dvar)}</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>\texttt{required}(\texttt{VECTOR1}.\texttt{var})</td>
<td>\texttt{required}(\texttt{VECTOR2}.\texttt{var})</td>
<td>| \texttt{VECTOR1}</td>
</tr>
<tr>
<td>Purpose</td>
<td>Vectors \texttt{VECTOR1} and \texttt{VECTOR2} differ in at least one component.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>[ \langle 5, 2, 7, 1 \rangle, \langle 5, 3, 7, 1 \rangle ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Typical</td>
<td>| \texttt{VECTOR1}</td>
<td>&gt; 1</td>
<td>\texttt{range}(\texttt{VECTOR1}.\texttt{var}) &gt; 1</td>
</tr>
<tr>
<td>Symmetries</td>
<td>- Arguments are \texttt{permutable} w.r.t. permutation \texttt{(VECTOR1, VECTOR2)}.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arg. properties</td>
<td>Extensible wrt. \texttt{VECTOR1} and \texttt{VECTOR2} \texttt{(add items at same position)}.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reformulation</td>
<td>The \texttt{lex_different} constraint can be expressed in term of the following disjunction of disequality constraints \texttt{U1} \neq \texttt{V1} \lor \texttt{U2} \neq \texttt{V2} \lor \ldots \lor \texttt{U</td>
<td>VECTOR1</td>
<td>} \neq \texttt{V</td>
</tr>
<tr>
<td>Used in</td>
<td>\texttt{lex_alldifferent, sort_permutation}.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>See also</td>
<td>common keyword: \texttt{lex_greatereq, lex_lesseq(vector)}.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>implied by: \texttt{disjoint, lex_greater, lex_less}.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>negation: \texttt{lex_equal}.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>system of constraints: \texttt{lex_alldifferent}.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Keywords

characteristic of a constraint: vector, disequality, automaton, automaton without counters, reified automaton constraint.

constraint network structure: Berge-acyclic constraint network.

filtering: arc-consistency.
Arc input(s) | VECTOR1 VECTOR2
---|---
Arc generator | \( PRODUCT(=) \rightarrow \text{collection}(\text{vector1}, \text{vector2}) \)
Arc arity | 2
Arc constraint(s) | vector1.var \( \neq \) vector2.var
Graph property(ies) | NARC \( \geq \) 1

Graph model

Parts (A) and (B) of Figure 5.384 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the unique arc of the final graph is stressed in bold. It corresponds to a component where the two vectors differ.

![Graph Diagram]

Figure 5.384: Initial and final graph of the lex_different constraint
Automaton

Figure 5.385 depicts the automaton associated with the lex\_different constraint. Let \( VAR_1 \) and \( VAR_2 \), respectively, be the \text{var} attributes of the \( i^{th} \) items of the VECTOR1 and the VECTOR2 collections. To each pair \((VAR_1, VAR_2)\) corresponds a 0-1 signature variable \( S_i \) as well as the following signature constraint: \( VAR_1 = VAR_2 \Leftrightarrow S_i \).

![Automaton Diagram]

Figure 5.385: Automaton of the lex\_different constraint

![Hypergraph Diagram]

Figure 5.386: Hypergraph of the reformulation corresponding to the automaton of the lex\_different constraint
5.208  lex_equal

Description Links Graph Automaton

Origin Initially introduced for defining nvector

Constraint \text{lex_equal}(\text{VECTOR1}, \text{VECTOR2})

Synonyms equal, eq.

Arguments \text{VECTOR1} : \text{collection(var−dvar)}\n\text{VECTOR2} : \text{collection(var−dvar)}

Restrictions \text{required(VECTOR1.var)}\n\text{required(VECTOR2.var)}\n|\text{VECTOR1}| = |\text{VECTOR2}|

Purpose \text{VECTOR1} is \textit{equal} to \text{VECTOR2}. Given two vectors, \( \vec{X} \) and \( \vec{Y} \) of \( n \) components, \( \langle X_0, \ldots, X_{n-1} \rangle \) and \( \langle Y_0, \ldots, Y_{n-1} \rangle \). \( \vec{X} \) is \textit{equal} to \( \vec{Y} \) if and only if \( n = 0 \) or \( X_0 = Y_0 \land X_1 = Y_1 \land \ldots \land X_{n-1} = Y_{n-1} \).

Example \( \begin{pmatrix} \langle 1,9,1,5 \rangle, \\
\langle 1,9,1,5 \rangle \end{pmatrix} \)

The \textit{lex_equal} constraint holds since (1) the first component of the first vector is equal to the first component of the second vector, (2) the second component of the first vector is equal to the second component of the second vector, (3) the third component of the first vector is equal to the third component of the second vector and (4) the fourth component of the first vector is equal to the fourth component of the second vector.

Typical \( |\text{VECTOR1}| > 1 \)
\( \text{range}(\text{VECTOR1.var}) > 1 \)
\( \text{range}(\text{VECTOR2.var}) > 1 \)

Symmetries • Arguments are \textit{permutable} w.r.t. permutation (VECTOR1, VECTOR2).

• Items of VECTOR1 and VECTOR2 are \textit{permutable} (same permutation used).

Arg. Properties Contractible wrt. VECTOR1 and VECTOR2 (remove items from same position).

Used in atleast_nvector, atmost_nvector, nvector, nvectors.

See also common keyword: nvector (vector).

implied by: vec_eq_tuple.

implies: lex_greatereq, lex_lessseq, same.

negation: lex_different.

specialisation: vec_eq_tuple (variable replaced by integer in second argument).
Keywords

- **characteristic of a constraint:** vector, automaton, automaton without counters, reified automaton constraint,
- **constraint network structure:** Berge-acyclic constraint network.
- **filtering:** arc-consistency.
- **final graph structure:** acyclic, bipartite, no loop.
Arc input(s)  VECTOR1, VECTOR2
Arc generator  \( \text{PRODUCT}(=) \rightarrow \text{collection}(\text{vector1}, \text{vector2}) \)
Arc arity  2
Arc constraint(s)  \( \text{vector1}.\text{var} = \text{vector2}.\text{var} \)
Graph property(ies)  \( \text{NARC} = |\text{VECTOR1}| \)
Graph class  • ACYCLIC
• BIPARTITE
• NO LOOP

Graph model  Parts (A) and (B) of Figure 5.387 respectively show the initial and final graphs associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

Figure 5.387: Initial and final graph of the lex_equal constraint
Automaton

Figure 5.388 depicts the automaton associated with the `lex_equal` constraint. Let \( \text{VAR1}_i \) and \( \text{VAR2}_i \) respectively be the var attributes of the \( i^{th} \) items of the VECTOR1 and the VECTOR2 collections. To each pair \((\text{VAR1}_i, \text{VAR2}_i)\) corresponds a signature variable \( S_i \) as well as the following signature constraint: \((\text{VAR1}_i \neq \text{VAR2}_i \leftrightarrow S_i = 0) \land (\text{VAR1}_i = \text{VAR2}_i \leftrightarrow S_i = 1)\).

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{Automaton.png}
\caption{Automaton of the `lex_equal` constraint}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{Hypergraph.png}
\caption{Hypergraph of the reformulation corresponding to the automaton of the `lex_equal` constraint}
\end{figure}
5.209  \textbf{lex\_greater}

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>CHIP</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>\texttt{lex_greater(VECTOR1,VECTOR2)}</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Synonyms</strong></td>
<td>\texttt{lex, lex_chain, rel_greater, gt.}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| **Arguments** | \begin{itemize}
| & \texttt{VECTOR1} : \texttt{collection(var\text{-}dvar)}
| & \texttt{VECTOR2} : \texttt{collection(var\text{-}dvar)}
| \end{itemize} | |
| **Restrictions** | \begin{itemize}
| & \texttt{required(VECTOR1, var)}
| & \texttt{required(VECTOR2, var)}
| & \texttt{|VECTOR1| = |VECTOR2|}
| \end{itemize} | |
| **Purpose** | \texttt{VECTOR1} is \textit{lexicographically strictly greater than} \texttt{VECTOR2}. Given two vectors, \( \vec{X} \) and \( \vec{Y} \) of \( n \) components, \( \langle X_0, \ldots, X_{n-1} \rangle \) and \( \langle Y_0, \ldots, Y_{n-1} \rangle \), \( \vec{X} \) is \textit{lexicographically strictly greater than} \( \vec{Y} \) if and only if \( X_0 > Y_0 \) or \( X_0 = Y_0 \) and \( \langle X_1, \ldots, X_{n-1} \rangle \) is \textit{lexicographically strictly greater than} \( \langle Y_1, \ldots, Y_{n-1} \rangle \). | |
| **Example** | \begin{itemize}
| & \( \langle 5, 2, 7, 1 \rangle, \langle 5, 2, 6, 2 \rangle \) |
| \end{itemize} | |
| The \textit{lex\_greater} constraint holds since \( \text{VECTOR1} = \langle 5, 2, 7, 1 \rangle \) is lexicographically strictly greater than \( \text{VECTOR2} = \langle 5, 2, 6, 2 \rangle \). | |
| **Typical** | \( |\text{VECTOR1}| > 1 \) | |
| **Symmetries** | \begin{itemize}
| & \text{VECTOR1}.\text{var} can be increased.
| & \text{VECTOR2}.\text{var} can be decreased.
| \end{itemize} | |
| **Arg. properties** | \textit{Suffix-extensible} wrt. \texttt{VECTOR1} and \texttt{VECTOR2} (\textit{add items at same position}). | |
| **Remark** | A \textit{multiset ordering} constraint and its corresponding filtering algorithm are described in [161]. | |
| **Algorithm** | The first filtering algorithm maintaining \textit{arc\-consistency} for this constraint was presented in [160]. A second filtering algorithm maintaining \textit{arc\-consistency} and detecting entailment in a more eager way, was given in [91]. This second algorithm was derived from a deterministic finite automata. A third filtering algorithm extending the algorithm presented in [160] detecting entailment is given in the PhD thesis of Z. Kızıltan [221, page 95]. The previous thesis [221, pages 105–109] presents also a filtering algorithm handling the fact that a given variable has more than one occurrence. Finally, T. Frühwirth shows how to encode lexicographic ordering constraints within the context of CHR [162] in [163]. | |
Reformulation

The following reformulations in terms of arithmetic and/or logical expressions exist for enforcing the lexicographically strictly greater than constraint. The first one converts $\vec{X}$ and $\vec{Y}$ into numbers and post an inequality constraint. It assumes all components of $\vec{X}$ and $\vec{Y}$ to be within $[0, a-1]$:

$$a^{n-1}Y_0 + a^{n-2}Y_1 + \ldots + a^0Y_{n-1} < a^{n-1}X_0 + a^{n-2}X_1 + \ldots + a^0X_{n-1}$$

Since the previous reformulation can only be used with small values of $n$ and $a$, W. Harvey came up with the following alternative model that maintains arc-consistency:

$$(Y_0 < X_0 + (Y_1 < X_1 + (\ldots + (Y_{n-1} < X_{n-1} + 0)\ldots))) = 1$$

Finally, the lexicographically strictly greater than constraint can be expressed as a conjunction or a disjunction of constraints:

$$Y_0 \leq X_0 \land (Y_0 = X_0) \Rightarrow Y_1 \leq X_1 \land (Y_0 = X_0 \land Y_1 = X_1) \Rightarrow Y_2 \leq X_2 \land \ldots$$

$$(Y_0 = X_0 \land Y_1 = X_1 \land \ldots \land Y_{n-2} = X_{n-2}) \Rightarrow Y_{n-1} < X_{n-1}$$

$$Y_0 < X_0 \lor Y_0 = X_0 \land Y_1 < X_1 \lor Y_0 = X_0 \land Y_1 = X_1 \land Y_2 < X_2 \lor \ldots$$

When used separately, the two previous logical decompositions do not maintain arc-consistency.

Systems

lex in Choco, rel in Gecode, lex_greater in MiniZinc, lex_chain in SICStus.

See also

common keyword: cond_lex_greater, lex_between, lex_chain_less, lex_chain_leqseq (lexicographic order).

implies: lex_different, lex_greatereq.

implies (if swap arguments): lex_less.

negation: lex_leqseq.

Keywords

characteristic of a constraint: vector, automaton, automaton without counters, reified automaton constraint, derived collection.

constraint network structure: Berge-acyclic constraint network.

constraint type: order constraint.

filtering: duplicated variables, arc-consistency.

heuristics: heuristics and lexicographical ordering.

symmetry: symmetry, matrix symmetry, lexicographic order, multiset ordering.
### Derived Collections

<table>
<thead>
<tr>
<th>Collection Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| \( \text{DESTINATION} \) | \begin{align*}
\text{col} ( & \text{collection}(\text{index-int, x-int, y-int}), \\
& \text{item}(\text{index} - 0, x - 0, y - 0) )
\end{align*} |
| \( \text{COMPONENTS} \) | \begin{align*}
\text{col} ( & \text{collection}(\text{index-int, x-dvar, y-dvar}), \\
& \text{item}(\text{index} - \text{VECTOR1.key}, x - \text{VECTOR1.var}, y - \text{VECTOR2.var}) )
\end{align*} |

### Arc input(s)

| Arc generator | \( \text{PRODUCT}(\text{PATH, VOID}) \mapsto \text{collection}(\text{item1, item2}) \) |

### Arc constraint(s)

\[
\bigvee \left( \begin{array}{l}
\text{item2.index} > 0 \land \text{item1.x} = \text{item1.y}, \\
\text{item2.index} = 0 \land \text{item1.x} > \text{item1.y}
\end{array} \right)
\]

### Graph property(ies)

\( \text{PATH_FROM_TO}(\text{index, 1, 0}) = 1 \)

### Graph model

Parts (A) and (B) of Figure 5.390 respectively show the initial and final graph associated with the Example slot. Since we use the \( \text{PATH_FROM_TO} \) graph property we show the following information on the final graph:

- The vertices, which respectively correspond to the start and the end of the required path, are stressed in bold.
- The arcs on the required path are also stressed in bold.

![Graph Model](image)

(A) (B)

Figure 5.390: Initial and final graph of the \textit{lex_greater} constraint

The vertices of the initial graph are generated in the following way:

- We create a vertex \( c_i \) for each pair of components that both have the same index \( i \).
- We create an additional dummy vertex called \( d \).

The arcs of the initial graph are generated in the following way:
• We create an arc between \( c_i \) and \( d \). We associate to this arc the arc constraint 
\[ \text{item}_1.x > \text{item}_2.y. \]

• We create an arc between \( c_i \) and \( c_{i+1} \). We associate to this arc the arc constraint 
\[ \text{item}_1.x = \text{item}_2.y. \]

The \textit{lex greater} constraint holds when there exist a path from \( c_1 \) to \( d \). This path can be interpreted as a sequence of \textit{equality} constraints on the prefix of both vectors, immediately followed by a \textit{greater than} constraint.

\textbf{Signature}

Since the maximum value returned by the graph property \texttt{PATH\_FROM\_TO} is equal to 1 we can rewrite \texttt{PATH\_FROM\_TO(index,1,0)} = 1 to \texttt{PATH\_FROM\_TO(index,1,0)} \( \geq \) 1. Therefore we simplify \texttt{PATH\_FROM\_TO} to \texttt{PATH\_FROM\_TO}.
Automaton

Figure 5.391 depicts the automaton associated with the \texttt{lex\_greater} constraint. Let \texttt{VAR1} and \texttt{VAR2} respectively be the \texttt{var} attributes of the \texttt{i}\textsuperscript{th} items of the \texttt{VECTOR1} and the \texttt{VECTOR2} collections. To each pair (\texttt{VAR1}\textsubscript{i}, \texttt{VAR2}\textsubscript{i}) corresponds a signature variable \texttt{S}\textsubscript{i} as well as the following signature constraint: (\texttt{VAR1}\textsubscript{i} < \texttt{VAR2}\textsubscript{i} \iff \texttt{S}\textsubscript{i} = 1) \land (\texttt{VAR1}\textsubscript{i} = \texttt{VAR2}\textsubscript{i} \iff \texttt{S}\textsubscript{i} = 2) \land (\texttt{VAR1}\textsubscript{i} > \texttt{VAR2}\textsubscript{i} \iff \texttt{S}\textsubscript{i} = 3).

Figure 5.391: Automaton of the \texttt{lex\_greater} constraint

Figure 5.392: Hypergraph of the reformulation corresponding to the automaton of the \texttt{lex\_greater} constraint
### 5.210 lex_greatereq

<table>
<thead>
<tr>
<th><strong>DESCRIPTION</strong></th>
<th><strong>LINKS</strong></th>
<th><strong>GRAPH</strong></th>
<th><strong>AUTOMATON</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>CHIP</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>lex_greatereq(VECTOR1, VECTOR2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Synonyms</strong></td>
<td>leq, lex_chain, rel_greatereq, geq, lex_geq</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>VECTOR1 : <code>collection(var−dvar)</code>&lt;br&gt;VECTOR2 : <code>collection(var−dvar)</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Restrictions**
- `required(VECTOR1, var)`
- `required(VECTOR2, var)`
- `|VECTOR1| = |VECTOR2|

**Purpose**
VECTOR1 is lexicographically greater than or equal to VECTOR2. Given two vectors, \( \vec{X} \) and \( \vec{Y} \) of \( n \) components, \( \langle X_0, \ldots, X_{n-1} \rangle \) and \( \langle Y_0, \ldots, Y_{n-1} \rangle \), \( \vec{X} \) is lexicographically greater than or equal to \( \vec{Y} \) if and only if \( n = 0 \) or \( X_0 > Y_0 \) or \( X_0 = Y_0 \) and \( \langle X_1, \ldots, X_{n-1} \rangle \) is lexicographically greater than or equal to \( \langle Y_1, \ldots, Y_{n-1} \rangle \).

**Example**

\[
\begin{align*}
\{ (5, 2, 8, 9), \\
(5, 2, 6, 2) \\
(5, 2, 3, 9), \\
(5, 2, 3, 9) \\
\}
\end{align*}
\]

The `lex_greatereq` constraints associated with the first and second examples hold since:

- Within the first example `VECTOR1 = (5, 2, 8, 9)` is lexicographically greater than or equal to `VECTOR2 = (5, 2, 6, 2).
- Within the second example `VECTOR1 = (5, 2, 3, 9)` is lexicographically greater than or equal to `VECTOR2 = (5, 2, 3, 9).

**Typical**

\[|VECTOR1| > 1\]

**Symmetries**
- `VECTOR1.var` can be increased.
- `VECTOR2.var` can be decreased.

**Arg. properties**

Suffix-contractible wrt. `VECTOR1` and `VECTOR2` (remove items from same position).

**Remark**

A multiset ordering constraint and its corresponding filtering algorithm are described in [161].
Algorithm

The first filtering algorithm maintaining arc-consistency for this constraint was presented in [160]. A second filtering algorithm maintaining arc-consistency and detecting entailment in a more eager way, was given in [91]. This second algorithm was derived from a deterministic finite automata. A third filtering algorithm extending the algorithm presented in [160] detecting entailment is given in the PhD thesis of Z. Kızıltan [221], page 95. The previous thesis [221], pages 105–109 presents also a filtering algorithm handling the fact that a given variable has more than one occurrence. Finally, T. Frühwirth shows how to encode lexicographic ordering constraints within the context of CHR [162] in [163].

Reformulation

The following reformulations in term of arithmetic and/or logical expressions exist for enforcing the lexicographically greater than or equal to constraint. The first one converts \( \vec{X} \) and \( \vec{Y} \) into numbers and post an inequality constraint. It assumes all components of \( \vec{X} \) and \( \vec{Y} \) to be within \([0, a-1]\):

\[
a^{n-1}Y_0 + a^{n-2}Y_1 + \ldots + a^0Y_{n-1} \leq a^{n-1}X_0 + a^{n-2}X_1 + \ldots + a^0X_{n-1}
\]

Since the previous reformulation can only be used with small values of \( n \) and \( a \), W. Harvey came up with the following alternative model that maintains arc-consistency:

\[
(Y_0 < X_0 + (Y_1 < X_1 + ( \ldots + (Y_{n-1} < X_{n-1} + 1) \ldots ))) = 1
\]

Finally, the lexicographically greater than or equal to constraint can be expressed as a conjunction or a disjunction of constraints:

\[
Y_0 \leq X_0 \land (Y_0 = X_0) \Rightarrow Y_1 \leq X_1 \land (Y_0 = X_0 \land Y_1 = X_1) \Rightarrow Y_2 \leq X_2 \land \ldots
\]

\[
Y_0 < X_0 \lor Y_0 = X_0 \land Y_1 < X_1 \lor Y_0 = X_0 \land Y_1 = X_1 \land Y_2 < X_2 \lor \ldots
\]

\[
Y_0 = X_0 \land Y_1 = X_1 \land \ldots \land Y_{n-2} = X_{n-2} \land Y_{n-1} \leq X_{n-1}
\]

When used separately, the two previous logical decompositions do not maintain arc-consistency.

Systems

\texttt{lexEq} in Choco, \texttt{rel} in Gecode, \texttt{lex.greatereq} in MiniZinc, \texttt{lex.chain} in SICStus.

See also

common keyword: \texttt{cond_lex.greatereq}, \texttt{lex_between}, \texttt{lex_chain_less}, \texttt{lex_chain_greatereq} (lexicographic order), \texttt{lex_different} (vector).

implied by: \texttt{lex_equal}, \texttt{lex_greater}, \texttt{sort}.

implies (if swap arguments): \texttt{lex_lesseq}.

negation: \texttt{lex_less}.

Keywords

characteristic of a constraint: vector, automaton, automaton without counters, reified automaton constraint, derived collection.

constraint network structure: Berge-acyclic constraint network.
constraint type: order constraint.
filtering: duplicated variables, arc-consistency.
heuristics: heuristics and lexicographical ordering.
symmetry: symmetry, matrix symmetry, lexicographic order, multiset ordering.
Derived Collections

\[
\begin{align*}
\text{col} & \left( \text{DESTINATION} \rightarrow \text{collection(} \text{index} - \text{int, } x - \text{int, } y - \text{int} ) \right) \\
\text{col} & \left( \text{COMPONENTS} \rightarrow \text{collection(} \text{index} - \text{int, } x - \text{dvar, } y - \text{dvar} ) \right)
\end{align*}
\]

Arc input(s)

**COMPONENTS DESTINATION**

Arc generator

\[PRODUCT(PATH,VOID) \leftrightarrow \text{collection(item1, item2)}\]

Arc arity

2

Arc constraint(s)

\[\bigvee \begin{align*}
\text{item2.index} & > 0 \land \text{item1.x} = \text{item1.y} \\
\text{item1.index} & < \lvert \text{VECTOR1} \rvert \land \text{item2.index} = 0 \land \text{item1.x} > \text{item1.y} \\
\text{item1.index} & = \lvert \text{VECTOR1} \rvert \land \text{item2.index} = 0 \land \text{item1.x} \geq \text{item1.y}
\end{align*}\]

Graph property(ies)

**PATH_FROM_TO(index,1,0) = 1**

Graph model

Parts (A) and (B) of Figure 5.393 respectively show the initial and final graph associated with the first example of the Example slot. Since we use the **PATH_FROM_TO** graph property we show on the final graph the following information:

- The vertices, which respectively correspond to the start and the end of the required path, are stressed in bold.
- The arcs on the required path are also stressed in bold.

Figure 5.393: Initial and final graph of the `lex_greatereq` constraint

The vertices of the initial graph are generated in the following way:

- We create a vertex \(c_i\) for each pair of components that both have the same index \(i\).
- We create an additional dummy vertex called \(d\).
The arcs of the initial graph are generated in the following way:

- We create an arc between \( c_i \) and \( d \). When \( c_i \) was generated from the last components of both vectors we associate to this arc the arc constraint \( \text{item}_1.x \geq \text{item}_2.y \); otherwise we associate to this arc the arc constraint \( \text{item}_1.x > \text{item}_2.y \).

- We create an arc between \( c_i \) and \( c_{i+1} \). We associate to this arc the arc constraint \( \text{item}_1.x = \text{item}_2.y \).

The \texttt{lex.greatereq} constraint holds when there exist a path from \( c_1 \) to \( d \). This path can be interpreted as a maximum sequence of \textit{equality} constraints on the prefix of both vectors, possibly followed by a \textit{greater than} constraint.

**Signature**

Since the maximum value returned by the graph property \texttt{PATH\_FROM\_TO} is equal to 1 we can rewrite \texttt{PATH\_FROM\_TO(index,1,0) = 1} to \texttt{PATH\_FROM\_TO(index,1,0) \geq 1}. Therefore we simplify \texttt{PATH\_FROM\_TO} to \texttt{PATH\_FROM\_TO}. 
Automaton

Figure 5.394 depicts the automaton associated with the \texttt{lex\_greatereq} constraint. Let \( \text{VAR1}_i \) and \( \text{VAR2}_i \) respectively be the var attributes of the \( i \text{'th} \) items of the \texttt{VECTOR1} and the \texttt{VECTOR2} collections. To each pair \((\text{VAR1}_i, \text{VAR2}_i)\) corresponds a signature variable \( S_i \) as well as the following signature constraint: \( (\text{VAR1}_i < \text{VAR2}_i \Leftrightarrow S_i = 1) \land (\text{VAR1}_i = \text{VAR2}_i \Leftrightarrow S_i = 2) \land (\text{VAR1}_i > \text{VAR2}_i \Leftrightarrow S_i = 3) \).

\[ \]

**Figure 5.394:** Automaton of the \texttt{lex\_greatereq} constraint

\[ \]

**Figure 5.395:** Hypergraph of the reformulation corresponding to the automaton of the \texttt{lex\_greatereq} constraint
5.211 lex_less

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHIP</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Constraint**

`lex_less(VECTOR1, VECTOR2)`

**Synonyms**

`lex`, `lex_chain`, `rel`, `less`.

**Arguments**

- `VECTOR1 : collection(var−dvar)`
- `VECTOR2 : collection(var−dvar)`

**Restrictions**

- `required(VECTOR1.var)`
- `required(VECTOR2.var)`
- `|VECTOR1| = |VECTOR2|`

**Purpose**

`VECTOR1` is lexicographically strictly less than `VECTOR2`. Given two vectors, $\vec{X}$ and $\vec{Y}$ of $n$ components, $\langle X_0, \ldots, X_{n-1} \rangle$ and $\langle Y_0, \ldots, Y_{n-1} \rangle$, $\vec{X}$ is lexicographically strictly less than $\vec{Y}$ if and only if $X_0 < Y_0$ or $X_0 = Y_0$ and $\langle X_1, \ldots, X_{n-1} \rangle$ is lexicographically strictly less than $\langle Y_1, \ldots, Y_{n-1} \rangle$.

**Example**

```
\begin{pmatrix}
\langle 5, 2, 3, 9 \rangle, \\
\langle 5, 2, 6, 2 \rangle
\end{pmatrix}
```

The `lex_less` constraint holds since `VECTOR1 = \langle 5, 2, 3, 9 \rangle` is lexicographically strictly less than `VECTOR2 = \langle 5, 2, 6, 2 \rangle`.

**Symmetries**

- `VECTOR1.var` can be decreased.
- `VECTOR2.var` can be increased.

**Arg. properties**

Suffix-extensible wrt. `VECTOR1` and `VECTOR2` (add items at same position).

**Remark**

A multiset ordering constraint and its corresponding filtering algorithm are described in [161].

**Algorithm**

The first filtering algorithm maintaining arc-consistency for this constraint was presented in [160]. A second filtering algorithm maintaining arc-consistency and detecting entailment in a more eager way, was given in [91]. This second algorithm was derived from a deterministic finite automata. A third filtering algorithm extending the algorithm presented in [160] detecting entailment is given in the PhD thesis of Z. Kızıltan [221, page 95]. The previous thesis [221], pages 105–109] presents also a filtering algorithm handling the fact that a given variable has more than one occurrence. Finally, T. Frühwirth shows how to encode lexicographic ordering constraints within the context of CHR [162] in [163].
Reformulation

The following reformulations in term of arithmetic and/or logical expressions exist for enforcing the **lexicographically strictly less than** constraint. The first one converts $\vec{X}$ and $\vec{Y}$ into numbers and post an inequality constraint. It assumes all components of $\vec{X}$ and $\vec{Y}$ to be within $[0, a - 1]$:

$$a^{n-1}X_0 + a^{n-2}X_1 + \ldots + a^0X_{n-1} < a^{n-1}Y_0 + a^{n-2}Y_1 + \ldots + a^0Y_{n-1}$$

Since the previous reformulation can only be used with small values of $n$ and $a$, W. Harvey came up with the following alternative model that maintains arc-consistency:

$$(X_0 < Y_0 + (X_1 < Y_1 + (\ldots + (X_{n-1} < Y_{n-1} + 0)\ldots)))) = 1$$

Finally, the **lexicographically strictly less than** constraint can be expressed as a conjunction or a disjunction of constraints:

$$X_0 \leq Y_0 \land (X_0 = Y_0) \Rightarrow X_1 \leq Y_1 \land (X_0 = Y_0 \land X_1 = Y_1) \Rightarrow X_2 \leq Y_2 \land \ldots (X_0 = Y_0 \land X_1 = Y_1 \land \ldots \land X_{n-2} = Y_{n-2}) \Rightarrow X_{n-1} < Y_{n-1}$$

$$X_0 < Y_0 \lor X_0 = Y_0 \land X_1 < Y_1 \lor X_0 = Y_0 \land X_1 = Y_1 \land X_2 < Y_2 \lor \ldots$$

$$X_0 = Y_0 \land X_1 = Y_1 \land \ldots \land X_{n-2} = Y_{n-2} \land X_{n-1} < Y_{n-1}$$

When used separately, the two previous logical decompositions do not maintain arc-consistency.

**Systems**

- **lex** in Choco, **rel** in Gecode, **lex_less** in MiniZinc, **lex_chain** in SICStus.

**Used in**

- **lex_chain_less**, **ordered_atleast_nvector**, **ordered_atmost_nvector**, **ordered_nvector**.

**See also**

- common keyword: **cond_lex_less**, **lex_between**, **lex_chain_lesseq** (**lexicographic order**).
- implies: **lex_different**, **lex_lesseq**.
- implies (if swap arguments): **lex_greater**.
- negation: **lex_greatereq**.
- system of constraints: **lex_chain_less**.

**Keywords**

- characteristic of a constraint: **vector**, **automaton**, **automaton without counters**, **reified automaton constraint**, **derived collection**.
- constraint network structure: **Berge-acyclic constraint network**.
- constraint type: **order constraint**.
- filtering: **duplicated variables**, **arc-consistency**.
- heuristics: **heuristics and lexicographical ordering**.
- symmetry: **symmetry**, **matrix symmetry**, **lexicographic order**, **multiset ordering**.
Derived Collections

\[
\begin{align*}
\text{col} ( & \text{DESTINATION}\text{-}\text{collection}(\text{index}\text{-}\text{int}, x\text{-}\text{int}, y\text{-}\text{int}),) \\
& \text{col} ( \text{COMPONENTS}\text{-}\text{collection}(\text{index}\text{-}\text{int}, x\text{-}\text{dvar}, y\text{-}\text{dvar}), \\
& \quad \text{item}(\text{index} - \text{VECTOR1}.\text{key}, x - \text{VECTOR1}.\text{var}, y - \text{VECTOR2}.\text{var}) )
\end{align*}
\]

Arc input(s)

\text{COMPONENTS DESTINATION}

Arc generator

\text{PRODUCT}(\text{PATH}, \text{VOID}) \mapsto \text{collection}(\text{item1}, \text{item2})

Arc arity

2

Arc constraint(s)

\[ \lor \left( \begin{array}{c}
\text{item2}.\text{index} > 0 \land \text{item1}.x = \text{item1}.y, \\
\text{item2}.\text{index} = 0 \land \text{item1}.x < \text{item1}.y
\end{array} \right) \]

Graph property(ies)

\text{PATH\_FROM\_TO}(\text{index}, 1, 0) = 1

Graph model

Parts (A) and (B) of Figure 5.396 respectively show the initial and final graph associated with the Example slot. Since we use the \text{PATH\_FROM\_TO} graph property we show on the final graph the following information:

- The vertices, which respectively correspond to the start and the end of the required path, are stressed in bold.
- The arcs on the required path are also stressed in bold.

![Graph model](image)

Figure 5.396: Initial and final graph of the \text{lex\_less} constraint

The vertices of the initial graph are generated in the following way:

- We create a vertex \( c_i \) for each pair of components that both have the same index \( i \).
- We create an additional dummy vertex called \( d \).

The arcs of the initial graph are generated in the following way:
• We create an arc between $c_i$ and $d$. We associate to this arc the arc constraint $item_1.x < item_2.y$.

• We create an arc between $c_i$ and $c_{i+1}$. We associate to this arc the arc constraint $item_1.x = item_2.y$.

The $\text{lex}_{\text{less}}$ constraint holds when there exist a path from $c_i$ to $d$. This path can be interpreted as a sequence of equality constraints on the prefix of both vectors, immediately followed by a less than constraint.

**Signature**

Since the maximum value returned by the graph property $\text{PATH}_{\text{FROM}}_{\text{TO}}$ is equal to 1 we can rewrite $\text{PATH}_{\text{FROM}}_{\text{TO}}(\text{index}, 1, 0) = 1$ to $\text{PATH}_{\text{FROM}}_{\text{TO}}(\text{index}, 1, 0) \geq 1$. Therefore we simplify $\text{PATH}_{\text{FROM}}_{\text{TO}}$ to $\text{PATH}_{\text{FROM}}_{\text{TO}}$. 
Figure 5.397 depicts the automaton associated with the lex_less constraint. Let \text{VAR1}_i and \text{VAR2}_i respectively be the \text{var} attributes of the \text{i}^{th} items of the \text{VECTOR1} and the \text{VECTOR2} collections. To each pair (\text{VAR1}_i, \text{VAR2}_i) corresponds a signature variable \text{S}_i as well as the following signature constraint: (\text{VAR1}_i < \text{VAR2}_i \Leftrightarrow \text{S}_i = 1) \land (\text{VAR1}_i = \text{VAR2}_i \Leftrightarrow \text{S}_i = 2) \land (\text{VAR1}_i > \text{VAR2}_i \Leftrightarrow \text{S}_i = 3).
### 5.212 lex_lesseq

<table>
<thead>
<tr>
<th>DESCRIPITION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>CHIP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>lex_lesseq(VECTOR1, VECTOR2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>leseq, lex_chain, rel, lesseq, leq, lex_leq.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VECTOR1 : collection(var−dvar)</td>
<td>VECTOR2 : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(VECTOR1,var)</td>
<td>required(VECTOR2,var)</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>VECTORS is lexicographically less than or equal to VECTOR2. Given two vectors, ( \vec{X} ) and ( \vec{Y} ) of ( n ) components, ( (X_0, \ldots, X_{n-1}) ) and ( (Y_0, \ldots, Y_{n-1}) ), ( \vec{X} ) is lexicographically less than or equal to ( \vec{Y} ) if and only if ( n = 0 ) or ( X_0 &lt; Y_0 ) or ( X_0 = Y_0 ) and ( (X_1, \ldots, X_{n-1}) ) is lexicographically less than or equal to ( (Y_1, \ldots, Y_{n-1}) ).</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Example      | \[
\begin{pmatrix}
(5, 2, 3, 1), \\
(5, 2, 6, 2) \\
(5, 2, 3, 9), \\
(5, 2, 3, 9)
\end{pmatrix}
\] | | |
| Typical      | | | | | | | |
| Symmetries   | ● VECTOR1.var can be decreased. | ● VECTOR2.var can be increased. | |
| Arg. properties | Suffix-contractible wrt. VECTOR1 and VECTOR2 (remove items from same position). | |
| Remark       | A multiset ordering constraint and its corresponding filtering algorithm are described in [161]. | | |
**Algorithm**

The first filtering algorithm maintaining arc-consistency for this constraint was presented in [160]. A second filtering algorithm maintaining arc-consistency and detecting entailment in a more eager way, was given in [91]. This second algorithm was derived from a deterministic finite automata. A third filtering algorithm extending the algorithm presented in [160] detecting entailment is given in the PhD thesis of Z. Kızıltan [221, page 95]. The previous thesis [221, pages 105–109] presents also a filtering algorithm handling the fact that a given variable has more than one occurrence. Finally, T. Frühwirth shows how to encode lexicographic ordering constraints within the context of CHR [162] in [163].

**Reformulation**

The following reformulations in term of arithmetic and/or logical expressions exist for enforcing the lexicographically less than or equal to constraint. The first one converts $\vec{X}$ and $\vec{Y}$ into numbers and post an inequality constraint. It assumes all components of $\vec{X}$ and $\vec{Y}$ to be within $[0, a−1]$:

$$a^{n−1}X_0 + a^{n−2}X_1 + \ldots + a^0X_{n−1} \leq a^{n−1}Y_0 + a^{n−2}Y_1 + \ldots + a^0Y_{n−1}$$

Since the previous reformulation can only be used with small values of $n$ and $a$, W. Harvey came up with the following alternative model that maintains arc-consistency:

$$(X_0 < Y_0 + (X_1 < Y_1 + (\ldots + (X_{n−1} < Y_{n−1} + 1)\ldots))) = 1$$

Finally, the lexicographically less than or equal to constraint can be expressed as a conjunction or a disjunction of constraints:

$$X_0 \leq Y_0 \land (X_0 = Y_0 \Rightarrow X_1 \leq Y_1 \land (X_0 = Y_0 \land X_1 = Y_1) \Rightarrow X_2 \leq Y_2 \land \ldots$$

$$(X_0 = Y_0 \land X_1 = Y_1 \land \ldots \land X_{n−2} = Y_{n−2}) \Rightarrow X_{n−1} \leq Y_{n−1}$$

$$X_0 < Y_0 \lor X_0 = Y_0 \land X_1 < Y_1 \lor X_0 = Y_0 \land X_1 = Y_1 \land X_2 < Y_2 \lor \ldots$$

$$X_0 = Y_0 \land X_1 = Y_1 \land \ldots \land X_{n−2} = Y_{n−2} \land X_{n−1} \leq Y_{n−1}$$

When used separately, the two previous logical decompositions do not maintain arc-consistency.

**Systems**

- lexEq in Choco, rel in Gecode, lex.lesseq in MiniZinc, lex_chain in SICStus.

**Used in**

- lex_between, lex_chain_lesseq, ordered_atleast_nvector, ordered_atmost_nvector.

**See also**

- common keyword: allperm, cond_lex_lesseq(lexicographic order), lex2(matrix symmetry,lexicographic order), lex_chain_less(lexicographic order), lex_different(vector), strict_lex2(matrix symmetry,lexicographic order).

- implied by: lex_equal, lex_less, lex_lesseq_allperm.

- implies (if swap arguments): lex_greatereq.

- negation: lex_greater.

- system of constraints: lex_between, lex_chain_lesseq.
Keywords

**characteristic of a constraint:** vector, automaton, automaton without counters, reified automaton constraint, derived collection.

**constraint network structure:** Berge-acyclic constraint network.

**constraint type:** order constraint.

**filtering:** duplicated variables, arc-consistency.

**heuristics:** heuristics and lexicographical ordering.

**symmetry:** symmetry, matrix symmetry, lexicographic order, multiset ordering.
Derived Collections

Arc input(s) COMPONENTS DESTINATION
Arc generator $PRODUCT(PATH,VOID) \rightarrow \text{collection}(item1,item2)$
Arc arity 2
Arc constraint(s) $\bigvee \left( \begin{array}{l}
\text{item2.index} > 0 \land \text{item1.x} = \text{item1.y} \\
\text{item1.index} < |VECTOR1| \land \text{item2.index} = 0 \land \text{item1.x} < \text{item1.y} \\
\text{item1.index} = |VECTOR1| \land \text{item2.index} = 0 \land \text{item1.x} \leq \text{item1.y}
\end{array} \right)$
Graph property(ies) $PATH\_FROM\_TO(index,1,0) = 1$

Graph model

Parts (A) and (B) of Figure 5.399 respectively show the initial and final graph associated with the first example of the Example slot. Since we use the $PATH\_FROM\_TO$ graph property we show on the final graph the following information:

- The vertices, which respectively correspond to the start and the end of the required path, are stressed in bold.
- The arcs on the required path are also stressed in bold.

Figure 5.399: Initial and final graph of the $\text{lex.lesseq}$ constraint

The vertices of the initial graph are generated in the following way:

- We create a vertex $c_i$ for each pair of components that both have the same index $i$.
- We create an additional dummy vertex called $d$.
The arcs of the initial graph are generated in the following way:

- We create an arc between \( c_i \) and \( d \). When \( c_i \) was generated from the last components of both vectors we associate to this arc the arc constraint \( \text{item}_1.x \leq \text{item}_2.y \); otherwise we associate to this arc the arc constraint \( \text{item}_1.x < \text{item}_2.y \).
- We create an arc between \( c_i \) and \( c_{i+1} \). We associate to this arc the arc constraint \( \text{item}_1.x = \text{item}_2.y \).

The \text{lex}_{\text{lesseq}} \text{constraint holds when there exist a path from } c_1 \text{ to } d. \text{ This path can be interpreted as a maximum sequence of } \text{equality} \text{ constraints on the prefix of both vectors, possibly followed by a } \text{less than} \text{ constraint.}

**Signature**

Since the maximum value returned by the graph property \text{PATH\_FROM\_TO} is equal to 1 we can rewrite \text{PATH\_FROM\_TO(index,1,0)} = 1 to \text{PATH\_FROM\_TO(index,1,0)} \geq 1. Therefore we simplify \text{PATH\_FROM\_TO} to \text{PATH\_FROM\_TO}. 
Automaton

Figure 5.400 depicts the automaton associated with the `lex_lesseq` constraint. Let \( \text{VAR}1_i \) and \( \text{VAR}2_i \) respectively be the var attributes of the \( i^{th} \) items of the VECTOR1 and the VECTOR2 collections. To each pair \((\text{VAR}1_i, \text{VAR}2_i)\) corresponds a signature variable \( S_i \) as well as the following signature constraint: 

\[
(\text{VAR}1_i < \text{VAR}2_i \iff S_i = 1) \land (\text{VAR}1_i = \text{VAR}2_i \iff S_i = 2) \land (\text{VAR}1_i > \text{VAR}2_i \iff S_i = 3).
\]

![Automaton of the `lex_lesseq` constraint](image)

**Figure 5.400: Automaton of the `lex_lesseq` constraint**

Figure 5.401: Hypergraph of the reformulation corresponding to the automaton of the `lex_lesseq` constraint

![Hypergraph](image)
5.213 lex_lesseq_allperm

**DESCRIPTION**

**Origin**
Inspired by [155]

**Constraint**
lex_lesseq_allperm(VECTOR1, VECTOR2)

**Synonym**
leximin.

**Arguments**
VECTOR1 : collection(var−dvar)
VECTOR2 : collection(var−dvar)

**Restrictions**
required(VECTOR1, var)
required(VECTOR2, var)
|VECTOR1| = |VECTOR2|

**Purpose**
VECTOR1 is lexicographically less than or equal to all permutations of VECTOR2. Given two vectors, \( \vec{X} \) and \( \vec{Y} \) of \( n \) components, \( \langle X_0, \ldots, X_{n-1} \rangle \) and \( \langle Y_0, \ldots, Y_{n-1} \rangle \). \( \vec{X} \) is lexicographically less than or equal to \( \vec{Y} \) if and only if \( n = 0 \) or \( X_0 < Y_0 \) or \( X_0 = Y_0 \) and \( \langle X_1, \ldots, X_{n-1} \rangle \) is lexicographically less than or equal to \( \langle Y_1, \ldots, Y_{n-1} \rangle \).

**Example**

\[
\begin{pmatrix}
\langle 1, 2, 3 \rangle,
\langle 3, 1, 2 \rangle
\end{pmatrix}
\]

The lex_lesseq_allperm constraint holds since vector \( \langle 1, 2, 3 \rangle \) is lexicographically less than or equal to all the permutations of vector \( \langle 3, 1, 2 \rangle \) (i.e., \( \langle 1, 2, 3 \rangle, \langle 1, 3, 2 \rangle, \langle 2, 1, 3 \rangle, \langle 2, 3, 1 \rangle, \langle 3, 1, 2 \rangle, \langle 3, 2, 1 \rangle \)).

**Typical**

|VECTOR1| > 1

**Symmetry**
All occurrences of two distinct values in VECTOR1.var or VECTOR2.var can be **swapped**; all occurrences of a value in VECTOR1.var or VECTOR2.var can be **renamed** to any unused value.

**Arg. properties**
Suffix-contractible wrt. VECTOR1 and VECTOR2 (remove items from same position).

**Remark**
The lex_lesseq_allperm(VECTOR1, VECTOR2) can be reformulated as the conjunction sort(VECTOR2, VECTOR), lex_lesseq(VECTOR1, VECTOR).

**Systems**
leximin in Choco.

**Used in**
allperm.

**See also**
common keyword: allperm (matrix symmetry, lexicographic order).
implies: lex_lesseq.
system of constraints: allperm.
Keywords

characteristic of a constraint: vector.
constraint type: predefined constraint, order constraint.
symmetry: symmetry, matrix symmetry, lexicographic order.
5.214  link_set_to_booleans

**DESCRIPTION**

Inspired by domain_constraint.

**LINKS**

link_set_to_booleans(SVAR, BOOLEANS)

**GRAPH**

- SVAR : svar
- BOOLEANS : collection(bool-dvar, val-int)

**Purpose**

Make the link between a set variable SVAR and those 0-1 variables that are associated with each potential value belonging to SVAR. The 0-1 variables, which are associated with a value belonging to the set variable SVAR, are equal to 1, while the remaining 0-1 variables are all equal to 0.

**Example**

\[
\{1, 3, 4\},
\begin{pmatrix}
\text{bool} & \text{val} \\
0 & 0 \\
1 & 1 \\
0 & 2 \\
1 & 3 \\
1 & 4 \\
0 & 5
\end{pmatrix}
\]

In the example, the 0-1 variables associated with the values 1, 3 and 4 are all set to 1, while the other 0-1 variables are set to 0. Consequently, the link_set_to_booleans constraint holds since its first argument SVAR is set to \(\{1, 3, 4\}\).

**Typical**

- \(|\text{BOOLEANS}| > 1\)
- \(\text{range}(|\text{BOOLEANS.bool}|) > 1\)

**Symmetry**

Items of BOOLEANS are permutable.

**Usage**

This constraint is used in order to make the link between a formulation using set variables and a formulation based on linear programming.

**Systems**

- channel in Gecode, link_set_to_booleans in MiniZinc

**See also**

- common keyword: alldifferent_between_sets, clique (constraint involving set variables), domain_constraint (channelling constraint), k_cut, path_from_to, roots, strongly_connected, symmetric_cardinality, symmetric_gcc_tour (constraint involving set variables).
Keywords

**characteristic of a constraint:** derived collection.

**constraint arguments:** constraint involving set variables.

**constraint type:** decomposition, value constraint.

**filtering:** linear programming.

**modelling:** channelling constraint, set channel.
Derived Collection

\[
\text{col}(\text{SET−collection(one−int,setvar−svar)},
\text{[item(one−1,setvar−SVAR)]})
\]

Arc input(s) 
- SET BOOLEANS

Arc generator 
- \(PRODUCT\) \(\rightarrow\) \(collection\)(set, booleans)

Arc arity 
- 2

Arc constraint(s) 
- booleans.bool = set.one \(\leftrightarrow\) \(\text{in}\_set\)(booleans.val, set.setvar)

Graph property(ies) 
- \(NARC = |\text{BOOLEANS}|\)

Graph model 
- The link set_to_booleans constraint is modelled with the following bipartite graph. The first set of vertices corresponds to one single vertex containing the set variable. The second class of vertices contains one vertex for each item of the collection BOOLEANS. The arc constraint between the set variable SVAR and one potential value \(v\) of the set variable expresses the following:
  - If the 0-1 variable associated with \(v\) is equal to 1 then \(v\) should belong to SVAR.
  - Otherwise if the 0-1 variable associated with \(v\) is equal to 0 then \(v\) should not belong to SVAR.

Since all arc constraints should hold the final graph contains exactly \(|\text{BOOLEANS}|\) arcs.

Parts (A) and (B) of Figure 5.402 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold. The link_set_to_booleans constraint holds since the final graph contains exactly 6 arcs (one for each 0-1 variable).

Signature 
- Since the initial graph contains \(|\text{BOOLEANS}|\) arcs the maximum number of arcs of the final graph is equal to \(|\text{BOOLEANS}|\). Therefore we can rewrite the graph property \(NARC = |\text{BOOLEANS}|\) to \(NARC \geq |\text{BOOLEANS}|\) and simplify NARC to NARC.
Figure 5.402: Initial and final graph of the `link_set_to_booleans` constraint
5.215 longest_change

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from change.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>longest_change(SIZE, VARIABLES, CTR)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>SIZE : dvar</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES : collection(var−dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CTR : atom</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>SIZE ≥ 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SIZE &lt;</td>
<td>VARIABLES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES, var)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CTR ∈ [\text{\texttt{=},\texttt{\neq},\texttt{&lt;},\texttt{\geq},\texttt{&gt;},\texttt{\leq}}]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>SIZE is the maximum number of consecutive variables of the collection VARIABLES for which constraint CTR holds in an uninterrupted way. We count a change when X CTR Y holds; X and Y are two consecutive variables of the collection VARIABLES.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example

\[
\begin{pmatrix}
\text{var} - 8, \\
\text{var} - 8, \\
\text{var} - 3, \\
\text{var} - 4, \\
\text{var} - 1, \\
\text{var} - 5, \\
\text{var} - 5, \\
\text{var} - 2
\end{pmatrix}, \neq
\]

The longest_change constraint holds since its first argument SIZE = 4 is fixed to the length of the longest subsequence of consecutive values of the collection \(8, 8, 3, 4, 1, 1, 5, 5, 2\) such that two consecutive values are distinct (i.e., subsequence 8 3 4 1).

Typical

<table>
<thead>
<tr>
<th>VARIABLES</th>
<th>&gt; 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>range(VARIABLES, var)</td>
<td>&gt; 1</td>
</tr>
<tr>
<td>CTR ∈ [\text{\texttt{\neq}}]</td>
<td></td>
</tr>
</tbody>
</table>

Symmetry

One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

Functional dependency: SIZE determined by VARIABLES and CTR.

See also

root concept: change.
Keywords

characteristic of a constraint: automaton, automaton with counters.
constraint arguments: pure functional dependency.
constraint network structure: sliding cyclic(1) constraint network(3).
constraint type: timetabling constraint.
miscellaneous: obscure.
modelling: functional dependency.
Arc input(s)  VARIABLES
Arc generator  \( \text{PATH} \rightarrow \text{collection}(\text{variables}_1, \text{variables}_2) \)
Arc arity  2
Arc constraint(s)  \( \text{variables}_1.\text{var} \; \text{CTR} \; \text{variables}_2.\text{var} \)
Graph property(ies)  \( \text{MAX}_NCC = \text{SIZE} \)

**Graph model**

In order to specify the \text{longest_change} constraint, we use \text{MAX}_NCC, which is the number of vertices of the largest connected component. Since the initial graph corresponds to a path, this will be the length of the longest path in the final graph.

Parts (A) and (B) of Figure 5.403 respectively show the initial and final graph associated with the \text{Example} slot. Since we use the \text{MAX}_NCC graph property we show the largest connected component of the final graph. It corresponds to the longest period of uninterrupted changes: sequence 8, 3, 4, 1 that involves 4 consecutive variables.

![Figure 5.403](image.png)

**Figure 5.403**: Initial and final graph of the \text{longest_change} constraint
Automaton

Figure 5.404 depicts the automaton associated with the longest_change constraint. To each pair of consecutive variables \((\text{VAR}_i, \text{VAR}_{i+1})\) of the collection VARIABLES corresponds a 0-1 signature variable \(S_i\). The following signature constraint links \(\text{VAR}_i\), \(\text{VAR}_{i+1}\) and \(S_i\):

\[
\text{VAR}_i \text{ CTR } \text{VAR}_{i+1} \iff S_i.
\]

\[
\{C=0, D=1\} \quad \{C=\max(C, D), D=1\} \quad \{C=\max(C, D)\}
\]

Figure 5.404: Automaton of the longest_change constraint

Figure 5.405: Hypergraph of the reformulation corresponding to the automaton of the longest_change constraint
5.216 lt

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td></td>
</tr>
<tr>
<td>Arithmetic.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td></td>
</tr>
<tr>
<td>lt(VAR1, VAR2)</td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td></td>
</tr>
<tr>
<td>rel, xlty.</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td></td>
</tr>
<tr>
<td>VAR1 : dvar</td>
<td></td>
</tr>
<tr>
<td>VAR2 : dvar</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td></td>
</tr>
<tr>
<td>Enforce the fact that the first variable is strictly less than the second variable.</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td></td>
</tr>
<tr>
<td>(1, 8)</td>
<td></td>
</tr>
<tr>
<td>The lt constraint holds since 1 is strictly less than 8.</td>
<td></td>
</tr>
<tr>
<td>Symmetries</td>
<td></td>
</tr>
<tr>
<td>• VAR1 can be replaced by any value &lt; VAR2.</td>
<td></td>
</tr>
<tr>
<td>• VAR2 can be replaced by any value &gt; VAR1.</td>
<td></td>
</tr>
<tr>
<td>Systems</td>
<td></td>
</tr>
<tr>
<td>lt in Choco, rel in Gecode, xlty in JaCoP, #&lt; in SICStus.</td>
<td></td>
</tr>
<tr>
<td>See also</td>
<td></td>
</tr>
<tr>
<td>common keyword: eq(binary constraint,arithmetic constraint).</td>
<td></td>
</tr>
<tr>
<td>implies: leq, neq.</td>
<td></td>
</tr>
<tr>
<td>implies (if swap arguments): gt.</td>
<td></td>
</tr>
<tr>
<td>negation: geq.</td>
<td></td>
</tr>
<tr>
<td>Keywords</td>
<td></td>
</tr>
<tr>
<td>constraint arguments: binary constraint.</td>
<td></td>
</tr>
<tr>
<td>constraint type: predefined constraint, arithmetic constraint.</td>
<td></td>
</tr>
<tr>
<td>filtering: arc-consistency.</td>
<td></td>
</tr>
</tbody>
</table>
### 5.217 map

<table>
<thead>
<tr>
<th>Description</th>
<th>Links</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Inspired by [355]</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>$\text{map(NBCycle, NTree, Nodes)}$</td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>$\text{NBCycle : dvar}$</td>
<td>$\text{NTree : dvar}$</td>
</tr>
<tr>
<td></td>
<td>$\text{Nodes : collection(index-int, succ-dvar)}$</td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>$\text{NBCycle} \geq 0$</td>
<td>$\text{NTree} \geq 0$</td>
</tr>
<tr>
<td></td>
<td>$\text{required}($Nodes,$[\text{index}, \text{succ}])$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{Nodes.index} \geq 1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{Nodes.index} \leq</td>
<td>\text{Nodes}</td>
</tr>
<tr>
<td></td>
<td>$\text{distinct}($Nodes,$\text{index})$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{Nodes.succ} \geq 1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{Nodes.succ} \leq</td>
<td>\text{Nodes}</td>
</tr>
</tbody>
</table>

**Purpose**

“Every map decomposes into a set of connected components, also called connected maps. Each component consists of the set of all points that wind up on the same cycle, with each point on the cycle attached to a tree of all points that enter the cycle at that point.”

**Example**

$\begin{pmatrix}
\text{index - 1 succ - 5,} \\
\text{index - 2 succ - 9,} \\
\text{index - 3 succ - 8,} \\
\text{index - 4 succ - 2,} \\
\text{index - 5 succ - 9,} \\
\text{index - 6 succ - 2,} \\
\text{index - 7 succ - 9,} \\
\text{index - 8 succ - 8,} \\
\text{index - 9 succ - 1}
\end{pmatrix}$

The map constraint holds since, as shown by part (B) of Figure 5.406, the graph corresponding to the Nodes collection is a map containing NBCycle = 2 cycles (i.e., a first cycle involving vertices 1, 5 and 9 and a second cycle involving vertex 8) and 3 trees (i.e., two trees respectively involving vertices 7 and 4, 6, 2 and attached to the first cycle, and one tree mentioning vertex 3 linked to the second cycle.)

**Typical**

- NBCycle > 0
- NTree > 0
- NBCycle < |Nodes|
- NBCycle < NTree
- |Nodes| > 2
Symmetry

Items of NODES are permutable.

Arg. properties

- Functional dependency: NBCYCLE determined by NODES.
- Functional dependency: NBTREE determined by NODES.

See also

common keyword: cycle, graph_crossing, tree (graph partitioning constraint).

Keywords

constraint arguments: pure functional dependency.
constraint type: graph constraint, graph partitioning constraint.
final graph structure: connected component.
modelling: functional dependency.
Arc input(s)  NODES
Arc generator  \textit{CLIQUE}\rightarrow\textit{collection}(\textit{nodes}_1,\textit{nodes}_2)
Arc arity  2
Arc constraint(s)  \textit{nodes}_1.\text{succ} = \textit{nodes}_2.\text{index}
Graph property(ies)  
  \begin{itemize}
    \item NCC\,=\,NBCYCLE
    \item NTREE\,=\,NBTREE
  \end{itemize}

Graph model
Note that, for the argument NBTREE of the \texttt{map} constraint, we consider a definition different from the one used for the argument NTREES of the \texttt{tree} constraint:

- In the \texttt{map} constraint the number of trees NBTREE is equal to the number of vertices of the final graph, which both do not belong to any circuit and have a successor that is located on a circuit. Therefore we count three trees in the context of the \texttt{Example} slot.
- In the \texttt{tree} constraint the number of trees NTREES is equal to the number of connected components of the final graph.

Parts (A) and (B) of Figure 5.406 respectively show the initial and final graph associated with the \texttt{Example} slot. Since we use the NCC graph property, we display the two connected components of the final graph. Each of them corresponds to a connected map. The first connected map is made up from one circuit and two trees, while the second one consists of one circuit and one tree. Since we also use the NBTREE graph property, we display with a double circle those vertices that do not belong to any circuit but for which at least one successor belongs to a circuit.

![Graph](image)

Figure 5.406: Initial and final graph of the \texttt{map} constraint
### max_index

<table>
<thead>
<tr>
<th>Description</th>
<th>Links</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>N. Beldiceanu</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td><code>max_index(MAX_INDEX, VARIABLES)</code></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td><code>MAX_INDEX : dvar</code>&lt;br&gt;<code>VARIABLES : collection(index-int, var-dvar)</code></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>`</td>
<td>VARIABLES</td>
</tr>
<tr>
<td>Purpose</td>
<td><code>MAX_INDEX</code> is one of the indices of the collection of variables <code>VARIABLES</code> corresponding to its maximum value.</td>
<td></td>
</tr>
</tbody>
</table>
| Example     | `\[
\begin{pmatrix}
\text{index - 1} & \text{var} & 3, \\
\text{index - 2} & \text{var} & 2, \\
\text{index - 3} & \text{var} & 7, \\
\text{index - 4} & \text{var} & 2, \\
\text{index - 5} & \text{var} & 7
\end{pmatrix}
\]`<br>The attribute `var = 7` of the third and fifth items of the collection `VARIABLES` is the maximum value over values `3, 2, 7, 2, 7`. Consequently, the `max_index` constraint holds since its first argument `MAX_INDEX` is set to `3 ∈ \{3, 5\}`. |       |
| Typical     | `|VARIABLES| > 0`<br>`range(VARIABLES.var) > 1` |       |
| Symmetries  | • Items of `VARIABLES` are **permutable**.<br>• One and the same constant can be **added** to the `var` attribute of all items of `VARIABLES`. |       |
| See also    | comparison swapped: `min_index`. |       |
| Keywords    | characteristic of a constraint: maximum.<br>constraint type: order constraint.<br>modelling: functional dependency. |       |
Arc input(s)  VARIABLES
Arc generator  $\text{CLIQUIE} \rightarrow \text{collection}(\text{variables1}, \text{variables2})$
Arc arity  2
Arc constraint(s)  $\bigvee \left( \text{variables1}.\text{key} = \text{variables2}.\text{key}, \right.$
$\left. \text{variables1}.\text{var} > \text{variables2}.\text{var} \right)$
Graph property(ies)  ORDER(0, 0, index) = MAX_INDEX

Graph model
Parts (A) and (B) of Figure 5.407 respectively show the initial and final graph associated with the Example slot. Since we use the ORDER graph property, the vertex of rank 0 (without considering the loops) of the final graph is outlined with a thick circle.

Figure 5.407: Initial and final graph of the max_index constraint
5.219  **max_n**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[26]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>\texttt{max_n}(\texttt{MAX}, \texttt{RANK}, \texttt{VARIABLES})</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>\texttt{MAX} : dvar \texttt{RANK} : int \texttt{VARIABLES} : \texttt{collection}(\texttt{var} - \texttt{dvar})</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>\texttt{RANK} \geq 0 \texttt{RANK} &lt;</td>
<td>\texttt{VARIABLES}</td>
</tr>
<tr>
<td>Purpose</td>
<td>\texttt{MAX} is the maximum value of rank \texttt{RANK} (i.e., the \texttt{RANK}^{th} largest distinct value) of the collection of domain variables \texttt{VARIABLES}. Sinks have a rank of 0.</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>\texttt{(6, 1, \langle 3, 1, 7, 1, 6 \rangle)}</td>
<td></td>
</tr>
</tbody>
</table>

The \texttt{max_n} constraint holds since its first argument \texttt{MAX} = 6 is fixed to the second (i.e., \texttt{RANK} + 1) largest distinct value of the collection \langle 3, 1, 7, 1, 6 \rangle.

| Typical     | \texttt{RANK} \geq 0 \texttt{RANK} < 3 \texttt{|VARIABLES|} > 1 \texttt{range}(\texttt{VARIABLES}.\texttt{var}) > 1 |       |
| Symmetries  | • Items of \texttt{VARIABLES} are \texttt{permutable}. • One and the same constant can be \texttt{added} to \texttt{MAX} as well as to the \texttt{var} attribute of all items of \texttt{VARIABLES}. |       |
| Arg. properties | Functional dependency: \texttt{MAX} determined by \texttt{RANK} and \texttt{VARIABLES}. |       |
| Algorithm   | [26]. |       |
| Reformulation | The constraint \texttt{among_var}(1, \langle \texttt{MAX}, \texttt{VARIABLES} \rangle) enforces \texttt{MAX} to be assigned one of the values of \texttt{VARIABLES}. The constraint \texttt{nvalue}(\texttt{NVAL}, \texttt{VARIABLES}) provides a hand on the number of distinct values assigned to the variables of \texttt{VARIABLES}. By associating to each variable \texttt{V}_i (i \in [1, |\texttt{VARIABLES}|]) of the \texttt{VARIABLES} collection a rank variable \texttt{R}_i \in [0, |\texttt{VARIABLES}| - 1] with the reified constraint \texttt{R}_i = \texttt{RANK} \Leftrightarrow \texttt{V}_i = \texttt{MAX}, the inequality \texttt{R}_i \leq \texttt{NVAL}, and by creating for each pair of variables \texttt{V}_i, \texttt{V}_j (i < j \in [1, |\texttt{VARIABLES}|]) the reified constraints \texttt{V}_i > \texttt{V}_j \Leftrightarrow \texttt{R}_i < \texttt{R}_j. |       |
\[ V_i = V_j \iff R_i = R_j, \]
\[ V_i < V_j \iff R_i > R_j, \]

one can reformulate the max,\( n \) constraint in term of \( 3 \cdot \frac{|\text{VARIABLES}| \cdot (|\text{VARIABLES}| - 1)}{2} + 1 \) reified constraints.

See also

- comparison swapped: \( \text{min}, n \).
- generalisation: \( \text{maximum} \) (absolute maximum replaced by maximum or order \( n \)).

Keywords

- characteristic of a constraint: rank, maximum.
- constraint arguments: pure functional dependency.
- constraint type: order constraint.
Arc input(s)  VARIABLES
Arc generator  \( \text{CLIQUE} \rightarrow \text{collection}(\text{variables1}, \text{variables2}) \)
Arc arity  2
Arc constraint(s)  \( \bigvee \left( \begin{array}{l}
\text{variables1.key} = \text{variables2.key}, \\
\text{variables1.var} > \text{variables2.var}
\end{array} \right) \)
Graph property(ies)  \( \text{ORDER}(\text{RANK}, \text{MININT}, \text{var}) = \text{MAX} \)

Graph model

Parts (A) and (B) of Figure 5.408 respectively show the initial and final graph associated with the Example slot. Since we use the ORDER graph property, the vertex of rank 1 (without considering the loops) of the final graph is outlined with a thick circle.

Figure 5.408: Initial and final graph of the max_n constraint
### 5.220 max_nvalue

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from \textit{nvalue}.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>\texttt{max_nvalue(MAX, VARIABLES)}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Arguments | \begin{align*} \text{MAX} &: \hspace{1em} \texttt{dvar} \\
| & \text{VARIABLES} &: \hspace{1em} \texttt{collection(var-dvar)} \end{align*} |       |           |
| Restrictions | \begin{align*} \text{MAX} &\geq 1 \\
| & \text{MAX} \leq |\text{VARIABLES}| \\
| & \text{required(VARIABLES, var)} \end{align*} |       |           |
| Purpose | \begin{quote} \texttt{MAX} is the maximum number of times that the same value is taken by the variables of the collection \texttt{VARIABLES}. \end{quote} |       |           |

\[
\begin{pmatrix}
\text{var} - 9, \\
\text{var} - 1, \\
\text{var} - 7, \\
\text{var} - 1, \\
\text{var} - 6, \\
\text{var} - 7, \\
\text{var} - 7, \\
\text{var} - 4, \\
\text{var} - 9
\end{pmatrix}
\]

In the example, values 1, 4, 6, 7, 9 are respectively used 3, 1, 1, 3, 2 times. So the maximum number of time \texttt{MAX} that a same value occurs is 3. Consequently the \texttt{max_nvalue} constraint holds.

| Typical | \begin{align*} \text{MAX} &> 1 \\
| & \text{MAX} \leq |\text{VARIABLES}| \\
| & |\text{VARIABLES}| > 1 \\
| & \text{range(VARIABLES,var)} > 1 \end{align*} |       |           |
| Symmetries | \begin{itemize} \\
| & \text{Items of VARIABLES} \text{ are \textit{permutable}.} \\
| & \text{All occurrences of two distinct values of VARIABLES.var can be \textit{swapped}; all occurrences of a value of VARIABLES.var can be \textit{renamed} to any unused value.} \end{itemize} |       |           |
| Arg. properties | \textbf{Functional dependency}: \texttt{MAX} determined by \texttt{VARIABLES}. |       |           |
| Usage | This constraint may be used in order to replace a set of \textit{count} or \textit{among} constraints were one would have to generate explicitly one constraint for each potential value. Also useful for constraining the number of occurrences of the mostly used value without knowing this value in advance and without giving explicitly an upper limit on the number of occurrences of each value as it is done in the \textit{global_cardinality} constraint. |       |           |
Reformulation

Assume that VARIABLES is not empty. Let $\alpha$ and $\beta$ respectively denote the smallest and largest possible values that can be assigned to the variables of the VARIABLES collection. Let the variables $O_\alpha, O_{\alpha+1}, \ldots, O_\beta$ respectively correspond to the number of occurrences of values $\alpha, \alpha + 1, \ldots, \beta$ within the variables of the VARIABLES collection.

The max_nvalue constraint can be expressed as the conjunction of the following two constraints:

\[
\text{global_cardinality} (\text{VARIABLES}, \langle \text{val} - \alpha \text{ nocurrence} - O_\alpha, \text{val} - \alpha + 1 \text{ nocurrence} - O_{\alpha+1}, \ldots, \text{val} - \beta \text{ nocurrence} - O_\beta \rangle),
\]

\[
\text{maximum}(\text{MAX}, \langle O_\alpha, O_{\alpha+1}, \ldots, O_\beta \rangle).
\]

See also

common keyword: among (counting constraint), count, global_cardinality (value constraint, counting constraint), min_nvalue, nvalue (counting constraint).

Keywords

application area: assignment.
characteristic of a constraint: maximum, automaton, automaton with array of counters.
constraint arguments: pure functional dependency.
constraint type: value constraint, counting constraint.
final graph structure: equivalence.
modelling: maximum number of occurrences, functional dependency.
<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>(CLIQUE)→collection(variables1,variables2)</td>
</tr>
<tr>
<td>Arc arity</td>
<td>2</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>variables1.var = variables2.var</td>
</tr>
<tr>
<td>Graph property(ies)</td>
<td>MAX_NSCC = (\text{MAX})</td>
</tr>
</tbody>
</table>

**Graph model**

Because of the arc constraint, each strongly connected component of the final graph corresponds to a distinct value that is assigned to a subset of variables of the VARIABLES collection. Therefore the number of vertices of the largest strongly connected component is equal to the mostly used value.

Parts (A) and (B) of Figure 5.409 respectively show the initial and final graph associated with the Example slot. Since we use the MAX_NSCC graph property, we show the largest strongly connected component of the final graph.
Figure 5.409: Initial and final graph of the max_nvalue constraint
Automaton

Figure 5.410 depicts the automaton associated with the max_nvalue constraint. To each item of the collection VARIABLES corresponds a signature variable $S_i$ that is equal to 0.

$$
\{C[=0] \}
\downarrow
\
S_i : \text{maximum}(N,C) \quad 0, \quad \{C[VAR_i] = C[VAR_i] + 1\}
$$

Figure 5.410: Automaton of the max_nvalue constraint
5.221  max.size set.of_consecutive_var

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>N. Beldiceanu</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>max.size set.of_consecutive_var(MAX, VARIABLES)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>MAX : dvar VARIABLES : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>MAX ≥ 1 MAX ≤</td>
<td>VARIABLES</td>
</tr>
<tr>
<td>Purpose</td>
<td>MAX is the size of the largest set of variables of the collection VARIABLES that all take their value in a set of consecutive values.</td>
<td></td>
</tr>
</tbody>
</table>

Example

\[
\begin{pmatrix}
\text{var} - 3, \\
\text{var} - 1, \\
\text{var} - 3, \\
\text{var} - 7, \\
\text{var} - 4, \\
\text{var} - 1, \\
\text{var} - 2, \\
\text{var} - 8, \\
\text{var} - 7, \\
\text{var} - 6
\end{pmatrix}
\]

In the example, the two sets \{3, 1, 3, 4, 1, 2\} and \{7, 8, 7, 6\} take respectively their values in the two following sets of consecutive values \{1, 2, 3, 4\} and \{6, 7, 8\}. Consequently, the max.size set.of_consecutive_var constraint holds since the cardinality of the largest set of variables is 6.

Typical

MAX < |VARIABLES|
|VARIABLES| > 0
range(VARIABLES.var) > 1

Symmetries

- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

Functional dependency: MAX determined by VARIABLES.

See also

common keyword: nset.of_consecutive_values (consecutive values).
Keywords

characteristic of a constraint: consecutive values, maximum.
constraint arguments: pure functional dependency.
constraint type: value constraint.
modelling: functional dependency.
**Arc input(s)**

VARIABLES

**Arc generator**

$CLIQUE \rightarrow \text{collection}(\text{variables1,variables2})$

**Arc arity**

2

**Arc constraint(s)**

$\text{abs(variables1.var - variables2.var)} \leq 1$

**Graph property(ies)**

$\text{MAX_NSCC} = \text{MAX}$

---

**Graph model**

Since the arc constraint is symmetric each strongly connected component of the final graph corresponds exactly to one connected component of the final graph.

Parts (A) and (B) of Figure 5.4 show respectively the initial and final graph associated with the Example slot. Since we use the MAX_NSCC graph property, we show the largest strongly connected component of the final graph.
Figure 5.411: Initial and final graph of the max_size_set_of_consecutive_var constraint
### 5.222 maximum

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin: CHIP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint: maximum(MAX, VARIABLES)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonym: max.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments: MAX : dvar, VARIABLES : collection(var - dvar)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose: MAX is the maximum value of the collection of domain variables VARIABLES.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example: ((7,(3,2,7,2,6)))</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The maximum constraint holds since its first argument (\text{MAX} = 7) is fixed to the maximum value of the collection (\langle 3,2,7,2,6 \rangle).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Typical:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symmetries:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arg. properties:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Usage: In some project scheduling problems one has to introduce dummy activities that correspond for instance to the completion time of a given set of activities. In this context one can use the maximum constraint to get the maximum completion time of a set of tasks.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remark: Note that maximum is a constraint and not just a function that computes the maximum value of a collection of variables; potential values of MAX influence the variables of VARIABLES, and reciprocally potential values that can be assigned to variables of VARIABLES influence MAX. The maximum constraint is called max in JaCoP (<a href="http://www.jacop.eu/">http://www.jacop.eu/</a>).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algorithm: A filtering algorithm for the maximum constraint is described in [26]. The maximum constraint is entailed if all the following conditions hold:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. MAX is fixed.
2. At least one variable of VARIABLES is assigned value MAX.
3. All variables of VARIABLES have their maximum value less than or equal to value MAX.

**Systems**

- max in Choco, max in Gecode, max in JaCoP, maximum in MiniZinc, maximum in SICStus.

**See also**

- common keyword: minimum (order constraint).
- comparison swapped: minimum.
- generalisation: maximum modulo (variable replaced by variable mod constant).
- implied by: or.
- implies: in.
- soft variant: open maximum (open constraint).
- specialisation: max_n (maximum or order n replaced by absolute maximum).
- uses in its reformulation: tree_range.

**Keywords**

- characteristic of a constraint: maximum, automaton, automaton without counters, reified automaton constraint.
- constraint arguments: pure functional dependency.
- constraint network structure: centered cyclic(1) constraint network(1).
- constraint type: order constraint.
- filtering: arc-consistency, entailment.
- modelling: balanced assignment, functional dependency.
**Arc input(s)**
VARIABLES

**Arc generator**
$CLIQUE \rightarrow \text{collection}(\text{variables1}, \text{variables2})$

**Arc arity**
2

**Arc constraint(s)**
$\forall \left( \begin{array}{l}
\text{variables1.key} = \text{variables2.key}, \\
\text{variables1.var} > \text{variables2.var}
\end{array} \right)$

**Graph property(ies)**
$\text{ORDER}(0, \text{MININT}.\text{var}) = \text{MAX}$

**Graph model**
We use a similar definition that the one that was utilised for the minimum constraint. Within the arc constraint, we replace the comparison operator $<$ by $>$. Parts (A) and (B) of Figure 5.412 respectively show the initial and final graph associated with the Example slot. Since we use the ORDER graph property, the vertex of rank 0 (without considering the loops) of the final graph is outlined with a thick circle.

(A)

(B)

Figure 5.412: Initial and final graph of the maximum constraint
Figure 5.413 depicts the automaton associated with the maximum constraint. Let \( \text{VAR}_i \) be the \( i^{th} \) variable of the \text{VARIABLES} collection. To each pair \( (\text{MAX}, \text{VAR}_i) \) corresponds a signature variable \( S_i \) as well as the following signature constraint: \( (\text{MAX} > \text{VAR}_i \iff S_i = 0) \land (\text{MAX} = \text{VAR}_i \iff S_i = 1) \land (\text{MAX} < \text{VAR}_i \iff S_i = 2) \).

Figure 5.413: Automaton of the maximum constraint

Figure 5.414: Hypergraph of the reformulation corresponding to the automaton of the maximum constraint
5.223 **maximum_modulo**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
</table>

**Origin**
Derived from maximum.

**Constraint**
maximum_modulo(MAX, VARIABLES, M)

**Arguments**
- **MAX**: dvar
- **VARIABLES**: collection(var−dvar)
- **M**: int

**Restrictions**
- |VARIABLES| > 0
- M > 0
- required(VARIABLES, var)

**Purpose**
MAX is a maximum value of the collection of domain variables VARIABLES according to the following partial ordering: (X mod M) < (Y mod M).

**Example**
(5, (9, 1, 7, 6, 5), 3)

The maximum_modulo constraint holds since its first argument MAX is set to value 5, where 5 mod 3 = 2 is greater than or equal to all the expressions 9 mod 3 = 0, 1 mod 3 = 1, 7 mod 3 = 1 and 6 mod 3 = 0.

**Typical**
- M > 1
- M < maxval(VARIABLES.var)
- |VARIABLES| > 1
- range(VARIABLES.var) > 1

**Symmetry**
Items of VARIABLES are permutable.

**Arg. properties**
Functional dependency: MAX determined by VARIABLES and M.

**See also**
- comparison swapped: minimum_modulo.
- specialisation: maximum(variable mod constant replaced by variable).

**Keywords**
characteristic of a constraint: modulo, maximum.
constraint arguments: pure functional dependency.
constraint type: order constraint.
modelling: functional dependency.
Graph model

Parts (A) and (B) of Figure 5.415 respectively show the initial and final graph associated with the Example slot. Since we use the ORDER graph property, the vertex of rank 0 (without considering the loops) of the final graph is outlined with a thick circle.

Figure 5.415: Initial and final graph of the maximum modulo constraint
5.224 meet_sboxes

**DESCRIPTION**

Geometry, derived from [318]

**Constraint**

meet_sboxes(K, DIMS, OBJECTS, SBOXES)

**Synonym**

meet.

**Types**

VARIABLES : \(\text{collection}(v\leftarrow	ext{dvar})\)
INTEGERS : \(\text{collection}(v\leftarrow\text{int})\)
POSITIVES : \(\text{collection}(v\leftarrow\text{int})\)

**Arguments**

K : \(\text{int}\)
DIMS : \(\text{sint}\)
OBJECTS : \(\text{collection}(\text{oid}\leftarrow\text{int}, \text{sid}\leftarrow\text{int}, x\leftarrow\text{VARIABLES})\)
SBOXES : \(\text{collection}(\text{sid}\leftarrow\text{int}, t\leftarrow\text{INTEGERS}, l\leftarrow\text{POSITIVES})\)

**Restrictions**

\[|\text{VARIABLES}| \geq 1\]
\[|\text{INTEGERS}| \geq 1\]
\[|\text{POSITIVES}| \geq 1\]

required(\text{VARIABLES}, v)
\[|\text{VARIABLES}| = K\]

required(\text{INTEGERS}, v)
\[|\text{INTEGERS}| = K\]

required(\text{POSITIVES}, v)
\[|\text{POSITIVES}| = K\]

POSITIVES.v > 0

K > 0

DIMS \geq 0

DIMS < K

increasing_seq(\text{OBJECTS}, [\text{oid}])

required(\text{OBJECTS}, [\text{oid}, \text{sid}, x])

\text{OBJECTS.oid} \geq 1

\text{OBJECTS.oid} \leq |\text{OBJECTS}|

\text{OBJECTS.sid} \geq 1

\text{OBJECTS.sid} \leq |\text{SBOXES}|

|\text{SBOXES}| \geq 1

required(\text{SBOXES}, [\text{sid}, t, l])

\text{SBOXES.sid} \geq 1

\text{SBOXES.sid} \leq |\text{SBOXES}|

do_not_overlap(\text{SBOXES})
Holds if, for each pair of objects \((O_i, O_j), i \neq j, O_i\) and \(O_j\) meet with respect to a set of dimensions depicted by \(\text{DIMS}\). Each \textit{shape} is defined as a finite set of shifted boxes, where each shifted box is described by a box in a \(K\)-dimensional space at a given offset (from the origin of the shape) with given sizes. More precisely, a \textit{shifted box} is an entity defined by its shape id \(\text{sid}\), shift offset \(\tau\), and sizes \(\ell\). Then, a shape is defined as the union of shifted boxes sharing the same shape id. An \textit{object} is an entity defined by its unique object identifier \(\text{oid}\), shape id \(\text{sid}\) and origin \(x\).

Two objects \(O_i\) and object \(O_j\) \textit{meet} with respect to a set of dimensions depicted by \(\text{DIMS}\) if and only if the two following conditions hold:

- For all shifted box \(s_i\) associated with \(O_i\) and for all shifted box \(s_j\) associated with \(O_j\) there exists a dimension \(d \in \text{DIMS}\) such that (1) the start of \(s_j\) in dimension \(d\) is greater than or equal to the end of \(s_j\) in dimension \(d\), or (2) the start of \(s_j\) in dimension \(d\) is greater than or equal to the end of \(s_j\) in dimension \(d\) (i.e., there is no overlap between the shifted box of \(O_i\) and the shifted box of \(O_j\)).

- There exists a shifted box \(s_i\) of \(O_i\) and there exists a shifted box \(s_j\) of \(O_j\) such that for all dimensions \(d\) (1) the end of \(s_i\) in dimension \(d\) is greater than or equal to the start of \(s_j\) in dimension \(d\), and (2) the end of \(s_j\) in dimension \(d\) is greater than or equal to the start of \(s_i\) in dimension \(d\) (i.e., at least two shifted box of \(O_i\) and \(O_j\) are in contact).

\[
\begin{align*}
2, \{0, 1\}, & \\
\{\text{oid} = 1, \text{sid} = 1, x = \{3, 2\}, \}
\{\text{oid} = 2, \text{sid} = 2, x = \{4, 1\},\} & \\
\{\text{oid} = 3, \text{sid} = 4, x = \{3, 4\}\} & \\
\text{sid} = 1, \tau = (0, 0), x = \{1, 2\} & \\
\text{sid} = 2, \tau = (0, 0), x = \{1, 1\} & \\
\text{sid} = 2, \tau = (1, 0), x = \{1, 3\} & \\
\text{sid} = 2, \tau = (0, 2), x = \{1, 1\} & \\
\text{sid} = 3, \tau = (0, 0), x = \{3, 1\} & \\
\text{sid} = 3, \tau = (0, 1), x = \{1, 1\} & \\
\text{sid} = 3, \tau = (2, 1), x = \{1, 1\} & \\
\text{sid} = 4, \tau = (0, 0), x = \{1, 1\} & \\
\end{align*}
\]

Figure \ref{fig:objects} shows the objects of the example. Since all the pairs of objects meet the \textit{meet\_sboxes} constraint holds.

**Typical**

\([\text{OBJECTS}] > 1\)

**Symmetries**

- Items of \textit{OBJECTS} are \textit{permutable}.
- Items of \textit{SBOXES} are \textit{permutable}.
- Items of \textit{OBJECTS.x}, \textit{SBOXES.t} and \textit{SBOXES.1} are \textit{permutable} \textit{(same permutation used)}.

**Arg. properties**

\textit{Suffix-contractible} wrt. \textit{OBJECTS}.

**Remark**

One of the eight relations of the \textit{Region Connection Calculus} \cite{[318]}. 
See also common keyword: contains_sboxes, coveredby_sboxes, covers_sboxes, disjoint_sboxes, equal_sboxes, inside_sboxes(rcc8), non_overlap_sboxes(geometrical constraint,logic), overlap_sboxes(rcc8).

Keywords constraint type: logic.
geometry: geometrical constraint, rcc8.
Figure 5.416: The three objects of the example
Logic

- `\operatorname{origin}(O1, S1, D) \overset{\text{def}}{=} O1 \cdot x(D) + S1 \cdot t(D)`
- `\operatorname{end}(O1, S1, D) \overset{\text{def}}{=} O1 \cdot x(D) + S1 \cdot t(D) + S1 \cdot l(D)`
- `\operatorname{non\_overlap\_sboxes}(\text{Dims}, O1, S1, O2, S2) \overset{\text{def}}{=} \exists D \in \text{Dims} \left( \begin{array}{l}
    \operatorname{end}(O1, S1, D) \\
    \operatorname{origin}(S2, D)
  \end{array} \right) \lor
\left( \begin{array}{l}
    \operatorname{end}(O2, S2, D) \\
    \operatorname{origin}(O1, D)
  \end{array} \right)

- `\operatorname{meet\_sboxes}(\text{Dims}, O1, S1, O2, S2) \overset{\text{def}}{=} \exists D \in \text{Dims} \left( \begin{array}{l}
    \operatorname{end}(O1, S1, D) = \\
    \operatorname{origin}(O2, S2, D)
  \end{array} \right) \lor
\left( \begin{array}{l}
    \operatorname{end}(O2, S2, D) = \\
    \operatorname{origin}(O1, S1, D)
  \end{array} \right)

- `\operatorname{meet\_objects}(\text{Dims}, O1, O2) \overset{\text{def}}{=} \forall S1 \in \operatorname{sboxes}(O1, \text{sid}) \left( \begin{array}{l}
    \forall S2 \in \operatorname{sboxes}(O2, \text{sid}) \\
    \operatorname{non\_overlap\_sboxes}(\text{Dims}, O1, S1, O2, S2)
  \end{array} \right) \land
\left( \begin{array}{l}
    \exists S1 \in \operatorname{sboxes}(O1, \text{sid}) \\
    \exists S2 \in \operatorname{sboxes}(O2, \text{sid}) \\
    \operatorname{meet\_sboxes}(S1, S2)
  \end{array} \right)

- `\operatorname{all\_meet}(\text{Dims}, \text{OIDS}) \overset{\text{def}}{=} \forall O1 \in \operatorname{objects}(\text{OIDS}) \forall O2 \in \operatorname{objects}(\text{OIDS}) \left( \begin{array}{l}
    O1.\text{oid} < \Rightarrow \\
    O2.\text{oid}
  \end{array} \right) \operatorname{meet\_objects}(\text{Dims}, O1, O2)

- `\operatorname{all\_meet}(\text{DIMENSIONS}, \text{OIDS})`
5.225 min_index

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>N. Beldiceanu</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>( \text{min}_\text{index}(\text{MIN}_\text{INDEX}, \text{VARIABLES}) )</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>( \text{MIN}_\text{INDEX} : \text{dvar} ) &lt;br&gt; ( \text{VARIABLES} : \text{collection}(\text{index} - \text{int}, \text{var} - \text{dvar}) )</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>(</td>
<td>\text{VARIABLES}</td>
</tr>
<tr>
<td>Purpose</td>
<td>( \text{MIN}_\text{INDEX} ) is one of the indices of the collection of variables ( \text{VARIABLES} ) corresponding to its minimum value.</td>
<td></td>
</tr>
</tbody>
</table>

Example

\[
\begin{align*}
2, & \{ \text{index} - 1, \text{var} - 3, \\
3, & \{ \text{index} - 2, \text{var} - 2, \\
4, & \{ \text{index} - 3, \text{var} - 7, \\
index - 5, & \{ \text{var} - 6, \\
index - 1, & \{ \text{var} - 3, \\
index - 2, & \{ \text{var} - 2, \\
index - 4, & \{ \text{var} - 7, \\
index - 5, & \{ \text{var} - 6, \\
\end{align*}
\]

The attribute \( \text{var} = 2 \) of the second and fourth items of the collection \( \text{VARIABLES} \) is the minimum value over values 3, 2, 7, 2, 6. Consequently, both \( \text{min}\_\text{index} \) constraints hold since their first arguments \( \text{MIN}\_\text{INDEX} \) are respectively set to 2 and 4.

Typical

\[
\begin{align*}
|\text{VARIABLES}| & > 0 \\
\text{range}(\text{VARIABLES}.\text{var}) & > 1 \\
\end{align*}
\]

Symmetries

- Items of \( \text{VARIABLES} \) are permmutable.
- One and the same constant can be added to the \( \text{var} \) attribute of all items of \( \text{VARIABLES} \).

Usage

Within the context of scheduling, assume the variables of the \( \text{VARIABLES} \) collection correspond to the starts of a set of tasks. Then \( \text{MIN}\_\text{INDEX} \) gives the indexes of those tasks that can be scheduled first.
See also

- comparison swapped: max_index.

Keywords

- characteristic of a constraint: minimum.
- constraint type: order constraint.
Arc input(s)  VARIABLES
Arc generator  $CLIQUE \rightarrow \text{collection}(\text{variables1}, \text{variables2})$
Arc arity  2
Arc constraint(s)  $\bigvee \left(\text{variables1.key} = \text{variables2.key}, \quad \text{variables1.var} < \text{variables2.var}\right)$
Graph property(ies)  ORDER($0, 0, \text{index}$) = MIN_INDEX

Graph model

Parts (A) and (B) of Figure 5.417 respectively show the initial and final graph associated with the two examples of the Example slot. Since we use the ORDER graph property, the vertices of rank 0 (without considering the loops) of the final graph are outlined with a thick circle.

![Graph Model](image)

Figure 5.417: Initial and final graph of the min_index constraint
### 5.226 min_n

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>[26]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>min_n(MIN, RANK, VARIABLES)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| **Arguments** | MIN : dvar  
RANK : int  
VARIABLES : collection(var–dvar) |       |           |
| **Restrictions** | |       |           |
| | | |           |
| **Purpose** | MIN is the minimum value of rank RANK (i.e., the RANK<sup>th</sup> smallest distinct value) of the collection of domain variables VARIABLES. Sources have a rank of 0. |       |           |
| **Example** | (3, 1, ⟨3, 1, 7, 1, 6⟩) |       |           |
| | The min_n constraint holds since its first argument MIN = 3 is fixed to the second (i.e., RANK + 1) smallest distinct value of the collection ⟨3, 1, 7, 1, 6⟩. Note that identical values are only counted once: this is why the minimum of order 1 is 3 instead of 1. |       |           |
| **Typical** | RANK > 0  
RANK < 3  
| |           |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| **Symmetries** | • Items of VARIABLES are permutable.  
• One and the same constant can be added to MIN as well as to the var attribute of all items of VARIABLES. |       |           |
| **Arg. properties** | Functional dependency: MIN determined by RANK and VARIABLES. |       |           |
| **Algorithm** | [26] |       |           |
| **Reformulation** | The constraint among_var(1, ⟨MIN⟩, VARIABLES) enforces MIN to be assigned one of the values of VARIABLES. The constraint nvalue(NVAL, VARIABLES) provides a hand on the number of distinct values assigned to the variables of VARIABLES. By associating to each variable V<sub>i</sub> (i ∈ [1, |VARIABLES|]) of the VARIABLES collection a rank variable R<sub>i</sub> ∈ [0, |VARIABLES| − 1] with the reified constraint R<sub>i</sub> = RANK ⇔ V<sub>i</sub> = MIN, the inequality R<sub>i</sub> < NVAL, and by creating for each pair of variables V<sub>i</sub>, V<sub>j</sub> (i, j < i ∈ [1, |VARIABLES|]) the reified constraints |       |           |
\[ V_i < V_j \iff R_i < R_j, \]
\[ V_i = V_j \iff R_i = R_j, \]
\[ V_i > V_j \iff R_i > R_j, \]

one can reformulate the \texttt{min\_n} constraint in term of
\[ 3 \cdot \left\lceil \frac{|\text{Variables}|}{2} \right\rceil + 1 \text{ reified constraints.} \]

See also

- comparison swapped: \texttt{max\_n}.
- generalisation: \texttt{minimum} (absolute minimum replaced by minimum or order \texttt{n}).
- used in reformulation: \texttt{among\_var, nvalue}.

Keywords

- characteristic of a constraint: rank, minimum, maxint, automaton, automaton with array of counters.
- constraint arguments: pure functional dependency.
- constraint type: order constraint.
### Arc input(s)

VARIABLES

### Arc generator

CLIQUE→\textit{collection}(variables1, variables2)

### Arc arity

2

### Arc constraint(s)

\[ \bigvee \left( \begin{array}{c} \text{variables1.key} = \text{variables2.key}, \\ \text{variables1.var} < \text{variables2.var} \end{array} \right) \]

### Graph property(ies)

ORDER(RANK, MAXINT, var) = MIN

### Graph model

Parts (A) and (B) of Figure 5.418 respectively show the initial and final graph associated with the Example slot. Since we use the ORDER graph property, the vertex of rank 1 (without considering the loops) of the final graph is shown in grey.

![Graph Model](image)

Figure 5.418: Initial and final graph of the min_n constraint
Automaton

Figure 5.419 depicts the automaton associated with the \texttt{min\_n} constraint. Figure 5.419 depicts the automaton associated with the \texttt{min\_n} constraint. To each item of the collection \texttt{VARIABLES} corresponds a signature variable \( S_i \) that is equal to 1.

\[
\begin{cases}
\{C[0]=0, D=\text{maxint}\} \\
S_i \text{ith\_pos\_different\_from\_0(RANK\_+1,M,C)} \text{\ MIN=M+D-1} \\
1, \{C[VAR_i]=C[VAR_i]+1, D=\text{min}(D,VAR_i)\}
\end{cases}
\]

Figure 5.419: Automaton of the \texttt{min\_n} constraint
### 5.227 min_nvalue

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>N. Beldiceanu</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>min_nvalue(MIN, VARIABLES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>MIN : dvar</td>
<td>VARIABLES : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>MIN ≥ 1</td>
<td>MIN ≤</td>
<td>VARIABLES</td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES, var)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>MIN is the minimum number of times that the same value is taken by the variables of the collection VARIABLES.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the example, values 1, 7, 9 are respectively used 3, 5, 2 times. So the minimum number of time MIN that a same value occurs is 2. Consequently the min_nvalue constraint holds.

#### Typical

\[
2 \times \text{MIN} \leq |\text{VARIABLES}|
\]

\[
|\text{VARIABLES}| > 1
\]

\[
\text{range}(\text{VARIABLES}.\text{var}) > 1
\]

#### Symmetries

- Items of VARIABLES are **permutable**.
- All occurrences of two distinct values of VARIABLES.var can be **swapped**; all occurrences of a value of VARIABLES.var can be **renamed** to any unused value.

#### Arg. properties

**Functional dependency**: MIN determined by VARIABLES.

#### Usage

This constraint may be used in order to replace a set of **count** or **among** constraints were one would have to generate explicitly one constraint for each potential value. Also useful for constraining the number of occurrences of the less used value without knowing this value in advance and without giving explicitly a lower limit on the number of occurrences of each value as it is done in the **global_cardinality** constraint.
Assume that VARIABLES is not empty. Let $\alpha$ and $\beta$ respectively denote the smallest and largest possible values that can be assigned to the variables of the VARIABLES collection. Let the variables $O_{\alpha}, O_{\alpha+1}, \ldots, O_{\beta}$ respectively correspond to the number of occurrences of values $\alpha, \alpha + 1, \ldots, \beta$ within the variables of the VARIABLES collection. The $\min_n$ value constraint can be expressed as the conjunction of the following two constraints:

$$\text{global_cardinality} \ (\text{VARIABLES},$$
$$\langle \text{val} - \alpha \ \text{noccurrence} - O_{\alpha},$$
$$\text{val} - \alpha + 1 \ \text{noccurrence} - O_{\alpha+1},$$
$$\ldots$$
$$\text{val} - \beta \ \text{noccurrence} - O_{\beta} \rangle),$$

$$\min_n (\text{MIN}, 1, \langle 0, O_{\alpha}, O_{\alpha+1}, \ldots, O_{\beta} \rangle).$$

We use a $\min_n$ constraint (with its RANK parameter set to 1) instead of a minimum constraint in order to discard the smallest value 0.

See also

**common keyword:**
- among (counting constraint),
- count,
- global_cardinality (value constraint,counting constraint),
- max_nvalue,
- nvalue (counting constraint).

**Keywords**

- application area: assignment.
- characteristic of a constraint: minimum, automaton, automaton with array of counters.
- constraint arguments: pure functional dependency.
- constraint type: value constraint, counting constraint.
- final graph structure: equivalence.
- modelling: minimum number of occurrences, functional dependency.
<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>$CLIQUE\rightarrow collection(variables_1, variables_2)$</td>
</tr>
<tr>
<td>Arc arity</td>
<td>2</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>$variables_1.var = variables_2.var$</td>
</tr>
<tr>
<td>Graph property(ies)</td>
<td>$MIN_{NSCC} = \text{MIN}$</td>
</tr>
</tbody>
</table>

**Graph model**

Parts (A) and (B) of Figure 5.420 respectively show the initial and final graph. Since we use the $MIN_{NSCC}$ graph property, we show the smallest strongly connected component of the final graph associated with the **Example** slot.
Figure 5.420: Initial and final graph of the min_nvalue constraint
Figure 5.421 depicts the automaton associated with the min_nvalue constraint. To each item of the collection VARIABLES corresponds a signature variable \( S_i \) that is equal to 0.

\[
\begin{align*}
S: & \quad \text{minimum_except_0}(N, C) \rightarrow 0, \\
& \quad \{ C[VAR_1] = C[VAR_1] + 1 \}
\end{align*}
\]

Figure 5.421: Automaton of the min_nvalue constraint
### 5.228  min_size_set_of_consecutive_var

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>N. Beldiceanu</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>\text{min_size_set_of_consecutive_var}(\text{MIN}, \text{VARIABLES})</td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\text{MIN} : dvar</td>
<td>\text{VARIABLES} : \text{collection}(\text{var} - \text{dvar})</td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\text{MIN} \geq 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\text{MIN} \leq</td>
<td>\text{VARIABLES}</td>
<td></td>
</tr>
<tr>
<td>\text{required}(\text{VARIABLES}, \text{var})</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIN is the size of the smallest set of variables of the collection \text{VARIABLES} that all take their value in a set of consecutive values.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the example, the two parts 3, 1, 3, 4, 1, 2 and 7, 8, 7, 6 take respectively their values in the two following sets of consecutive values \{1, 2, 3, 4\} and \{6, 7, 8\}. Consequently, the min_size_set_of_consecutive_var constraint holds since the cardinality of the smallest set of variables is 4.

| **Typical** | |
| \text{MIN} > 1 | |
| \text{MIN} < |\text{VARIABLES}| | |
| |\text{VARIABLES}| > 0 | |
| \text{range}(\text{VARIABLES}, \text{var}) > 1 | |

| **Symmetries** | |
| - Items of \text{VARIABLES} are \text{permutable}. | |
| - All occurrences of two distinct values of \text{VARIABLES}.\text{var} can be \text{swapped}. | |
| - One and the same constant can be \text{added} to the \text{var} attribute of all items of \text{VARIABLES}. | |

| **Arg. properties** | |
| Functional dependency: \text{MIN} determined by \text{VARIABLES}. | |

| **See also** | |
| common keyword: nset_of_consecutive_values(\text{consecutive values}). | |
Keywords

application area: assignment.
characteristic of a constraint: consecutive values, minimum.
constraint arguments: pure functional dependency.
constraint type: value constraint.
modelling: functional dependency.
Arc input(s) | VARIABLES
---|---
Arc generator | $\text{CLIQUE} \leftrightarrow \text{collection}(\text{variables1}, \text{variables2})$
Arc arity | 2
Arc constraint(s) | $\text{abs}(\text{variables1}.\text{var} - \text{variables2}.\text{var}) \leq 1$
Graph property(ies) | $\text{MIN\_NSCC} = \text{MIN}$

Graph model

Since the arc constraint is symmetric each strongly connected component of the final graph corresponds exactly to one connected component of the final graph.

Parts (A) and (B) of Figure 5.422 respectively show the initial and final graph associated with the Example slot. Since we use the MIN\_NSCC graph property, we show the smallest strongly connected component of the final graph.
Figure 5.422: Initial and final graph of the \texttt{min\_size\_set\_of\_consecutive\_var} constraint
5.229 minimum

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>CHIP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>minimum(MIN, VARIABLES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td>min.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Arguments   | MIN : dvar  
VARIABLES : collection(var − dvar) | | |
| Restrictions| | | |
| Purpose     | MIN is the minimum value of the collection of domain variables VARIABLES. | | |

Example

\[(2, (3, 2, 7, 2, 6))\]

The minimum constraint holds since its first argument \(\text{MIN} = 2\) is set to the minimum value of the collection \((3, 2, 7, 2, 6)\).

Typical

\[|\text{VARIABLES}| > 1\]
\[\text{range}(|\text{VARIABLES}.\text{var}|) > 1\]

Symmetries

- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped.
- One and the same constant can be added to MIN as well as to the var attribute of all items of VARIABLES.

Arg. properties

- Functional dependency: MIN determined by VARIABLES.
- Aggregate: MIN(min), VARIABLES(union).

Usage

In some project scheduling problems one has to introduce dummy activities that correspond for instance to the starting time of a given set of activities. In this context one can use the minimum constraint to get the minimum starting time of a set of tasks.

Remark

Note that minimum is a constraint and not just a function that computes the minimum value of a collection of variables; potential values of MIN influence the variables of VARIABLES, and reciprocally potential values that can be assigned to variables of VARIABLES influence MIN.

The minimum constraint is called min in JaCoP (http://www.jacop.eu/).
Algorithm

A filtering algorithm for the minimum constraint is described in [26].
The minimum constraint is entailed if all the following conditions hold:

1. MIN is fixed.
2. At least one variable of VARIABLES is assigned value MIN.
3. All variables of VARIABLES have their minimum value greater than or equal to value MIN.

Systems

min in Choco, min in Gecode, min in JaCoP, minimum in MiniZinc, minimum in SICStus.

Used in

minimum_greater_than, next_element, next_greater_element.

See also

common keyword: maximum (order constraint).
comparison swapped: maximum.
generalisation: minimum_modulo (variable replaced by variable mod constant).
implies by: and.
implies: in.
soft variant: minimum_except_0 (value 0 is ignored), open_minimum (open constraint).
specialisation: min_n (minimum or order n replaced by absolute minimum).
uses in its reformulation: cycle.

Keywords

characteristic of a constraint: minimum, maxint, automaton, automaton without counters, reified automaton constraint.
constraint arguments: pure functional dependency.
constraint network structure: centered cyclic(1) constraint network(1).
constraint type: order constraint.
filtering: arc-consistency, entailment.
modelling: functional dependency.
### Arc input(s)
- **VARIABLES**

### Arc generator
- $CLIQUE \rightarrow collection(variables1, variables2)$

### Arc arity
- 2

### Arc constraint(s)
- $\bigvee \left( variables1.key = variables2.key, variables1.var < variables2.var \right)$

### Graph property(ies)
- $ORDER(0, MAXINT, var) = \text{MIN}$

#### Graph model

The condition $variables1.key = variables2.key$ holds if and only if $variables1$ and $variables2$ correspond to the same vertex. It is used in order to enforce to keep all the vertices of the initial graph. $ORDER(0, MAXINT, var)$ refers to the source vertices of the graph, i.e., those vertices that do not have any predecessor.

Parts (A) and (B) of Figure 5.423 respectively show the initial and final graph associated with the Example slot. Since we use the $ORDER$ graph property, the vertices of rank 0 (without considering the loops) of the final graph are outlined with a thick circle.

![Graph Model](image)

Figure 5.423: Initial and final graph of the minimum constraint
Figure 5.424 depicts the automaton associated with the minimum constraint. Let \( \text{VAR}_i \) be the \( i^{th} \) variable of the VARIABLES collection. To each pair \((\text{MIN}, \text{VAR}_i)\) corresponds a signature variable \( S_i \) as well as the following signature constraint: \( (\text{MIN} < \text{VAR}_i \leftrightarrow S_i = 0) \land (\text{MIN} = \text{VAR}_i \leftrightarrow S_i = 1) \land (\text{MIN} > \text{VAR}_i \leftrightarrow S_i = 2) \).

Figure 5.424: Automaton of the minimum constraint

Figure 5.425: Hypergraph of the reformulation corresponding to the automaton of the minimum constraint
### 5.230 minimum_except_0

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Derived from minimum.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>minimum_except_0(MIN, VARIABLES, DEFAULT)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIN : dvar</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARIABLES : collection(var−dvar)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEFAULT : int</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIN &gt; 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIN ≤ DEFAULT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES</td>
<td>&gt; 0</td>
<td></td>
</tr>
<tr>
<td>required(VARIABLES, var)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARIABLES.var ≥ 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARIABLES.var ≤ DEFAULT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEFAULT &gt; 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

All variables of the collection VARIABLES are assigned a value that belongs to interval [0, DEFAULT]. MIN is the minimum value of the collection of domain variables VARIABLES, ignoring all variables that take 0 as value. When all variables of the collection VARIABLES are assigned value 0, MIN is set to the default value DEFAULT.

#### Purpose

The three examples of the minimum_except_0 constraint respectively hold since:

- Within the first example, MIN is set to the minimum value 3 of the collection (3, 7, 6, 7, 4, 7).
Within the second example, MIN is set to the minimum value 2 (ignoring value 0) of the collection \(\{3, 2, 0, 7, 2, 6\}\).

Finally, within the third example, MIN is set to the default value 1000000 since all items of the collection \(\{0, 0, 0, 0, 0\}\) are set to 0.

**Typical**

- Typical

```
| VARIABLES | > 1
range(VARIABLES.var) > 1
atleast(1, VARIABLES, 0)
```

**Symmetries**

- Items of VARIABLES are *permutable*.
- All occurrences of two distinct values of VARIABLES.var can be *swapped*.

**Arg. properties**

*Functional dependency:* MIN determined by VARIABLES and DEFAULT.

**Remark**

The joker value 0 makes sense only because we restrict the variables of the VARIABLES collection to take non-negative values.

**Reformulation**

By (1) associating to each variable \(V_i \ (i \in [1, |VARIABLES|])\) of the VARIABLES collection a *rank* variable \(R_i \in [0, |VARIABLES| - 1]\) with the reified constraint \(R_i = 1 \Leftrightarrow V_i = \text{MIN}\), and by creating for each pair of variables \(V_i, V_j \ (i, j < i \in [1, |VARIABLES|])\) the reified constraints

\[
V_i < V_j \Leftrightarrow R_i < R_j,
V_i = V_j \Leftrightarrow R_i = R_j,
V_i > V_j \Leftrightarrow R_i > R_j,
\]

and by (2) creating the reified constraint

\[
V_1 = 0 \land V_2 = 0 \land \ldots \land V_n = 0 \Rightarrow \text{MIN} = \text{DEFAULT},
\]

one can reformulate the *minimum_except_0* constraint in term of \(3 \cdot |VARIABLES| \cdot (|VARIABLES| - 1) / 2 + 2\) reified constraints.

**See also**

*hard version:* minimum (value 0 is not ignored any more).

**Keywords**

- *characteristic of a constraint:* joker value, minimum, automaton, automaton without counters, reified automaton constraint.
- *constraint arguments:* pure functional dependency.
- *constraint network structure:* centered cyclic(1) constraint network(1).
- *constraint type:* order constraint.
- *modelling:* functional dependency.
Arc input(s) VARIABLES
Arc generator $\text{CLIQUE} \rightarrow \text{collection}(\text{variables}1, \text{variables}2)$
Arc arity 2
Arc constraint(s)
• $\text{variables}1.\text{var} \neq 0$
• $\text{variables}2.\text{var} \neq 0$
• $\bigvee \left( \text{variables}1.\text{key} = \text{variables}2.\text{key}, \text{variables}1.\text{var} < \text{variables}2.\text{var} \right)$

Graph property(ies) $\text{ORDER}(0, \text{DEFAULT}, \text{var}) = \text{MIN}$

**Graph model**

Because of the first two conditions of the arc constraint, all vertices that correspond to 0 will be removed from the final graph.

Parts (A) and (B) of Figure 5.426 respectively show the initial and final graph of the second example of the Example slot. Since we use the ORDER graph property, the vertices of rank 0 (without considering the loops) of the final graph are outlined with a thick circle.

![Graph Model](image)

**Figure 5.426**: Initial and final graph of the minimum_except_0 constraint

Since the graph associated with the third example does not contain any vertex, ORDER returns the default value DEFAULT.
Automaton

Figure 5.427 depicts the automaton associated with the minimum_except_0 constraint. Let $\text{VAR}_i$ be the $i^{th}$ variable of the VARIABLES collection. To each pair (MIN, $\text{VAR}_i$) corresponds a signature variable $S_i$ as well as the following signature constraint:

$$
((\text{VAR}_i = 0) \land (\text{MIN} \neq \text{DEFAULT})) \Leftrightarrow S_i = 0 \land
((\text{VAR}_i = 0) \land (\text{MIN} = \text{DEFAULT})) \Leftrightarrow S_i = 1 \land
((\text{VAR}_i \neq 0) \land (\text{MIN} = \text{VAR}_i)) \Leftrightarrow S_i = 2 \land
((\text{VAR}_i \neq 0) \land (\text{MIN} < \text{VAR}_i)) \Leftrightarrow S_i = 3.
$$

![Automaton Image](image)

Figure 5.427: Automaton of the minimum_except_0 constraint

![Hypergraph Image](image)

Figure 5.428: Hypergraph of the reformulation corresponding to the automaton of the minimum_except_0 constraint
### 5.231 minimum_greater_than

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>N. Beldiceanu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>minimum_greater_than(VAR1, VAR2, VARIABLES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VAR1 : dvar, VAR2 : dvar, VARIABLES : collection(var–dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>VAR1 &gt; VAR2, (</td>
<td>\text{VARIABLES}</td>
<td>&gt; 0) required(VARIABLES, var)</td>
</tr>
<tr>
<td>Purpose</td>
<td>VAR1 is the smallest value strictly greater than VAR2 of the collection of variables VARIABLES: this concretely means that there exists at least one variable of VARIABLES that takes a value strictly greater than VAR2.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>(5, 3, (\langle 8, 5, 3, 8 \rangle))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The minimum_greater_than constraint holds since value 5 is the smallest value strictly greater than value 3 among values 8, 5, 3 and 8.

Typical \(|\text{VARIABLES}| > 1\) range(VARIABLES.var) > 1

Symmetry Items of VARIABLES are permutable.

Arg. properties Aggregate: VAR1(min), VAR2(id), VARIABLES(union).

Reformulation Let \(V_1, V_2, \ldots, V_{|\text{VARIABLES}|}\) denote the variables of the collection of variables VARIABLES. By creating the extra variables \(M\) and \(U_1, U_2, \ldots, U_{|\text{VARIABLES}|}\), the minimum_greater_than constraint can be expressed in term of the following constraints:

1. maximum\((M, \text{VARIABLES})\).
2. VAR1 > VAR2,
3. VAR1 \(\leq\) M,
4. \(V_i \leq \text{VAR2} \Rightarrow U_i = M\) \((i \in [1,|\text{VARIABLES}|])\),
5. \(V_i > \text{VAR2} \Rightarrow U_i = V_i\) \((i \in [1,|\text{VARIABLES}|])\),
6. minimum\((\text{VAR1}, \langle U_1, U_2, \ldots, U_{|\text{VARIABLES}|} \rangle)\).

See also common keyword: next_greater_element (order constraint).

implied by: next_greater_element.
related: next_element (identify an element in a table).
Keywords

characteristic of a constraint: minimum, automaton, automaton without counters, reified automaton constraint, derived collection.

constraint network structure: centered cyclic(2) constraint network(1).

constraint type: order constraint.
Derived Collection

\[
\text{col}(\text{ITEM} - \text{collection}(\text{var} - \text{dvar}), [\text{item} (\text{var} \rightarrow \text{VAR2})])
\]

Arc input(s) ITEM VARIABLES
Arc generator \( PRODUCT \rightarrow \text{collection} (\text{item} . \text{variables}) \)
Arc arity 2
Arc constraint(s) \( \text{item} . \text{var} < \text{variables} . \text{var} \)
Graph property(ies) \( \text{NARC} > 0 \)
Sets \( \text{SUCC} \mapsto [\text{source} . \text{variables}] \)
Constraint(s) on sets \( \text{minimum}(\text{VAR1} . \text{variables}) \)

Graph model

Similar to the next_greater_element constraint, except that there is no order on the variables of the collection VARIABLES.

Parts (A) and (B) of Figure 5.429 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold. The source and the sinks of the final graph respectively correspond to the variable VAR2 and to the variables of the VARIABLES collection that are strictly greater than VAR2. VAR1 is set to the smallest value of the var attribute of the sinks of the final graph.

![Diagram](image-url)

Figure 5.429: Initial and final graph of the minimum_greater_than constraint
Automaton

Figure 5.430 depicts the automaton associated with the minimum_greater_than constraint. Let $VAR_i$ be the $i^{th}$ variable of the VARIABLES collection. To each triple $(VAR_1, VAR_2, VAR_i)$ corresponds a signature variable $S_i$ as well as the following signature constraint:

- $((VAR_i < VAR_1) \land (VAR_i \leq VAR_2)) \iff S_i = 0 \land$
- $((VAR_i = VAR_1) \land (VAR_i \leq VAR_2)) \iff S_i = 1 \land$
- $((VAR_i > VAR_1) \land (VAR_i \leq VAR_2)) \iff S_i = 2 \land$
- $((VAR_i < VAR_1) \land (VAR_i > VAR_2)) \iff S_i = 3 \land$
- $((VAR_i = VAR_1) \land (VAR_i > VAR_2)) \iff S_i = 4 \land$
- $((VAR_i > VAR_1) \land (VAR_i > VAR_2)) \iff S_i = 5$.

The automaton is constructed in order to fulfill the following conditions:

- We look for an item of the VARIABLES collection such that $var_i = VAR_1$ and $var_i > VAR_2$.
- There should not exist any item of the VARIABLES collection such that $var_i < VAR_1$ and $var_i > VAR_2$.

Figure 5.430: Automaton of the minimum_greater_than constraint
Figure 5.431: Hypergraph of the reformulation corresponding to the automaton of the minimum_greater_than constraint
### 5.232 minimum_modulo

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from minimum.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>minimum_modulo(MIN, VARIABLES, M)</td>
<td></td>
</tr>
</tbody>
</table>
| Arguments   | MIN : dvar  
VARIABLES : collection(var−dvar)  
M : int |       |
| Restrictions| | |
| Purpose     | MIN is a minimum value of the collection of domain variables VARIABLES according to the following partial ordering: \((X \mod M) < (Y \mod M)\). |       |
| Example     | (6, (9,1,7,6,5),3)  
(9, (9,1,7,6,5),3) |       |
| Typical     | | |
| Symmetry    | Items of VARIABLES are permutable. |       |
| Arg. properties | Functional dependency: MIN determined by VARIABLES and M. |       |
| See also    | comparison swapped: maximum_modulo.  
specialisation: minimum(variable mod constant replaced by variable). |       |
| Keywords    | characteristic of a constraint: modulo, maxint, minimum.  
constraint arguments: pure functional dependency.  
constraint type: order constraint.  
modelling: functional dependency. |       |
Arc input(s)

Arc generator

Arc arity

Arc constraint(s)

Graph property(ies)

Graph model

We use a similar definition that the one that was utilised for the minimum constraint. Within the arc constraint we replace the condition \( X < Y \) by the condition \( (X \mod M) < (Y \mod M) \).

Parts (A) and (B) of Figure 5.432 respectively show the initial and final graph associated with the second example of the Example slot. Since we use the ORDER graph property, the vertex of rank 0 (without considering the loops) associated with value 9 is outlined with a thick circle.

Figure 5.432: Initial and final graph of the minimum modulo constraint
5.233  minimum_weight_alldifferent

### DESCRIPTION

<table>
<thead>
<tr>
<th>Origin</th>
<th>[158]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint</td>
<td>minimum_weight_alldifferent(VARIABLES, MATRIX, COST)</td>
</tr>
<tr>
<td>Synonyms</td>
<td>minimum_weight_alldiff, minimum_weight_alldistinct, min_weight_alldiff, min_weight_alldifferent, min_weight_alldistinct.</td>
</tr>
<tr>
<td>Arguments</td>
<td>VARIABLES : collection(var−dvar)</td>
</tr>
<tr>
<td></td>
<td>MATRIX : collection(i−int,j−int,c−int)</td>
</tr>
<tr>
<td></td>
<td>COST : dvar</td>
</tr>
</tbody>
</table>

### Purpose

All variables of the VARIABLES collection should take a distinct value located within interval [1,|VARIABLES|]. In addition COST is equal to the sum of the costs associated with the fact that we assign value i to variable j. These costs are given by the matrix MATRIX.

### Example

\[
\begin{pmatrix}
(2,3,1,4), \\
(2,1, j−1, c−4), \\
i−1, j−1, c−1, \\
i−1, j−3, c−7, \\
i−1, j−4, c−0, \\
i−2, j−1, c−1, \\
i−2, j−2, c−0, \\
i−2, j−3, c−8, \\
i−3, j−1, c−3, \\
i−3, j−2, c−2, \\
i−3, j−3, c−1, \\
i−3, j−4, c−6, \\
i−4, j−1, c−0, \\
i−4, j−2, c−0, \\
i−4, j−3, c−6, \\
i−4, j−4, c−5
\end{pmatrix}, 17
\]
The minimum_weight_alldifferent constraint holds since the cost 17 corresponds to the sum $\text{MATRIX}[(1 - 1) \cdot 4 + 2] \cdot c + \text{MATRIX}[(2 - 1) \cdot 4 + 3] \cdot c + \text{MATRIX}[(3 - 1) \cdot 4 + 1] \cdot c + \text{MATRIX}[(4 - 1) \cdot 4 + 4] \cdot c = \text{MATRIX}[2] \cdot c + \text{MATRIX}[7] \cdot c + \text{MATRIX}[9] \cdot c + \text{MATRIX}[16] \cdot c = 1 + 8 + 3 + 5$.

Typical

- $|\text{VARIABLES}| > 1$
- $\text{range} (\text{MATRIX} \cdot c) > 1$
- $\text{MATRIX} \cdot c > 0$

Arg. properties

- Functional dependency: COST determined by VARIABLES and MATRIX.

Algorithm

The Hungarian method for the assignment problem [225] can be used for evaluating the bounds of the COST variable. A filtering algorithm is described in [356]. It can be used for handling both side of the minimum_weight_alldifferent constraint:

- Evaluating a lower bound of the COST variable and pruning the variables of the VARIABLES collection in order to not exceed the maximum value of COST.
- Evaluating an upper bound of the COST variable and pruning the variables of the VARIABLES collection in order to not be under the minimum value of COST.

Systems

- all_different in SICStus, all_distinct in SICStus.

See also

- attached to cost variant: alldifferent.
- common keyword: global_cardinality_with_costs (cost filtering constraint, weighted assignment), sum_of_weights_of_distinct_values (weighted assignment), weighted_partial_alldiff (cost filtering constraint, weighted assignment).

Keywords

- application area: assignment.
- characteristic of a constraint: core.
- filtering: cost filtering constraint, Hungarian method for the assignment problem.
- final graph structure: one_succ.
- modelling: cost matrix, functional dependency.
- problems: weighted assignment.
Arc input(s) | VARIABLES
---|---
Arc generator | $CLIQUE \rightarrow \text{collection}(\text{variables1,variables2})$
Arc arity | 2
Arc constraint(s) | variables1.var = variables2.key
Graph property(ies) | • $\text{NTREE}=0$
| • $\text{SUM\_WEIGHT\_ARC} \left( \text{MATRIX} \sum_{\text{variables1.var}} \left( (\text{variables1.key} - 1) \ast |\text{VARIABLES}|, \text{variables1.var} \right) \right) . c = \text{COST}$

Graph model

Since each variable takes one value, and because of the arc constraint $\text{variables1} = \text{variables.key}$, each vertex of the initial graph belongs to the final graph and has exactly one successor. Therefore the sum of the out-degrees of the vertices of the final graph is equal to the number of vertices of the final graph. Since the sum of the in-degrees is equal to the sum of the out-degrees, it is also equal to the number of vertices of the final graph. Since $\text{NTREE} = 0$, each vertex of the final graph belongs to a circuit. Therefore each vertex of the final graph has at least one predecessor. Since we saw that the sum of the in-degrees is equal to the number of vertices of the final graph, each vertex of the final graph has exactly one predecessor. We conclude that the final graph consists of a set of vertex-disjoint elementary circuits.

Finally the graph constraint expresses that the $\text{COST}$ variable is equal to the sum of the elementary costs associated with each variable-value assignment. All these elementary costs are recorded in the $\text{MATRIX}$ collection. More precisely, the cost $c_{ij}$ is recorded in the attribute $c$ of the $((i - 1) \cdot |\text{VARIABLES}| + j)^{th}$ entry of the $\text{MATRIX}$ collection. This is ensured by the increasing restriction that enforces that the items of the $\text{MATRIX}$ collection are sorted in lexicographically increasing order according to attributes $i$ and $j$.

![Figure 5.433: Initial and final graph of the minimum_weight_alldifferent constraint](image)

Parts (A) and (B) of Figure 5.433 respectively show the initial and final graph associated with the **Example** slot. Since we use the $\text{SUM\_WEIGHT\_ARC}$ graph property, the
arcs of the final graph are stressed in bold. We also indicate their corresponding weight.
### 5.234 multi_global_contiguity

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from <code>global_contiguity</code></td>
</tr>
<tr>
<td>Constraint</td>
<td><code>multi_global_contiguity(VARIABLES)</code></td>
</tr>
<tr>
<td>Synonym</td>
<td><code>multi_contiguity</code></td>
</tr>
<tr>
<td>Argument</td>
<td><code>VARIABLES : collection(var−dvar)</code></td>
</tr>
</tbody>
</table>
| Restrictions| `required(VARIABLES, var)`  
              `VARIABLES.var ≥ 0` |
| Purpose     | Enforce all variables of the `VARIABLES` collection to be assigned a value greater than or equal to 0. In addition, each value $v$ strictly greater than 0 should appear contiguously. |
| Example     | The `multi_global_contiguity` constraint holds since the sequence $0 \ 2 \ 2 \ 1 \ 1 \ 0 \ 0 \ 5$ contains no more than one group of contiguous 1, no more than one group of contiguous 2, and no more than one group of contiguous 5. |
| Typical     | $|VARIABLES| > 2$  
              `range(VARIABLES.var) > 2` |
| Symmetry    | Items of `VARIABLES` can be reversed. |
| Arg. properties | Contractible wrt. `VARIABLES`. |
| See also    | common keyword: `group(sequence)`.  
              implied by: `global_contiguity`. |
| Keywords    | combinatorial object: sequence.  
              constraint type: predefined constraint. |
5.235 multi_inter_distance

**DESCRIPTION**

**SYNONYMS**

multi_all_min_distance, multi_all_min_dist, sliding_atmost, atmost_sliding.

**ARGUMENTS**

- VARIABLES: collection(var−dvar)
- LIMIT: int
- DIST: int

**RESTRICTIONS**

required(VARIABLES, var)
- LIMIT > 0
- DIST > 0

**PURPOSE**

Enforce that at most LIMIT variables of the collection VARIABLES are assigned values in any set consisting of DIST consecutive integer values.

**EXAMPLE**

\[(4, 0, 9, 4, 7) \rightarrow 2, 3\]

The multi_inter_distance constraint holds since, for each set of DIST = 3 consecutive values, no more than LIMIT = 2 variables of the VARIABLES collection \((4, 0, 9, 4, 7)\) are assigned a value from that set:

- At most two, in fact one, variables of the VARIABLES collection are assigned a value from the set \(\{0, 1, 2\}\).
- At most two, in fact zero, variables of the VARIABLES collection are assigned a value from the set \(\{1, 2, 3\}\).
- At most two, in fact two, variables of the VARIABLES collection are assigned a value from the set \(\{2, 3, 4\}\).
- At most two, in fact two, variables of the VARIABLES collection are assigned a value from the set \(\{3, 4, 5\}\).
- At most two, in fact two, variables of the VARIABLES collection are assigned a value from the set \(\{4, 5, 6\}\).
- At most two, in fact one, variables of the VARIABLES collection are assigned a value from the set \(\{5, 6, 7\}\).
- At most two, in fact one, variables of the VARIABLES collection are assigned a value from the set \(\{6, 7, 8\}\).
- At most two, in fact two, variables of the VARIABLES collection are assigned a value from the set \(\{7, 8, 9\}\).
Typical

<table>
<thead>
<tr>
<th>( \text{LIMIT} &gt; 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{LIMIT} &lt;</td>
</tr>
<tr>
<td>( \text{DIST} &gt; 1 )</td>
</tr>
<tr>
<td>( \text{DIST} &lt; \text{range}(\text{VARIABLES.var}) )</td>
</tr>
</tbody>
</table>

Symmetries

- Items of \( \text{VARIABLES} \) are permutable.
- One and the same constant can be added to the \( \text{var} \) attribute of all items of \( \text{VARIABLES} \).
- \( \text{LIMIT} \) can be increased.
- \( \text{MINDIST} \) can be decreased to any value \( \geq 1 \).

Arg. properties

Contractible wrt. \( \text{VARIABLES} \).

Usage

The multi_inter_distance constraint was tested for scheduling tasks that all have the same fixed duration in the context of air traffic management.

Algorithm

P. Ouellet and C.-G. Quimper came up with a cubic time complexity algorithm achieving bound-consistency in [282].

See also

- generalisation: cumulative(line segment, of same length, replaced by line segment).
- specialisation: all_min_dist(LIMIT parameter set to 1), cardinality_atmost(window of DIST consecutive values replaced by window of size 1).

Keywords

- application area: air traffic management.
- constraint type: predefined constraint, value constraint, scheduling constraint.
- filtering: bound-consistency.
- modelling: at most.
5.236 nand

**DESCRIPTION**

Origin: Logic

Constraint: \( \text{nand}(\text{VAR}, \text{VARIABLES}) \)

Synonym: clause.

Arguments:
- \( \text{VAR} : \text{dvar} \)
- \( \text{VARIABLES} : \text{collection(\text{var} - \text{dvar})} \)

Restrictions:
- \( \text{VAR} \geq 0 \)
- \( \text{VAR} \leq 1 \)
- \( |\text{VARIABLES}| \geq 2 \)
- \( \text{required}(\text{VARIABLES}, \text{var}) \)
- \( \text{VARIABLES}.\text{var} \geq 0 \)
- \( \text{VARIABLES}.\text{var} \leq 1 \)

Purpose:
Let \( \text{VARIABLES} \) be a collection of 0-1 variables \( \text{VAR}_1, \text{VAR}_2, \ldots, \text{VAR}_n \) \( (n \geq 2) \). Enforce \( \text{VAR} = \neg(\text{VAR}_1 \land \text{VAR}_2 \land \ldots \land \text{VAR}_n) \).

Example:
- \((1, (0, 0))\)
- \((1, (0, 1))\)
- \((1, (1, 0))\)
- \((0, (1, 1))\)
- \((1, (1, 0, 1))\)

Symmetry:
Items of \( \text{VARIABLES} \) are permutable.

**Arg. properties**

- **Functional dependency**: \( \text{VAR} \) determined by \( \text{VARIABLES} \).
- **Contractible** wrt. \( \text{VARIABLES} \) when \( \text{VAR} = 0 \).
- **Extensible** wrt. \( \text{VARIABLES} \) when \( \text{VAR} = 1 \).
- **Aggregate**: \( \text{VAR}(\lor), \text{VARIABLES}(\text{union}) \).

**Systems**

clause in Choco, clause in Gecode, \#/\ in SICStus.

See also:
**common keyword**: and, equivalent, imply, nor, or, xor (Boolean constraint).

implies: atleast_nvalue.
Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.
constraint arguments: pure functional dependency.
constraint network structure: Berge-acyclic constraint network.
constraint type: Boolean constraint.
filtering: arc-consistency.
modelling: functional dependency.
Automaton

Figure 5.434 depicts the automaton associated with the nand constraint. To the first argument VAR of the nand constraint corresponds the first signature variable. To each variable VAR\textsubscript{i} of the second argument VARIABLES of the nand constraint corresponds the next signature variable. There is no signature constraint.

Figure 5.434: Automaton of the nand constraint

Figure 5.435: Hypergraph of the reformulation corresponding to the automaton of the nand constraint
### 5.237 nclass

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Derived from nvalue.</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>nclass(NCLASS, VARIABLES, PARTITIONS)</td>
<td></td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>VALUES : collection(val−int)</td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>NCLASS : dvar</td>
<td>VARIABLES : collection(var−dvar)</td>
</tr>
<tr>
<td></td>
<td>PARTITIONS : collection(p − VALUES)</td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>Number of partitions of the collection PARTITIONS such that at least one value is assigned to at least one variable of the collection VARIABLES.</td>
<td></td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Symmetries

- Items of VARIABLES are permutable.
- Items of PARTITIONS are permutable.
- Items of PARTITIONS.p are permutable.
- An occurrence of a value of VARIABLES.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.
- All occurrences of two distinct tuples of values in VARIABLES.var or PARTITIONS.p.val can be swapped; all occurrences of a tuple of values in VARIABLES.var or PARTITIONS.p.val can be renamed to any unused tuple of values.

Arg. properties

- Functional dependency: NCLASS determined by VARIABLES and PARTITIONS.
- Extensible wrt. VARIABLES when NCLASS = |PARTITIONS|.

Algorithm

[26, 38].

See also

related: nequivalence (variable ∈ partition replaced by variable mod constant), ninterval (variable ∈ partition replaced by variable/constant), npair (variable ∈ partition replaced by pair of variables).

specialisation: nvalue (variable ∈ partition replaced by variable).

used in graph description: in_same_partition.

Keywords

characteristic of a constraint: partition.
constraint arguments: pure functional dependency.
constraint type: counting constraint, value partitioning constraint.
final graph structure: strongly connected component, equivalence.
modelling: number of distinct equivalence classes, functional dependency.
Arc input(s)  VARIABLES
Arc generator  \( CLIQUE \rightarrow \text{collection}(\text{variables1,variables2}) \)
Arc arity  2
Arc constraint(s)  \( \text{in\_same\_partition}(\text{variables1.var,variables2.var,PARTITIONS}) \)
Graph property(ies)  \( \text{NSCC} = \text{NCLASS} \)

Graph model

Parts (A) and (B) of Figure 5.436 respectively show the initial and final graph associated with the Example slot. Since we use the NSCC graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a class of values that was assigned to some variables of the VARIABLES collection. We effectively use two classes of values that respectively correspond to values \{3\} and \{2, 6\}. Note that we do not consider value 7 since it does not belong to the different classes of values we gave: all corresponding arc constraints do not hold.

![Graph](image)

Figure 5.436: Initial and final graph of the nclass constraint
neq

5.238 neq

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Arithmetic.</td>
</tr>
<tr>
<td>Constraint</td>
<td>neq(VAR1, VAR2)</td>
</tr>
<tr>
<td>Synonym</td>
<td>rel.</td>
</tr>
<tr>
<td>Arguments</td>
<td>VAR1 : dvar</td>
</tr>
<tr>
<td></td>
<td>VAR2 : dvar</td>
</tr>
<tr>
<td>Purpose</td>
<td>Enforce the fact that two variables are not equal.</td>
</tr>
<tr>
<td>Example</td>
<td>(1, 8)</td>
</tr>
<tr>
<td></td>
<td>The neq constraint holds since 1 is not equal to 8.</td>
</tr>
<tr>
<td>Symmetries</td>
<td>- Arguments are permutable w.r.t. permutation (VAR1, VAR2).</td>
</tr>
<tr>
<td></td>
<td>- A value in VAR1 or VAR2 can be renamed to any unused value.</td>
</tr>
<tr>
<td>Systems</td>
<td>neq in Choco, rel in Gecode, #= in SICStus.</td>
</tr>
<tr>
<td>See also</td>
<td>common keyword: geq, leq (binary constraint, arithmetic constraint).</td>
</tr>
<tr>
<td></td>
<td>generalisation: neq_cst (constant added), not_all_equal.</td>
</tr>
<tr>
<td></td>
<td>implied by: gt, lt.</td>
</tr>
<tr>
<td></td>
<td>negation: eq.</td>
</tr>
<tr>
<td></td>
<td>system of constraints: alldifferent.</td>
</tr>
<tr>
<td>Keywords</td>
<td>constraint arguments: binary constraint.</td>
</tr>
<tr>
<td></td>
<td>constraint type: predefined constraint, arithmetic constraint.</td>
</tr>
<tr>
<td></td>
<td>filtering: arc-consistency.</td>
</tr>
</tbody>
</table>
### 5.239  neq_cst

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Arithmetic.</td>
</tr>
<tr>
<td>Constraint</td>
<td><code>neq_cst(VAR1, VAR2, CST)</code></td>
</tr>
</tbody>
</table>
| Arguments   | \(\text{VAR1}: \text{dvar}\)  \\
|             | \(\text{VAR2}: \text{dvar}\)  \\
|             | \(\text{CST2}: \text{int}\) |
| Purpose     | Enforce the fact that the first variable is different from the sum of the second variable and the constant. |
| Example     | \((8, 2, 7)\)  \\
|             | The \(\text{neq_cst}\) constraint holds since 8 is different from 2 + 7. |
| Typical     | \(\text{CST2} \neq 0\)  \\
|             | \(\text{VAR1} \neq \text{VAR2} + \text{CST2}\) |
| Symmetries  | • Arguments are \textit{permutable} w.r.t. permutation \((\text{VAR1}) (\text{VAR2}, \text{CST2})\).  \\
|             | • One and the same constant can be \textit{added} to \text{VAR1} and \text{VAR2}.  \\
|             | • One and the same constant can be \textit{added} to \text{VAR1} and \text{CST2}. |
| See also    | \textit{negation}: \textit{eq_cst}.  \\
|             | \textit{specialisation}: \textit{neq(constant removed)}. |
| Keywords    | \textit{characteristic of a constraint}: disequality.  \\
|             | \textit{constraint arguments}: binary constraint.  \\
|             | \textit{constraint type}: predefined constraint, arithmetic constraint.  \\
|             | \textit{filtering}: arc-consistency. |
5.240 nequivalence

### Description

<table>
<thead>
<tr>
<th></th>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from nvalue.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>nequivalence(NEQUIV, M, VARIABLES)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Arguments | NEQUIV : dvar  
M : int  
VARIABLES : collection(var−dvar) | | |
| Restrictions | required(VARIABLES, var)  
NEQUIV ≥ min(1, |VARIABLES|)  
NEQUIV ≤ min(M, |VARIABLES|)  
NEQUIV ≤ range(VARIABLES, var)  
M > 0 | | |
| Purpose | NEQUIV is the number of distinct rests obtained by dividing the variables of the collection VARIABLES by M. | | |

#### Example

\[
\begin{pmatrix}
\text{var − 3,} \\
\text{var − 2,} \\
\text{var − 5,} \\
2,3, \\
\text{var − 6,} \\
\text{var − 15,} \\
\text{var − 3,} \\
\text{var − 3}
\end{pmatrix}
\]

Since the expressions 3 mod 3 = 0, 2 mod 3 = 2, 5 mod 3 = 2, 6 mod 3 = 0, 15 mod 3 = 0, 3 mod 3 = 0, and 3 mod 3 = 0 involve two distinct values (values 0 and 2), the first argument NEQUIV of the nequivalence constraint is set to value 2.

#### Typical

- NEQUIV > 1
- NEQUIV < |VARIABLES|
- NEQUIV < range(VARIABLES, var)
- M > 1
- M < maxval(VARIABLES, var)

#### Symmetries

- Items of VARIABLES are permutable.
- An occurrence of a value \( u \) of VARIABLES.var can be replaced by any other value \( v \) such that \( v \) is congruent to \( u \) modulo \( M \).

#### Arg. properties

- Functional dependency: NEQUIV determined by \( M \) and VARIABLES.
- Contractible wrt. VARIABLES when NEQUIV = 1 and |VARIABLES| > 0.
- Contractible wrt. VARIABLES when NEQUIV = |VARIABLES|.
- Extensible wrt. VARIABLES when NEQUIV = \( M \).
Algorithm

Since constraints $X = Y$ and $X \equiv Y \pmod{M}$ are similar, one should also use a similar algorithm as the one [26, 38] provided for constraint nvalue.

See also

related: nclass (variable mod constant replaced by variable ∈ partition), ninterval (variable mod constant replaced by variable/constant), npair (variable mod constant replaced by pair of variables).

specialisation: nvalue (variable mod constant replaced by variable).

Keywords

constraint arguments: pure functional dependency.

constraint type: counting constraint, value partitioning constraint.

final graph structure: strongly connected component, equivalence.

modelling: number of distinct equivalence classes, functional dependency.
Arc input(s) VARIABLES
Arc generator $\text{CLIQUE} \mapsto \text{collection}(\text{variables}_1, \text{variables}_2)$
Arc arity 2
Arc constraint(s) $\text{variables}_1.\text{var} \mod \text{M} = \text{variables}_2.\text{var} \mod \text{M}$
Graph property(ies) \text{NSCC} = \text{NEQUIV}

Graph model
Parts (A) and (B) of Figure 5.437 respectively show the initial and final graph associated with the Example slot. Since we use the NSCC graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to one equivalence class: We have two equivalence classes that respectively correspond to values \{3, 6, 15\} and \{2, 5\}.

Figure 5.437: Initial and final graph of the nequivalence constraint
5.241 next_element

**DESCRIPTION**

**LINKS**

**GRAPH**

**AUTOMATON**

**Origin**

N. Beldiceanu

**Constraint**

next_element(THRESHOLD, INDEX, TABLE, VAL)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>THRESHOLD</td>
<td>dvar</td>
</tr>
<tr>
<td>INDEX</td>
<td>dvar</td>
</tr>
<tr>
<td>TABLE</td>
<td>collection(index=int,value=dvar)</td>
</tr>
<tr>
<td>VAL</td>
<td>dvar</td>
</tr>
</tbody>
</table>

**Restrictions**

INDEX \( \geq 1 \)

INDEX \( \leq |TABLE| \)

THRESHOLD \(< INDEX \)

\( \text{required}(TABLE, [index, value]) \)

|TABLE| \( \geq 0 \)

TABLE.index \( \geq 1 \)

TABLE.index \( \leq |TABLE| \)

distinct(TABLE,index)

**Purpose**

INDEX is the smallest entry of TABLE strictly greater than THRESHOLD containing value VAL.

**Example**

\[
\begin{pmatrix}
\text{index} - 1 & \text{value} - 1, \\
\text{index} - 2 & \text{value} - 8, \\
\text{index} - 3 & \text{value} - 9, \\
2, 3, & , 9 \\
\text{index} - 4 & \text{value} - 5, \\
\text{index} - 5 & \text{value} - 9
\end{pmatrix}
\]

The next_element constraint holds since 3 is the smallest entry located after entry 2 that contains value 9.

**Typical**

|TABLE| \( \geq 1 \)

range(TABLE.value) \( > 1 \)

**Usage**

Originally introduced for modelling the fact that a nucleotide has to be consumed as soon as possible at cycle INDEX after a given cycle represented by variable THRESHOLD.

**See also**

related: minimum_greater_than (identify an element in a table), next_greater_element (allow to iterate over the values of a table).

**Keywords**

characteristic of a constraint: minimum, automaton, automaton without counters, reified automaton constraint, derived collection.

constraint network structure: centered cyclic(3) constraint network(1).

constraint type: data constraint.

modelling: table.
Derived Collection

\[
\text{col}(\text{ITEM-collection(index-dvar, value-dvar),})
\]

\[
[\text{item(index - THRESHOLD, value - VAL)}]
\]

Arc input(s) ITEM TABLE
Arc generator  \( \text{PRODUCT} \rightarrow \text{collection(item.table)} \)
Arc arity 2
Arc constraint(s)
- item.index < table.index
- item.value = table.value
Graph property(ies) NARC > 0
Sets
- SUCC \( \rightarrow \)
  \[
  \text{variables} \leftarrow \text{col}(\text{VARIABLES-collection(var-dvar),})
  \]
  \[
  [\text{item(var - TABLE.index)}]
  \]
Constraint(s) on sets \( \text{minimum(INDEX.variables)} \)
Graph model

Parts (A) and (B) of Figure 5.438 respectively show the initial and final graph associated with the second graph constraint of the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

![Graph model](image)

Figure 5.438: Initial and final graph of the next_element constraint
Automaton

Figure 5.439 depicts the automaton associated with the next_element constraint. Let $I_k$ and $V_k$ respectively be the index and the value attributes of the $k^{th}$ item of the TABLE collections. To each quintuple $(\text{THRESHOLD, INDEX, VAL, } I_k, V_k)$ corresponds a signature variable $S_k$ as well as the following signature constraint:

\[
((I_k \leq \text{THRESHOLD}) \land (I_k < \text{INDEX}) \land (V_k = \text{VAL})) \Leftrightarrow S_k = 0 \land
((I_k \leq \text{THRESHOLD}) \land (I_k < \text{INDEX}) \land (V_k \neq \text{VAL})) \Leftrightarrow S_k = 1 \land
((I_k \leq \text{THRESHOLD}) \land (I_k = \text{INDEX}) \land (V_k = \text{VAL})) \Leftrightarrow S_k = 2 \land
((I_k \leq \text{THRESHOLD}) \land (I_k = \text{INDEX}) \land (V_k \neq \text{VAL})) \Leftrightarrow S_k = 3 \land
((I_k \leq \text{THRESHOLD}) \land (I_k > \text{INDEX}) \land (V_k = \text{VAL})) \Leftrightarrow S_k = 4 \land
((I_k \leq \text{THRESHOLD}) \land (I_k > \text{INDEX}) \land (V_k \neq \text{VAL})) \Leftrightarrow S_k = 5 \land
((I_k > \text{THRESHOLD}) \land (I_k < \text{INDEX}) \land (V_k = \text{VAL})) \Leftrightarrow S_k = 6 \land
((I_k > \text{THRESHOLD}) \land (I_k < \text{INDEX}) \land (V_k \neq \text{VAL})) \Leftrightarrow S_k = 7 \land
((I_k > \text{THRESHOLD}) \land (I_k = \text{INDEX}) \land (V_k = \text{VAL})) \Leftrightarrow S_k = 8 \land
((I_k > \text{THRESHOLD}) \land (I_k = \text{INDEX}) \land (V_k \neq \text{VAL})) \Leftrightarrow S_k = 9 \land
((I_k > \text{THRESHOLD}) \land (I_k > \text{INDEX}) \land (V_k = \text{VAL})) \Leftrightarrow S_k = 10 \land
((I_k > \text{THRESHOLD}) \land (I_k > \text{INDEX}) \land (V_k \neq \text{VAL})) \Leftrightarrow S_k = 11.
\]

The automaton is constructed in order to fulfil the following conditions:

- We look for an item of the TABLE collection such that $\text{INDEX}_i > \text{THRESHOLD}$ and $\text{INDEX}_i = \text{INDEX}$ and $\text{VALUE}_i = \text{VAL}$.
- There should not exist any item of the TABLE collection such that $\text{INDEX}_i > \text{THRESHOLD}$ and $\text{INDEX}_i < \text{INDEX}$ and $\text{VALUE}_i = \text{VAL}$.
Figure 5.439: Automaton of the `next_element` constraint
Figure 5.440: Hypergraph of the reformulation corresponding to the automaton of the next element constraint
5.242 next_greater_element

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

next_greater_element(VAR1, VAR2, VARIABLES)

<table>
<thead>
<tr>
<th>Arguments</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>VAR1</td>
<td>dvar</td>
<td></td>
</tr>
<tr>
<td>VAR2</td>
<td>dvar</td>
<td></td>
</tr>
<tr>
<td>VARIABLES</td>
<td>collection(var−dvar)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Restrictions</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>VAR1 &lt; VAR2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES</td>
<td>&gt; 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Purpose</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>VAR2 is the value strictly greater than VAR1 located at the smallest possible entry of the table TABLE. In addition, the variables of the collection VARIABLES are sorted in strictly increasing order.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(7, 8, ⟨3, 5, 8, 9⟩)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The next_greater_element constraint holds since:

- VAR2 is fixed to the first value 8 strictly greater than VAR1 = 7,
- The var attributes of the items of the collection VARIABLES are sorted in strictly increasing order.

<table>
<thead>
<tr>
<th>Typical</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VARIABLES</td>
<td>&gt; 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usage</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Originally introduced in [92] for modelling the fact that a nucleotide has to be consumed as soon as possible at cycle VAR2 after a given cycle VAR1.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Remark</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Similar to the minimum_greater_than constraint, except for the fact that the var attributes are sorted.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reformulation</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Let $V_1, V_2, \ldots, V_{</td>
<td>\text{VARIABLES}</td>
<td>}$ denote the variables of the collection of variables VARIABLES. By creating the extra variables $M$ and $U_1, U_2, \ldots, U_{</td>
</tr>
</tbody>
</table>

1. $V_1 < V_2 < \ldots < V_{|\text{VARIABLES}|}$
2. $\text{maximum}(M, \text{VARIABLES})$.
3. $\text{VAR2} > \text{VAR1}$.
4. $\text{VAR2} \leq M$.
5. $V_i \leq \text{VAR1} \implies U_i = M \ (i \in [1,|\text{VARIABLES}|])$. |
6. $V_i > \text{VAR1} \Rightarrow U_i = V_i \ (i \in [1, |\text{VARIABLES}|])$.
7. $\text{minimum}(\text{VAR2}, \langle U_1, U_2, \ldots, U_{|\text{VARIABLES}|} \rangle)$.

**See also**

- **common keyword**: `minimum_greater_than` *(order constraint)*.
- **implies**: `minimum_greater_than`.
- **related**: `next_element` *(allow to iterate over the values of a table)*.

**Keywords**

- **characteristic of a constraint**: `minimum`, `derived collection`.
- **constraint type**: `order constraint`, `data constraint`.
- **modelling**: `table`.
Derived Collection

\[ \text{col}(V \rightarrow \text{collection}(\text{var} \rightarrow \text{dvar}), [\text{item}(	ext{var} \rightarrow \text{VAR1})]) \]

Arc input(s) | VARIABLES
---|---
Arc generator | \( \text{PATH} \rightarrow \text{collection} (\text{variables1}, \text{variables2}) \)
Arc arity | 2
Arc constraint(s) | \text{variables1.var} < \text{variables2.var}
Graph property(ies) | \( \text{NARC} = |\text{VARIABLES}| - 1 \)

Arc input(s) | V VARIABLES
---|---
Arc generator | \( \text{PRODUCT} \rightarrow \text{collection}(v, \text{variables}) \)
Arc arity | 2
Arc constraint(s) | \( v.\text{var} < \text{variables.var} \)
Graph property(ies) | \( \text{NARC} > 0 \)
Sets | \text{SUCC} \rightarrow [\text{source}, \text{variables}]
Constraint(s) on sets | \text{minimum}(\text{VAR2}, \text{variables})

Graph model

Parts (A) and (B) of Figure 5.441 respectively show the initial and final graph associated with the second graph constraint of the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

![Graph model diagram]

Figure 5.441: Initial and final graph of the next_greater_element constraint

Signature

Since the first graph constraint uses the \( \text{PATH} \) arc generator on the VARIABLES collection, the number of arcs of the corresponding initial graph is equal to \( |\text{VARIABLES}| - 1 \). Therefore the maximum number of arcs of the final graph is equal to \( |\text{VARIABLES}| - 1 \). For this reason we can rewrite \( \text{NARC} = |\text{VARIABLES}|-1 \) to \( \text{NARC} \geq |\text{VARIABLES}|-1 \) and simplify \( \text{NARC} \) to \( \text{NARC} \).
5.243 ninterval

**DESCRIPTION**

<table>
<thead>
<tr>
<th>Origin</th>
<th>Derived from nvalue.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint</td>
<td>ninterval(NVAL, VARIABLES, SIZE_INTERVAL)</td>
</tr>
<tr>
<td>Arguments</td>
<td>NVAL : dvar</td>
</tr>
<tr>
<td></td>
<td>VARIABLES : collection(var−dvar)</td>
</tr>
<tr>
<td></td>
<td>SIZE_INTERVAL : int</td>
</tr>
<tr>
<td>Restrictions</td>
<td>NVAL ≥ min(1,</td>
</tr>
<tr>
<td></td>
<td>NVAL ≤</td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES, var)</td>
</tr>
<tr>
<td></td>
<td>SIZE_INTERVAL &gt; 0</td>
</tr>
<tr>
<td>Purpose</td>
<td>Consider the intervals of the form [SIZE_INTERVAL · k, SIZE_INTERVAL · k + SIZE_INTERVAL − 1] where k is an integer. NVAL is the number of intervals for which at least one value is assigned to at least one variable of the collection VARIABLES.</td>
</tr>
<tr>
<td>Example</td>
<td>(2, (3, 1, 9, 1, 9), 4)</td>
</tr>
</tbody>
</table>

In the example, the third argument SIZE_INTERVAL = 4 defines the following family of intervals [4 · k, 4 · k + 3], where k is an integer. Values 3, 1, 9, 1 and 9 are respectively located within intervals [0, 3], [0, 3], [8, 11], [0, 3] and [8, 11]. Since we only use the two intervals [0, 3] and [8, 11] the first argument of the ninterval constraint is set to value 2.

| Typical | NVAL > 1 |
| | NVAL < |VARIABLES| |
| | SIZE_INTERVAL > 1 |
| | SIZE_INTERVAL <range(VARIABLES.var) |

| Symmetries | Items of VARIABLES are permutable. |
| | An occurrence of a value of VARIABLES.var that belongs to the k-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval. |

| Arg. properties | Functional dependency: NVAL determined by VARIABLES and SIZE_INTERVAL. |
| | Contractible wrt. VARIABLES when NVAL = 1 and |VARIABLES| > 0. |
| | Contractible wrt. VARIABLES when NVAL = |VARIABLES|. |

| Usage | The ninterval constraint is useful for counting the number of effectively used periods, no matter how many time each period is used. A period can for example stand for a hour or for a day. |
Algorithm

See also 

Keywords 

constraint arguments: pure functional dependency.
constraint type: counting constraint, value partitioning constraint.
final graph structure: strongly connected component, equivalence.
modelling: number of distinct equivalence classes, interval, functional dependency.
Arc input(s) | VARIABLES
---|---
Arc generator | $\text{CLIQUE} \rightarrow \text{collection}(\text{variables1, variables2})$
Arc arity | 2
Arc constraint(s) | variables1.var/\text{SIZE\_INTERVAL} = \text{variables2.var/\text{SIZE\_INTERVAL}}
Graph property(ies) | \text{NSCC} = \text{NVAL}

Graph model

Parts (A) and (B) of Figure 5.442 respectively show the initial and final graph associated with the Example slot. Since we use the NSCC graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to those values of an interval that are assigned to some variables of the VARIABLES collection. The values 1, 3 and the value 9, which respectively correspond to intervals [0, 3] and [8, 11], are assigned to the variables of the VARIABLES collection.

![Figure 5.442: Initial and final graph of the ninterval constraint](image)

(A) \hspace{1cm} (B)
### 5.244 no_peak

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from peak.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>no_peak(VARIABLES)</td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>VARIABLES : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>A variable ( V_k ) (1 &lt; k &lt; m) of the sequence of variables ( VARIABLES = V_1, \ldots, V_m ) is a peak if and only if there exists an ( i ) (1 &lt; i ≤ k) such that ( V_{i−1} &lt; V_i ) and ( V_i = V_{i+1} = \ldots = V_k ) and ( V_k &gt; V_{k+1} ). The total number of peaks of the sequence of variables ( VARIABLES ) is equal to 0.</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>( ((1,1,4,8,8)) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The no_peak constraint holds since the sequence 1 1 4 8 8 does not contain any peak.</td>
<td></td>
</tr>
</tbody>
</table>

![Figure 5.443: A sequence without any peak](image)

**Typical**

\[ |VARIABLES| > 2 \]

\[ \text{range}(VARIABLES.\text{var}) > 1 \]

**Symmetries**

- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.
Arg. properties

Contractible wrt. VARIABLES.

See also

comparison swapped: no_valley.
generalisation: peak (introduce a variable counting the number of peaks).
implicated by: decreasing, increasing.
related: valley.

Keywords

characteristic of a constraint: automaton, automaton without counters,
reified automaton constraint.
combinatorial object: sequence.
constraint network structure: sliding cyclic(1) constraint network(1).
Automaton

Figure 5.444 depicts the automaton associated with the no_peak constraint. To each pair of consecutive variables \((\text{VAR}_i, \text{VAR}_{i+1})\) of the collection VARIABLES corresponds a signature variable \(S_i\). The following signature constraint links \(\text{VAR}_i\), \(\text{VAR}_{i+1}\) and \(S_i\): 

\[
\begin{align*}
\text{VAR}_i < \text{VAR}_{i+1} & \iff S_i = 0 \\
\text{VAR}_i = \text{VAR}_{i+1} & \iff S_i = 1 \\
\text{VAR}_i > \text{VAR}_{i+1} & \iff S_i = 2
\end{align*}
\]

Figure 5.444: Automaton of the no_peak constraint

Figure 5.445: Hypergraph of the reformulation corresponding to the automaton of the no_peak constraint
5.245 no_valley

**DESCRIPTION**

Origin

Derived from valley.

Constraint

no_valley(VARIABLES)

Argument

VARIABLES : collection(var–dvar)

Restrictions

| VARIABLES | > 0  
|-----------|------  
| required(VARIABLES, var) |

**Purpose**

A variable $V_k$ ($1 < k < m$) of the sequence of variables VARIABLES = $V_1, \ldots, V_m$ is a valley if and only if there exists an $i$ ($1 < i \leq k$) such that $V_{i-1} > V_i$ and $V_i = V_{i+1} = \ldots = V_k$ and $V_k < V_{k+1}$. The total number of valleys of the sequence of variables VARIABLES is equal to 0.

**Example**

$$\left\{ \begin{array}{l} var - 1, \\ var - 1, \\ var - 4, \\ var - 8, \\ var - 8, \\ var - 2 \end{array} \right.$$ 

The no_valley constraint holds since the sequence 1 1 4 8 8 2 does not contain any valley.

![Figure 5.446: A sequence without any valley](image-url)

**Typical**

| VARIABLES | > 2  
|-----------|------  
| range(VARIABLES, var) | > 1 |
Symmetries

- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

Contractible wrt. VARIABLES.

See also

comparison swapped: no_peak.
generalisation: valley (introduce a variable counting the number of valleys).
implied by: decreasing, global_contiguity, increasing.
related: peak.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint,
combinatorial object: sequence.
constraint network structure: sliding cyclic(1) constraint network(1).
Automaton

Figure 5.447 depicts the automaton associated with the no_valley constraint. To each pair of consecutive variables \((\text{VAR}_i, \text{VAR}_{i+1})\) of the collection VARIABLES corresponds a signature variable \(S_i\). The following signature constraint links \(\text{VAR}_i\), \(\text{VAR}_{i+1}\) and \(S_i\): \((\text{VAR}_i < \text{VAR}_{i+1} \Leftrightarrow S_i = 0) \land (\text{VAR}_i = \text{VAR}_{i+1} \Leftrightarrow S_i = 1) \land (\text{VAR}_i > \text{VAR}_{i+1} \Leftrightarrow S_i = 2)\).

\[
\begin{align*}
\text{VAR}_i &= \text{VAR}_{i+1} \\
\text{VAR}_i < \text{VAR}_{i+1} &
\end{align*}
\]

\[
\begin{align*}
\text{VAR}_i &= \text{VAR}_{i+1} \\
\text{VAR}_i > \text{VAR}_{i+1} &
\end{align*}
\]

Figure 5.447: Automaton of the no_valley constraint

Figure 5.448: Hypergraph of the reformulation corresponding to the automaton of the no_valley constraint
### 5.246 non_overlap_sboxes

<table>
<thead>
<tr>
<th>Description</th>
<th>Links</th>
<th>Logic</th>
</tr>
</thead>
</table>

**Origin**
Geometry, derived from [36]

**Constraint**
non_overlap_sboxes(K, DIMS, OBJECTS, SBOXES)

**Synonyms**
on_nonoverlap, non_overlapping.

**Types**
- **VARIABLES**: collection(v−dvar)
- **INTEGERS**: collection(v−int)
- **POSITIVES**: collection(v−int)

**Arguments**
- K : int
- DIMS : sint
- OBJECTS : collection(oid−int,sid−int,x − VARIABLES)
- SBOXES : collection(sid−int,t − INTEGERS,l − POSITIVES)

**Restrictions**
- \(|\text{VARIABLES}| \geq 1\)
- \(|\text{INTEGERS}| \geq 1\)
- \(|\text{POSITIVES}| \geq 1\)
- required(VARIABLES, v)
- \(|\text{VARIABLES}| = K\)
- required(INTEGERS, v)
- \(|\text{INTEGERS}| = K\)
- required(POSITIVES, v)
- \(|\text{POSITIVES}| = K\)
- POSITIVES.v > 0
- K > 0
- DIMS ≥ 0
- DIMS < K
- increasing_seq(OBJECTS, [oid])
- required(OBJECTS, [oid,sid,x])
- OBJECTS.oid ≥ 1
- OBJECTS.oid ≤ |OBJECTS|
- OBJECTS.sid ≥ 1
- OBJECTS.sid ≤ |OBJECTS|
- required(SBOXES, [sid,t,l])
- SBOXES.sid ≥ 1
- SBOXES.sid ≤ |SBOXES|
Holds if, for each pair of objects \((O_i, O_j)\), \(i < j\), \(O_i\) and \(O_j\) do not overlap with respect to a set of dimensions depicted by \(\text{DIMS}\). \(O_i\) and \(O_j\) are objects that take a shape among a set of shapes. Each shape is defined as a finite set of shifted boxes, where each shifted box is described by a box in a \(K\)-dimensional space at a given offset (from the origin of the shape) with given sizes. More precisely, a shifted box is an entity defined by its shape id \(\text{sid}\), shift offset \(t\), and sizes \(l\). Then, a shape is defined as the union of shifted boxes sharing the same shape id. An object is an entity defined by its unique object identifier \(\text{oid}\), shape id \(\text{sid}\) and origin \(x\).

An object \(O_i\) does not overlap an object \(O_j\) with respect to a set of dimensions depicted by \(\text{DIMS}\) if and only if, for all shifted box \(s_i\) associated with \(O_i\) and for all shifted box \(s_j\) associated with \(O_j\), there exists a dimension \(d \in \text{DIMS}\) such that the start of \(s_i\) in dimension \(d\) is greater than or equal to the end of \(s_j\) in dimension \(d\), or the start of \(s_j\) in dimension \(d\) is greater than or equal to the end of \(s_i\) in dimension \(d\).

Figure 5.449 shows the objects of the example. Since \(O_1\) and \(O_2\) do not overlap, since \(O_1\) and \(O_3\) do not overlap, and since \(O_2\) and \(O_3\) also do not overlap, the non_overlap_sboxes constraint holds.

Typical

\[|\text{OBJECTS}| > 1\]

Symmetries

- Items of \(\text{OBJECTS}\) are \textbf{permutable}.
- Items of \(\text{SBOXES}\) are \textbf{permutable}.
- Items of \(\text{OBJECTS}.x\), \(\text{SBOXES}.t\) and \(\text{SBOXES}.l\) are \textbf{permutable} \textit{(same permutation used)}.
- \(\text{SBOXES}.l.v\) can be \textbf{decreased} to any value \(\geq 1\).

Arg. properties

`Suffix-contractible wrt. OBJECTS.`

Remark

In addition from preventing objects to overlap, the disjoint_sboxes constraint also enforces that borders and corners of objects are not directly in contact.

See also

\textbf{common keyword:} contains_sboxes, coveredby_sboxes, covers_sboxes (geometrical constraint between shifted boxes), diffn (geometrical constraint, non-overlapping), disjoint_sboxes
equal_sboxes (geometrical constraint between shifted boxes), geost, geost_time (geometrical constraint, non-overlapping), inside_sboxes, meet_sboxes, overlap_sboxes (geometrical constraint between shifted boxes), visible (geometrical constraint).

Keywords

constraint type: logic.
geometry: geometrical constraint, non-overlapping.
and O2 does not overlap O3

Figure 5.449: The three objects of the example
Logic

• origin($O_1, S_1, D$) \textit{def} = $O_1.x(D) + S_1.t(D)$
• end($O_1, S_1, D$) \textit{def} = $O_1.x(D) + S_1.t(D) + S_1.l(D)$
• non_overlap_sboxes($\text{Dims}, O_1, S_1, O_2, S_2$) \textit{def} =
  \[ \exists D \in \text{Dims} \left( \begin{array}{c}
  \text{end}(O_1,S_1,D) \leq \text{end}(O_2,S_2,D) \\
  \text{origin}(S_2,D) \\
  \text{origin}(S_1,D)
\end{array} \right) \]
• non_overlap_objects($\text{Dims}, O_1, O_2$) \textit{def} =
  \[ \forall S_1 \in \text{sboxes}[O_1.\text{sid}] \quad \forall S_2 \in \text{sboxes}[O_2.\text{sid}] \]
  \[ \text{non\_overlap\_sboxes}(\text{Dims}, O_1, S_1, O_2, S_2) \]
• all_non_overlap($\text{Dims}, OIDS$) \textit{def} =
  \[ \forall O_1 \in \text{objects}(OIDS) \quad \forall O_2 \in \text{objects}(OIDS) \quad O_1.\text{oid} < O_2.\text{oid} \rightarrow \]
  \[ \text{non\_overlap\_objects}(\text{Dims}, O_1, O_2) \]
• all_non_overlap($\text{DIMENSIONS}, OIDS$)
### 5.247 nor

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Logic</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>nor(VAR, VARIABLES)</td>
<td></td>
</tr>
<tr>
<td><strong>Synonym</strong></td>
<td>clause</td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>VAR : dvar, VARIABLES : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>VAR ≥ 0, VAR ≤ 1,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES ≥ 2,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES, var)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES.var ≥ 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES.var ≤ 1</td>
<td></td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>Let VARIABLES be a collection of 0-1 variables VAR₁, VAR₂, ..., VARₙ (n ≥ 2). Enforce VAR = ¬(VAR₁ ∨ VAR₂ ∨ ... ∨ VARₙ).</td>
<td></td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>(1, (0, 0))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(0, (0, 1))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(0, (1, 0))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(0, (1, 1))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(0, (1, 0, 1))</td>
<td></td>
</tr>
<tr>
<td><strong>Symmetry</strong></td>
<td>Items of VARIABLES are permutable.</td>
<td></td>
</tr>
<tr>
<td><strong>Arg. properties</strong></td>
<td>Functional dependency: VAR determined by VARIABLES.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Contractible wrt. VARIABLES when VAR = 1.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Extensible wrt. VARIABLES when VAR = 0.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Aggregate: VAR(∧), VARIABLES(union).</td>
<td></td>
</tr>
<tr>
<td><strong>Systems</strong></td>
<td>reifiedXnor in Choco, clause in Gecode, #\ in SICStus.</td>
<td></td>
</tr>
<tr>
<td><strong>See also</strong></td>
<td>common keyword: and, equivalent, imply, nand, or, xor (Boolean constraint).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>implies: atleast.nvalue.</td>
<td></td>
</tr>
</tbody>
</table>
Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint arguments: pure functional dependency.

constraint network structure: Berge-acyclic constraint network.

constraint type: Boolean constraint.

filtering: arc-consistency.

modelling: functional dependency.
Figure 5.450 depicts the automaton associated with the **nor** constraint. To the first argument \( \text{VAR} \) of the **nor** constraint corresponds the first signature variable. To each variable \( \text{VAR}_i \) of the second argument \( \text{VARIABLES} \) of the **nor** constraint corresponds the next signature variable. There is no signature constraint.

Figure 5.450: Automaton of the **nor** constraint

Figure 5.451: Hypergraph of the reformulation corresponding to the automaton of the **nor** constraint
5.248  not_all_equal

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>CHIP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>not_all_equal(VARIABLES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>VARIABLES : collection(var−dvar)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Restrictions| required(VARIABLES, var)  
|             | | | |
| Purpose     | The variables of the collection VARIABLES should take more than one single value. |
| Example     | ((3,1,3,3,3)) |
|             | The not_all_equal constraint holds since the collection (3,1,3,3,3) involves more than one value (i.e., values 1 and 3). |
| Typical     | | | |
|             | | | |
| Symmetries  | • Items of VARIABLES are permutable. |
|             | • All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value. |
| Arg. properties | Extensible wrt. VARIABLES. |
| Algorithm   | If the intersection of the domains of the variables of the VARIABLES collection is empty the not_all_equal constraint is entailed. Otherwise, when only one single variable V remains not fixed, remove the unique value (unique since the constraint is not entailed) taken by the other variables from the domain of V. |
| Reformulation | The not_all_equal(VARIABLES) constraint can be expressed as atleast_nvalue(2, VARIABLES). |
| Systems     | rel in Gecode. |
| See also    | generalisation: nvalue (introduce a variable for counting the number of distinct values). |
|             | implied by: alldifferent. |
|             | negation: all_equal. |
|             | specialisation: neq (when go down to two variables). |
|             | used in reformulation: atleast_nvalue. |
Keywords

characteristic of a constraint: disequality, automaton, automaton without counters, reified automaton constraint,

constraint network structure: sliding cyclic(1) constraint network(1).

constraint type: value constraint.

filtering: arc-consistency.

final graph structure: equivalence.
Graph model

Parts (A) and (B) of Figure 5.452 respectively show the initial and final graph associated with the Example slot. Since we use the NSCC graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a value that is assigned to some variables of the VARIABLES collection. The not_all_equal holds since the final graph contains more than one strongly connected component.

![Graph Diagram](image)

**Figure 5.452: Initial and final graph of the not_all_equal constraint**
Automaton

Figure 5.453 depicts the automaton associated with the \texttt{not\_all\_equal} constraint. To each pair of consecutive variables \((\text{VAR}_i, \text{VAR}_{i+1})\) of the collection \texttt{VARIABLES} corresponds a signature variable \(S_i\). The following signature constraint links \(\text{VAR}_i, \text{VAR}_{i+1}\) and \(S_i\):

\[ \text{VAR}_i = \text{VAR}_{i+1} \iff S_i. \]

Figure 5.453: Automaton of the \texttt{not\_all\_equal} constraint

Figure 5.454: Hypergraph of the reformulation corresponding to the automaton of the \texttt{not\_all\_equal} constraint
5.249 **not_in**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from <strong>in</strong>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>not_in(VAR, VALUES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VAR : dvar</td>
<td>VALUES : collection(val-int)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(VALEUES, val)</td>
<td>distinct(VALEUES, val)</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>Enforce VAR to be assigned a value different from the values of the VALUES collection.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>(2, ⟨1, 3⟩)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The constraint not_in holds since the value of its first argument VAR = 2 does not occur within the collection ⟨1, 3⟩.

<table>
<thead>
<tr>
<th>Typical</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Symmetries**
- Items of VALUES are **permutable**.
- One and the same constant can be **added** to VAR as well as to the val attribute of all items of VALUES.

**Arg. properties**
- Contractible wrt. VALUES.

**Remark**
- **Entailment** occurs immediately after posting this constraint and removing all values in VALUES from VAR.

**Systems**
- notMember in Choco, rel in Gecode.

**Used in**
- group.

**See also**
- negation: in.

**Keywords**
- characteristic of a constraint: disequality, automaton, automaton without counters, reified automaton constraint, derived collection.
- constraint arguments: unary constraint.
- constraint network structure: centered cyclic(1) constraint network(1).
- constraint type: value constraint.
- filtering: arc-consistency, entailment.
- modelling: excluded, domain definition.
Derived Collection

\[
\text{col}([\text{VARIABLES}\rightarrow\text{collection(\text{var}\rightarrow\text{dvar}), [item(\text{var}\rightarrow\text{VAR})]})]
\]

Arc input(s) VARIABLES VALUES
Arc generator \text{PRODUCT}\rightarrow\text{collection(\text{variables, values})}
Arc arity 2
Arc constraint(s) \text{variables.var} = \text{values.val}
Graph property(ies) \text{NARC} = 0

Graph model
Figure 5.455 shows the initial graph associated with the Example slot. Since we use the NARC = 0 graph property the corresponding final graph is empty.

Figure 5.455: Initial graph of the not in constraint (the final graph is empty)

Signature
Since 0 is the smallest number of arcs of the final graph we can rewrite NARC = 0 to NARC ≤ 0. This leads to simplify NARC to NARC.
Automaton

Figure 5.456 depicts the automaton associated with the \texttt{not\_in} constraint. Let $\text{VAL}_i$ be the \texttt{val} attribute of the $i^{th}$ item of the \texttt{VALUES} collection. To each pair $(\text{VAR}, \text{VAL}_i)$ corresponds a 0-1 signature variable $S_i$ as well as the following signature constraint: $\text{VAR} = \text{VAL}_i \leftrightarrow S_i$.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{automaton.png}
\caption{Automaton of the \texttt{not\_in} constraint}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{hypergraph.png}
\caption{Hypergraph of the reformulation corresponding to the automaton of the \texttt{not\_in} constraint}
\end{figure}
5.250 npair

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from nvalue.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>npair(NPAIRS, PAIRS)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>NPAIRS : dvar</td>
<td>PAIRS : collection(x−dvar, y−dvar)</td>
</tr>
<tr>
<td>Restrictions</td>
<td>NPAIRS ≥ min(1,</td>
<td>PAIRS</td>
</tr>
<tr>
<td>Purpose</td>
<td>NPAIRS is the number of distinct pairs of values assigned to the pairs of variables of the collection PAIRS.</td>
<td></td>
</tr>
</tbody>
</table>
| Example     | \[
\begin{pmatrix}
  x - 3 & y - 1, \\
  x - 1 & y - 5, \\
  2, (x - 3 & y - 1), \\
  x - 3 & y - 1, \\
  x - 1 & y - 5
\end{pmatrix}
\] |
| Typical     | NPAIRS > 1 | NPAIRS < |PAIRS| | |PAIRS| > 1 | range(PAIRS.x) > 1 | range(PAIRS.y) > 1 |
| Symmetries  | • Items of PAIRS are permutable. | • Attributes of PAIRS are permutable w.r.t. permutation (x, y) (permutation applied to all items). | • All occurrences of two distinct tuples of values of NPAIRS can be swapped; all occurrences of a tuple of values of NPAIRS can be renamed to any unused tuple of values. |
| Arg. properties | • Functional dependency: NPAIRS determined by PAIRS. | • Contractible wrt. PAIRS when NPAIRS = 1 and |PAIRS| > 0. | • Contractible wrt. PAIRS when NPAIRS = |PAIRS|. |
| Remark      | This is an example of a number of distinct values constraint where there is more than one attribute that is associated with each vertex of the final graph. |   |
See also related: \textit{nclass}\hspace{1em}(\textit{pair of variables replaced by variable } \in \textit{ partition}).
\textit{nequivalence}\hspace{1em}(\textit{pair of variables replaced by variable } \text{mod constant}).
\textit{ninterval}\hspace{1em}(\textit{pair of variables replaced by variable}/\text{constant}).
\textit{specialisation}\hspace{1em}(\textit{pair of variables replaced by variable}).

Keywords characteristic of a constraint: pair.
\textit{constraint arguments}: pure functional dependency.
\textit{constraint type}: counting constraint, value partitioning constraint.
\textit{final graph structure}: strongly connected component, equivalence.
\textit{modelling}: number of distinct equivalence classes, functional dependency.
Arc input(s) PAIRS
Arc generator $\text{CLIQUE} \rightarrow \text{collection}(\text{pairs1}, \text{pairs2})$
Arc arity 2
Arc constraint(s)
• $\text{pairs1}.x = \text{pairs2}.x$
• $\text{pairs1}.y = \text{pairs2}.y$
Graph property(ies) $\text{NSCC} = \text{NPAIRS}$

Graph model
Parts (A) and (B) of Figure 5.458 respectively show the initial and final graph associated with the Example slot. Since we use the NSCC graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a pair of values that is assigned to some pairs of variables of the PAIRS collection. In our example we have the following pairs of values: $(x - 3, y - 1)$ and $(x - 1, y - 5)$.

![Graph model diagram](image-url)

Figure 5.458: Initial and final graph of the npair constraint
5.251 nset_of_consecutive_values

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>N. Beldiceanu</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>nset_of_consecutive_values(N, VARIABLES)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>N : dvar</td>
<td>VARIABLES : collection(var−dvar)</td>
</tr>
<tr>
<td>Restrictions</td>
<td>N ≥ 1</td>
<td>N ≤</td>
</tr>
<tr>
<td>Purpose</td>
<td>N is the number of set of consecutive values used by the variables of the collection VARIABLES.</td>
<td></td>
</tr>
</tbody>
</table>

Example

```
( var − 3, 
  var − 1, 
  var − 7, 
  var − 1, 
  var − 1, 
  var − 2, 
  var − 8 )
```

In the example, the two parts 3, 1, 1, 2 and 7, 8 take respectively their values in the following sets of consecutive values \{1, 2, 3\} and \{7, 8\}. Consequently, the nset_of_consecutive_values constraint holds since its first argument \(N = 2\) is set to the number of sets of consecutive values.

Typical

\[N > 1\]
\[|\text{VARIABLES}| > 1\]
\[\text{range}(\text{VARIABLES}.\text{var}) > 1\]

Symmetries

- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

Functional dependency: N determined by VARIABLES.

Usage

Used for specifying the fact that the values have to be used in a compact way is achieved by setting \(N\) to 1.

See also

common keyword:
max_size_set_of_consecutive_var, min_size_set_of_consecutive_var (consecutive values).
Keywords

- characteristic of a constraint: consecutive values.
- constraint arguments: pure functional dependency.
- constraint type: value constraint.
- final graph structure: strongly connected component.
Arc input(s)  VARIABLES
Arc generator  $CLIQUE \rightarrow \text{collection}(\text{variables1}, \text{variables2})$
Arc arity  2
Arc constraint(s)  $\text{abs}(\text{variables1}.\text{var} - \text{variables2}.\text{var}) \leq 1$
Graph property(ies)  $\text{NSCC} = N$

Graph model

Since the arc constraint is symmetric each strongly connected component of the final graph corresponds exactly to one connected component of the final graph.

Parts (A) and (B) of Figure 5.459 respectively show the initial and final graph associated with the Example slot. Since we use the NSCC graph property, we show the two strongly connected components of the final graph.

![Figure 5.459: Initial and final graph of the nset_of_consecutive_values constraint](image-url)
**5.252 nvalue**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[283]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td><code>nvalue(NVAL, VARIABLES)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td><code>cardinality_on_attributes.values,values.</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td><code>NVAL : dvar</code>&lt;br&gt;<code>VARIABLES : collection(var−dvar)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td><code>required(VARIABLES,var)</code>&lt;br&gt;`NVAL ≥ min(1,</td>
<td>VARIABLES</td>
<td>)<code>&lt;br&gt;</code>NVAL ≤</td>
</tr>
<tr>
<td>Purpose</td>
<td><code>NVAL</code> is the number of distinct values taken by the variables of the collection <code>VARIABLES</code>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td><code>(4, ⟨3, 1, 7, 1, 6⟩)</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The `nvalue` constraint holds since its first argument `NVAL = 4` is set to the number of distinct values occurring within the collection `(3, 1, 7, 1, 6)`.  

**Typical**

- `NVAL > 1`  
- `NVAL < |VARIABLES|`  
- `NVAL < range(VARIABLES.var)`  
- `|VARIABLES| > 1`  
- `NVAL < 0 ∨ NVAL > 1`

**Symmetries**

- Items of `VARIABLES` are **permutable**.
- All occurrences of two distinct values of `VARIABLES.var` can be **swapped**; all occurrences of a value of `VARIABLES.var` can be **renamed** to any unused value.

**Arg. properties**

- **Functional dependency**: `NVAL` determined by `VARIABLES`.
- **Contractible** wrt. `VARIABLES` when `NVAL = 1` and `|VARIABLES| > 0`.
- **Contractible** wrt. `VARIABLES` when `NVAL = |VARIABLES|`.

**Usage**

A classical example from the early 1850s is the **dominating queens** chess puzzle problem: Place a number of queens on a `n` by `n` chessboard in such a way that all squares are either attacked by a queen or are occupied by a queen. A queen can attack all squares located on the same column, on the same row or on the same diagonal. Part (A) of Figure 5.460 illustrates a set of five queens which together attack all of the squares of an `8` by `8` chessboard. The **dominating queens** problem can be modelled as one single `nvalue` constraint:
• We first label the different squares of the chessboard from 1 to \( n^2 \).

• We then associate to each square \( S \) of the chessboard a domain variable. Its initial domain is set to the numbers of the squares that can be attacked from \( S \). For instance, in the context of an 8 by 8 chessboard, the initial domain of \( V_{29} \) will be set to \( \{2,5,8,11,13,15,20,22,25,26,27,29,32,36,38,43,45,47,50,53,56,57,61\} \) (see the green squares of part (B) of Figure 5.460).

• Finally, we post the constraint \( \text{nvalue}(Q, (\text{var} - V_1, \text{var} - V_2, \ldots, \text{var} - V_n)) \) where \( Q \) is a domain variable in \([1, n^2]\) that gives the total number of queens used for controlling all squares of the chessboard. For the solution depicted by Part (A) of Figure 5.460, the number in each square of Part (C) of Figure 5.460 gives the value assigned to the corresponding variable. Note that, since a given square can be attacked by several queens, we have also other assignments corresponding to the solution depicted by Part (A) of Figure 5.460.

Figure 5.460: Modelling the dominating queens problem with one single nvalue constraint

The nvalue constraint occurs also in many practical applications. In the context of timetabling one wants to set up a limit on the maximum number of activity types it is possible to perform. For frequency allocation problems, one optimisation criteria is to minimise the number of distinct frequencies that you use all over the entire network. The nvalue constraint generalises several constraints like:

• alldifferent(VARIABLES): in order to get the alldifferent constraint, one has to set NVAL to the total number of variables.

• not_all_equal(VARIABLES): in order to get the not_all_equal constraint, one has to set the minimum value of NVAL to 2.

Remark

This constraint appears in [283, page 339] under the name of Cardinality on Attributes Values. The nvalue constraint is called values in JaCoP (http://www.jacop.eu/). A constraint called k_diff enforcing that a set of variables takes at least k distinct values appears in the PhD thesis of J.-C. Régis [321].

It was shown in [65] that, finding out whether a nvalue constraint has a solution or not is NP-hard. This was achieved by reduction from 3-SAT. In the same article, it is also shown, by reduction from minimum hitting set cardinality, that computing a sharp lower bound on NVAL is NP-hard.

Both reformulations of the coloured_cumulative constraint and of the coloured_cumulatives constraint use the nvalue constraint.
A first filtering algorithm for the \textit{nvalue} constraint was described in [26]. Assuming that the minimum value of variable NVAL is not constrained at all, two algorithms that both achieve bound-consistency were provided one year later in [38]. Under the same assumption, algorithms that partially take into account holes in the domains of the variables of the \textsc{Variables} collection are described in [38, 58].

A model, involving linear inequalities constraints, preserving bound-consistency was introduced in [68].

\textbf{Systems} \textit{nvalues in Gecode}, \textit{nvalue in MiniZinc}, \textit{nvalue in SICStus}.

\textbf{Used in} track.

\textbf{See also} assignment dimension added: assign\_and\_nvalues.

\textbf{Keywords} characteristic of a constraint: core, automaton, automaton with array of counters.

complexity: 3-SAT, minimum hitting set cardinality.

constraint arguments: pure functional dependency.

constraint type: counting constraint, value partitioning constraint.

filtering: bound-consistency, convex bipartite graph.

\textbf{Algorithm} A first filtering algorithm for the \textit{nvalue} constraint was described in [26]. Assuming that the minimum value of variable NVAL is not constrained at all, two algorithms that both achieve bound-consistency were provided one year later in [38]. Under the same assumption, algorithms that partially take into account holes in the domains of the variables of the \textsc{Variables} collection are described in [38, 58].

\textbf{Reformulation} A model, involving linear inequalities constraints, preserving bound-consistency was introduced in [68].

\textbf{Systems} \textit{nvalues in Gecode}, \textit{nvalue in MiniZinc}, \textit{nvalue in SICStus}.

\textbf{Used in} track.

\textbf{See also} assignment dimension added: assign\_and\_nvalues.

\textbf{Keywords} characteristic of a constraint: core, automaton, automaton with array of counters.

complexity: 3-SAT, minimum hitting set cardinality.

constraint arguments: pure functional dependency.

constraint type: counting constraint, value partitioning constraint.

filtering: bound-consistency, convex bipartite graph.
**final graph structure:** strongly connected component, equivalence.

**modelling:** number of distinct equivalence classes, number of distinct values, functional dependency.

**problems:** domination.

**puzzles:** dominating queens.
Arc input(s) | VARIABLES
---|---
Arc generator | $CLIQUE \rightarrow \text{collection}(\text{variables}_1, \text{variables}_2)$
Arc arity | 2
Arc constraint(s) | $\text{variables}_1.\text{var} = \text{variables}_2.\text{var}$
Graph property(ies) | $\text{NSCC} = \text{NVAL}$
Graph class | EQUIVALENCE

Graph model

Parts (A) and (B) of Figure 5.461 respectively show the initial and final graph associated with the Example slot. Since we use the NSCC graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a value that is assigned to some variables of the VARIABLES collection. The following values 1, 3, 6 and 7 are used by the variables of the VARIABLES collection.

![Initial and final graph of the nvalue constraint](image)

Figure 5.461: Initial and final graph of the nvalue constraint
Automaton Figure 5.462 depicts the automaton associated with the nvalue constraint. To each item of the collection VARIABLES corresponds a signature variable $S_i$ that is equal to 0.

\[ \text{among}_\text{diff}_0(N,C) \]

\[ S: \]

\[ \{ C[\_]=0 \} \]

\[ 0, \]

\[ \{ C[VAR_1]=C[VAR_1]+1 \} \]

Figure 5.462: Automaton of the nvalue constraint
5.253  nvalue_on_intersection

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
</table>

**Origin**

Derived from `common` and `nvalue`.

**Constraint**

nvalue_on_intersection(NVAL, VARIABLES1, VARIABLES2)

**Arguments**

NVAL : dvar
VARIABLES1 : collection(var−dvar)
VARIABLES2 : collection(var−dvar)

**Restrictions**

required(VARIABLES1.var)
required(VARIABLES2.var)
NVAL ≥ 0
NVAL ≤ |VARIABLES1|
NVAL ≤ |VARIABLES2|
NVAL ≤ range(VARIABLES1.var)
NVAL ≤ range(VARIABLES2.var)

**Purpose**

NVAL is the number of distinct values that both occur in the VARIABLES1 and VARIABLES2 collections.

**Example**

\[
\{2, (1, 9, 1, 5), \text{var} - 2, \text{var} - 1, \text{var} - 9, \text{var} - 9, \text{var} - 6, \text{var} - 9\}
\]

Note that the two collections \(\langle 1, 9, 1, 5 \rangle\) and \(\langle 2, 1, 9, 6, 9 \rangle\) share two values in common (i.e., values 1 and 9). Consequently the nvalue_on_intersection constraint holds since its first argument NVAL is set to 2.

**Typical**

NVAL > 0
NVAL < |VARIABLES1|
NVAL < |VARIABLES2|
NVAL < range(VARIABLES1.var)
NVAL < range(VARIABLES2.var)
|VARIABLES1| > 1
|VARIABLES2| > 1
### Symmetries

- Arguments are permutable w.r.t. permutation (NVAL) (VARIABLES1, VARIABLES2).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- All occurrences of two distinct values in VARIABLES1.var or VARIABLES2.var can be swapped; all occurrences of a value in VARIABLES1.var or VARIABLES2.var can be renamed to any unused value.

### Arg. properties

- **Functional dependency:** NVAL determined by VARIABLES1 and VARIABLES2.
- **Contractible** wrt. VARIABLES1 when NVAL = 0.
- **Contractible** wrt. VARIABLES2 when NVAL = 0.

### See also

- **common keyword:** alldifferent_on_intersection, common,
  same_intersection (constraint on the intersection).
- **root concept:** nvalue.

### Keywords

- **constraint arguments:** pure functional dependency.
- **constraint type:** counting constraint, constraint on the intersection.
- **final graph structure:** connected component.
- **modelling:** number of distinct values, functional dependency.
Graph model

Parts (A) and (B) of Figure 5.463 respectively show the initial and final graph associated with the Example slot. Since we use the NCC graph property we show the connected components of the final graph. The variable $NVAL$ is equal to this number of connected components. Note that all the vertices corresponding to the variables that take values 5, 2 or 6 were removed from the final graph since there is no arc for which the associated equality constraint holds.

![Graph Diagram](image)

Figure 5.463: Initial and final graph of the nvalue_on_intersection constraint
### 5.254 nvalues

**Description**

Inspired by nvalue and count.

**Constraint**

nvalues(VARIABLES, RELOP, LIMIT)

**Arguments**

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARIABLES</td>
<td>collection(var–dvar)</td>
</tr>
<tr>
<td>RELOP</td>
<td>atom</td>
</tr>
<tr>
<td>LIMIT</td>
<td>dvar</td>
</tr>
</tbody>
</table>

**Restrictions**

required(VARIABLES, var)

RELOP ∈ [=, ≠, <, ≥, >, ≤]

**Purpose**

Let \( N \) be the number of distinct values assigned to the variables of the VARIABLES collection. Enforce condition \( N \) RELOP LIMIT to hold.

**Example**

\[
\langle \text{var}−4, \text{var}−5, \text{var}−5, \text{var}−4, \text{var}−1, \text{var}−5 \rangle, =, 3
\]

The nvalues constraint holds since the number of distinct values occurring within the collection \( \langle 4, 5, 5, 4, 1, 5 \rangle \) is equal (i.e., RELOP is set to =) to its third argument LIMIT = 3.

**Typical**

\[ |\text{VARIABLES}| > 1 \]
\[ \text{LIMIT} > 1 \]
\[ \text{LIMIT} < |\text{VARIABLES}| \]
\[ \text{RELOP} \in [=, <, ≥, >, ≤] \]

**Symmetries**

- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

**Arg. properties**

- Contractible wrt. VARIABLES when RELOP ∈ [<, ≤].
- Contractible wrt. VARIABLES when RELOP ∈ [=], LIMIT = 1 and |VARIABLES| > 0.
- Contractible wrt. VARIABLES when RELOP ∈ [=] and LIMIT = |VARIABLES|.
- Extensible wrt. VARIABLES when RELOP ∈ [≥, >].

**Usage**

Used in the Constraint(s) on sets slot for defining some constraints like assign_and_nvalues, circuit_cluster or coloured_cumulative.
Reformulation

The \texttt{nvalues} \texttt{(VARIABLES, RELOP, LIMIT)} constraint can be expressed in term of the conjunction \texttt{nvalue}(\textit{NV}, \texttt{VARIABLES}) \& \textit{NV} \texttt{RELOP LIMIT}.

Systems

\texttt{nvalues} in \texttt{Gecode}.

Used in

\texttt{assign\_and\_nvalues}, \texttt{circuit\_cluster}, \texttt{coloured\_cumulative}, \texttt{coloured\_cumulatives}.

See also

\texttt{assignment dimension added}: \texttt{assign\_and\_nvalues}.
\texttt{common keyword}: \texttt{nvalues\_except\_0} \texttt{(counting constraint,number of distinct values)}.
\texttt{specialisation}: \texttt{nvalue} \texttt{(replace a comparison with the number of distinct values by an equality with the number of distinct values)}.

Keywords

\texttt{constraint type}: counting constraint, value partitioning constraint.
\texttt{final graph structure}: strongly connected component, equivalence.
\texttt{modelling}: number of distinct equivalence classes, number of distinct values.
\texttt{problems}: domination.
Arc input(s) \[ \text{VARIABLES} \]

Arc generator \[ \text{CLIQUE} \rightarrow \text{collection}(\text{variables1}, \text{variables2}) \]

Arc arity 2

Arc constraint(s) \[ \text{variables1}.\text{var} = \text{variables2}.\text{var} \]

Graph property(ies) \[ \text{NSCC RELOP LIMIT} \]

Graph class \[ \text{EQUIVALENCE} \]

Graph model

Parts (A) and (B) of Figure 5.464 respectively show the initial and final graph associated with the Example slot. Since we use the NSCC graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a value that is assigned to some variables of the VARIABLES collection. The 3 following values 1, 4 and 5 are used by the variables of the VARIABLES collection.

![Graph model](image)

Figure 5.464: Initial and final graph of the `nvalues` constraint
### 5.255 nvalues_except_0

**DESCRIPTION**

Derived from `nvalues`.

**LINKS**

nvalues_except_0(VARIABLES, RELOP, LIMIT)

**GRAPH**

- **Arguments**
  - VARIABLES : `collection(var–dvar)`
  - RELOP : `atom`
  - LIMIT : `dvar`

- **Restrictions**
  - required(VARIABLES, var)
  - RELOP ∈ `[=, ≠, <, ≥, >, ≤]`

- **Purpose**
  
  Let \( N \) be the number of distinct values, different from 0, assigned to the variables of the `VARIABLES` collection. Enforce condition \( N \text{ RELOP LIMIT} \) to hold.

- **Example**
  
  \[
  \begin{pmatrix}
  \text{var} - 4, \\
  \text{var} - 5, \\
  \text{var} - 5, \\
  \text{var} - 4, \\
  \text{var} - 0, \\
  \text{var} - 1
  \end{pmatrix}, =, 3
  \]

  The `nvalues_except_0` constraint holds since the number of distinct values, different from 0, occurring within the collection \( \langle 4, 5, 5, 4, 0, 1 \rangle \) is equal (i.e., `RELOP` is set to `=`) to its third argument `LIMIT = 3`.

- **Typical**
  
  \[
  \begin{align*}
  |\text{VARIABLES}| & > 1 \\
  \text{LIMIT} & < |\text{VARIABLES}| \\
  \text{atleast}(1, \text{VARIABLES}, 0) & \\
  \text{RELOP} & \in [<, ≥, >, ≤]
  \end{align*}
  \]

- **Symmetries**
  - Items of `VARIABLES` are permutable.
  - All occurrences of two distinct values of `VARIABLES.var` that are both different from 0 can be swapped; all occurrences of a value of `VARIABLES.var` that is different from 0 can be renamed to any unused value that is also different from 0.

- **Arg. properties**
  - **Contractible** wrt. `VARIABLES` when `RELOP ∈ [<, ≤]`.
  - **Extensible** wrt. `VARIABLES` when `RELOP ∈ [≥, >]`.

- **Reformulation**
  
  The `nvalues_except_0(V_1, V_2, \ldots, V_{|\text{VARIABLES}|}, \text{RELOP}, \text{LIMIT})` constraint can be expressed in term of the conjunction `nvalue(NVI, \langle 0, V_1, V_2, \ldots, V_{|\text{VARIABLES}|} \rangle) ∧ NVI - 1 \text{ RELOP LIMIT}`.
Used in cycle or accessibility.

See also common keyword: assign and nvalues (number of distinct values), nvalue, nvalues (counting constraint, number of distinct values).

Keywords characteristic of a constraint: joker value.
constraint type: counting constraint, value partitioning constraint.
final graph structure: strongly connected component.
modelling: number of distinct values.
Arc input(s) | VARIABLES
---|---
Arc generator | \( \text{CLIQUE} \rightarrow \text{collection}(\text{variables1, variables2}) \)
Arc arity | 2
Arc constraint(s) | • variables1.var \( \neq 0 \)

Graph property(ies) | \text{NSCC, RELOP LIMIT}

Graph model

Parts (A) and (B) of Figure 5.465 respectively show the initial and final graph associated with the Example slot. Since we use the NSCC graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a value distinct from 0 that is assigned to some variables of the VARIABLES collection. Beside value 0, the 3 following values 1, 4 and 5 are assigned to the variables of the VARIABLES collection.

![Graph Model](image)

Figure 5.465: Initial and final graph of the nvalues_except_0 constraint
5.256 \text{nvector}

**Description**
Introduced by G. Chabert as a generalisation of \text{nvalue}

**Constraint**
\text{nvector}(\text{NVEC}, \text{VECTORS})

**Synonyms**
nvectors, npoint, npoints.

**Type**
\text{VECTOR} : \text{collection}(\text{var} \rightarrow \text{dvar})

**Arguments**
\text{NVEC} : \text{dvar}
\text{VECTORS} : \text{collection}(\text{vec} \rightarrow \text{VECTOR})

**Restrictions**
- $|\text{VECTOR}| \geq 1$
- $\text{NVEC} \geq \text{min}(1, |\text{VECTORS}|)$
- $\text{NVEC} \leq |\text{VECTORS}|$
- \text{required}(\text{VECTORS}, \text{vec})
- \text{same_size}(\text{VECTORS}, \text{vec})

**Purpose**
\text{NVEC} is the number of distinct tuples of values taken by the vectors of the collection \text{VECTORS}. Two tuples of values \(\langle A_1, A_2, \ldots, A_m \rangle\) and \(\langle B_1, B_2, \ldots, B_m \rangle\) are distinct if and only if there exist an integer \(i \in [1, m]\) such that \(A_i \neq B_i\).

**Example**
\[
\begin{pmatrix}
\text{vec} \rightarrow (5, 6), \\
2, \\
\text{vec} \rightarrow (9, 3)
\end{pmatrix}
\]

The \text{nvector} constraint holds since its first argument \text{NVEC} = 2 is set to the number of distinct tuples of values (i.e., tuples \((5, 6)\) and \((9, 3)\)) occurring within the collection \text{VECTORS}. Figure 5.466 depicts with a thick rectangle a possible initial domain for each of the five vectors and with a grey circle each tuple of values of the corresponding solution.

**Typical**
- $|\text{VECTOR}| > 1$
- $\text{NVEC} > 1$
- $\text{NVEC} < |\text{VECTORS}|$
- $|\text{VECTORS}| > 1$

**Symmetries**
- Items of \text{VECTORS} are permutable.
- Items of \text{VECTORS.vec} are permutable (same permutation used).
- All occurrences of two distinct tuples of values of \text{VECTORS.vec} can be swapped; all occurrences of a tuple of values of \text{VECTORS.vec} can be renamed to any unused tuple of values.
Arg. properties

- Functional dependency: NVEC determined by VECTORS.
- Contractible w.r.t. VECTORS when NVEC = 1 and |VECTORS| > 0.
- Contractible w.r.t. VECTORS when NVEC = |VECTORS|.

Remark

It was shown in [103, 102] that, finding out whether a nvector constraint has a solution or not is NP-hard (i.e., the restriction to the rectangle case and to the atmost side of the nvector were considered for this purpose). This was achieved by reduction from the rectangle clique partition problem.

Reformulation

Assume the collection VECTORS is not empty (otherwise NVEC = 0). In this context, let \( n \) and \( m \) respectively denote the number of vectors of the collection VECTORS and the number of components of each vector. Furthermore, let \( \alpha_i = \min(C_{1i}, C_{2i}, \ldots, C_{mi}) \), \( \beta_i = \max(C_{1i}, C_{2i}, \ldots, C_{mi}) \), \( \gamma_i = \beta_i - \alpha_i + 1, (i \in [1, m]) \). By associating to each vector \( \langle C_{k1}, C_{k2}, \ldots, C_{km} \rangle, (k \in [1, n]) \) a variable

\[
D_k = \sum_{1 \leq i \leq m} \left( \prod_{i<j \leq m} \gamma_j \right) \cdot (C_{ki} - \alpha_i),
\]

the constraint

\[
\text{nvector}(\text{NVEC, vec} = \langle C_{11}, C_{12}, \ldots, C_{1m} \rangle,\text{vec} = \langle C_{21}, C_{22}, \ldots, C_{2m} \rangle,\text{vec} = \langle C_{n1}, C_{n2}, \ldots, C_{nm} \rangle)
\]

can be expressed in term of the constraint \( \text{nvalue}(\text{NVEC, } \langle D_1, D_2, \ldots, D_n \rangle) \).

Note that the previous reformulation does not work anymore if the variables have a continuous domain, or if an overflow occurs while propagating the equality constraint

\[
D_k = \sum_{1 \leq i \leq m} \left( \prod_{i<j \leq m} \gamma_j \right) \cdot (C_{ki} - \alpha_i)
\]

(i.e., the number of components \( m \) is too big).

When using this reformulation with respect to the Example slot we first introduce \( D_1 = 1\cdot6-3+(4\cdot5-20)) = 3, D_2 = 1\cdot6-3+(4\cdot5-20)) = 3, D_3 = 1\cdot3-3+(4\cdot9-20)) = 16, D_4 = 1\cdot6-3+(4\cdot5-20)) = 3, D_5 = 1\cdot3-3+(4\cdot9-20)) = 16 \) and then get the constraint \( \text{nvalue}(2, (3, 3, 16, 3, 16)) \).

See also

common keyword: \text{lex_equal, ordered_atleast_nvector, ordered_atmost_nvector(vectors)}.

generalisation: \text{nvectors (replace an equality with the number of distinct vectors by a comparison with the number of distinct vectors)}.

implied by: \text{ordered_nvector}.

implies: \text{atleast_nvector (= NVEC replaced by \( \geq \) NVEC), atmost_nvector (= NVEC replaced by \( \leq \) NVEC)}.

specialisation: \text{nvalue (vector replaced by variable)}.
Keywords

application area: SLAM problem.
characteristic of a constraint: vector.
complexity: rectangle clique partition.
constraint arguments: pure functional dependency.
constraint type: counting constraint, value partitioning constraint.
final graph structure: strongly connected component, equivalence.
modelling: number of distinct equivalence classes, functional dependency.
problems: domination.
Figure 5.466: Initial possible initial domains ($C_{11} \in [1, 6], C_{12} \in [2, 6], C_{21} \in [3, 5], C_{22} \in [6, 9], C_{31} \in [4, 10], C_{32} \in [1, 4], C_{41} \in [5, 9], C_{42} \in [3, 7], C_{51} \in [9, 11], C_{52} \in [0, 5]$) and solution corresponding to the example
Arc input(s) | VECTORS
---|---
Arc generator | \( \text{CLIQUE} \mapsto \text{collection}(\text{vectors1, vectors2}) \)
Arc arity | \( \text{2} \)
Arc constraint(s) | \( \text{lex.equal}(\text{vectors1.vec, vectors2.vec}) \)
Graph property(ies) | \( \text{NSCC} = \text{NVEC} \)
Graph class | \( \text{EQUIVALENCE} \)

Graph model
Parts (A) and (B) of Figure 5.467 respectively show the initial and final graph associated with the Example slot. Since we use the NSCC graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a tuple of values that is assigned to some vectors of the VECTORS collection. The 2 following tuple of values \( \langle 5, 6 \rangle \) and \( \langle 9, 3 \rangle \) are used by the vectors of the VECTORS collection.

![Graph](image)

Figure 5.467: Initial and final graph of the nvector constraint
5.257  nvectors

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>VECTORS : collection(var–dvar)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Origin**

Inspired by `nvector` and `count`.

**Constraint**

`nvectors(VECTORS, RELOP, LIMIT)`

**Synonym**

`npoints`.

**Type**

`VECTOR : collection(var–dvar)`

**Arguments**

- `VECTORS : collection(vec − VECTOR)`
- `RELOP : atom`
- `LIMIT : dvar`

**Restrictions**

- `|VECTOR| ≥ 1`
- `required(VECTORS, vec)`
- `same_size(VECTORS, vec)`
- `RELOP ∈ [=, ≠, <, ≥, >, ≤]`

**Purpose**

Let $N$ be the number of distinct tuples of values taken by the vectors of the `VECTORS` collection. Enforce condition $N \text{ } \text{RELOP} \text{ } \text{LIMIT}$ to hold.

**Example**

```plaintext
(vec − (5, 6),
  vec − (5, 6),
  vec − (9, 3),
  vec − (5, 6),
  vec − (9, 3))
```

The `nvectors` constraint holds since the number of distinct tuples of values (i.e., tuples `(5, 6)` and `(9, 3)`) occurring within the collection `VECTORS` is equal (i.e., `RELOP` is set to `=`) to its third argument `LIMIT = 2`.

**Typical**

- `|VECTOR| > 1`
- `|VECTORS| > 1`
- `RELOP ∈ [=, <, ≥, >, ≤]`
- `LIMIT > 1`
- `LIMIT < |VECTORS|`

**Symmetries**

- Items of `VECTORS` are permutable.
- Items of `VECTORS.vec` are permutable (same permutation used).
- All occurrences of two distinct values of `VECTORS.vec` can be swapped; all occurrences of a value of `VECTORS.vec` can be renamed to any unused value.
Arg. properties

- **Contractible** wrt. VECTORS when $\text{RELOP} \in [\langle, \leq]$.  
- **Extensible** wrt. VECTORS when $\text{RELOP} \in [\geq, \rangle$.

Reformulation

The $\text{nvec(n)\{\text{VECTORS, RELOP, LIMIT}\}}$ constraint can be expressed in term of the conjunction $\text{nvec(n, VECTORS) \land n \text{\ RELOP LIMIT}}$.

See also

**specialisation:** $\text{nvec(n)}$ (replace a comparison with the number of distinct vectors by an equality with the number of distinct vectors).

Keywords

- characteristic of a constraint: vector.
- constraint type: counting constraint, value partitioning constraint.
- final graph structure: strongly connected component, equivalence.
- modelling: number of distinct equivalence classes.
- problems: domination.
Parts (A) and (B) of Figure 5.468 respectively show the initial and final graph associated with the Example slot. Since we use the NSCC graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a tuple of values that is assigned to some vectors of the VECTORS collection. The following tuple of values $\langle 5, 6 \rangle$ and $\langle 9, 3 \rangle$ are used by the vectors of the VECTORS collection.

Figure 5.468: Initial and final graph of the n_vectors constraint
5.258  **nvisible_from_end**

**DESCRIPTION**

**Origin**
Derived from `nvisible_from_start`

**Constraint**

`nvisible_from_end(N, VARIABLES)`

**Synonyms**
`nvisible, nvisible_from_right`

**Arguments**

- `N : dvar`
- `VARIABLES : collection(var−dvar)`

**Restrictions**

- `required(VARIABLES, var)`
- `N ≥ min(1, |VARIABLES|)`
- `N ≤ |VARIABLES|`

**Purpose**

The `i^{th}` variable of the sequence `VARIABLES` is **visible** if and only if all variables after the `i^{th}` variable are strictly smaller than the `i^{th}` variable itself. `N` is the total number of visible variables of the sequence of variables `VARIABLES`.

**Example**

```
( var−1,
  var−6,
  var−2,
  var−1,
  var−4,
  var−8,
  var−2 )
```

The `nvisible` constraint holds since the sequence 1 6 2 1 4 8 2 contains two visible items that respectively correspond to the seventh and sixth items.

**Typical**

`|VARIABLES| > 2`

**Symmetry**

One and the same constant can be **added** to the `var` attribute of all items of `VARIABLES`.

**Arg. properties**

- **Functional dependency**: `N` determined by `VARIABLES`.

**See also**

`related: nvisible_from_start` *(count from the start of the sequence rather than from the end)*

**Keywords**
- **combinatorial object**: sequence.
- **constraint arguments**: pure functional dependency.
- **modelling**: functional dependency.
5.259  nvisible_from_start

**DESCRIPTION**

**Origin**
Derived from a puzzle called skyscraper

**Constraint**
\( \text{nvisible\_from\_start}(N, \text{VARIABLES}) \)

**Synonyms**
\( \text{nvisible}, \text{nvisible\_from\_left} \)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>dvar</td>
</tr>
<tr>
<td>VARIABLES</td>
<td>collection(var−dvar)</td>
</tr>
</tbody>
</table>

**Restrictions**

\[
\begin{align*}
\text{required}(\text{VARIABLES}, \text{var}) \\
N \geq \min(1, |\text{VARIABLES}|) \\
N \leq |\text{VARIABLES}|
\end{align*}
\]

**Purpose**
The \( i^{th} (1 \leq i \leq |\text{VARIABLES}|) \) variable of the sequence \( \text{VARIABLES} \) is \textit{visible} if and only if all variables before the \( i^{th} \) variable are strictly smaller than the \( i^{th} \) variable itself. \( N \) is the total number of visible variables of the sequence of variables \( \text{VARIABLES} \).

**Example**

\[
\begin{pmatrix}
\text{var} - 1, \\
\text{var} - 6, \\
\text{var} - 2, \\
3, \\
\text{var} - 1, \\
\text{var} - 4, \\
\text{var} - 8, \\
\text{var} - 2
\end{pmatrix}
\]

The \text{invisible} constraint holds since the sequence 1 6 2 1 4 8 2 contains three visible items that respectively correspond to the first, second and sixth items.

**Typical**

\(|\text{VARIABLES}| > 2\)

**Symmetry**
One and the same constant can be added to the \text{var} attribute of all items of \( \text{VARIABLES} \).

**Arg. properties**

Functional dependency: \( N \) determined by \( \text{VARIABLES} \).

See also

related: \text{nvisible\_from\_end} (count from the end of the sequence rather than from the start).

**Keywords**

combinatorial object: sequence.

constraint arguments: pure functional dependency.

modelling: functional dependency.
5.260 open_alldifferent

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[402]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>open_alldifferent(S, VARIABLES)</td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>open_alldiff, open_alldistinct, open_distinct.</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>S : svar</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES : collection(var − dvar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>S ≥ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S ≤</td>
<td>VARIABLES</td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES, var)</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>Let ( V ) be the variables of the collection VARIABLES for which the corresponding position belongs to the set ( S ). Positions are numbered from 1. Enforce all variables of ( V ) to take distinct values.</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>(({2, 3, 4}, \langle 9, 1, 9, 3 \rangle))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The open_alldifferent constraint holds since the last three (i.e., ( S = {2, 3, 4} )) values of the collection ( \langle 9, 1, 9, 3 \rangle ) are distinct.</td>
<td></td>
</tr>
<tr>
<td>Typical</td>
<td>(</td>
<td>\text{VARIABLES}</td>
</tr>
<tr>
<td>Symmetry</td>
<td>All occurrences of two distinct values of VARIABLES:var can be swapped; all occurrences of a value of VARIABLES:var can be renamed to any unused value.</td>
<td></td>
</tr>
<tr>
<td>Arg. properties</td>
<td>Suffix-contractible wrt. VARIABLES.</td>
<td></td>
</tr>
<tr>
<td>Usage</td>
<td>In their article [402], W.-J. van Hoeve and J.-C. Régin motivate the open_alldifferent constraint by the following scheduling problem. Consider a set of activities (where each activity has a fixed duration 1 and a start variable) that can be processed on two factory lines such that all the activities that will be processed on a given line must be pairwise distinct. This can be modelled by using one open_alldifferent constraint for each line, involving all the start variables as well as a set variable whose final value specifies the set of activities assigned to that specific factory line.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Note that this can also be directly modelled by one single diffn constraint. This is done by introducing an assignment variable for each activity. The initial domain of each assignment variable consists of two values that respectively correspond to the two factory lines.</td>
<td></td>
</tr>
<tr>
<td>Algorithm</td>
<td>A slight adaptation of the flow model that handles the original global_cardinality constraint [322] is described in [402].</td>
<td></td>
</tr>
</tbody>
</table>
common keyword: \textit{size} \textit{max_seq} \textit{alldifferent}, \textit{size max starting seq alldifferent} (\textit{all different}, \textit{disequality}).

generalisation: \textit{open global cardinality} (control the number of occurrence of each active value\textsuperscript{10} with a counter variable), \textit{open global cardinality low up} (control the number of occurrence of each active value with an interval).

hard version: \textit{alldifferent}.

used in graph description: \textit{in set}.

Keywords

characteristic of a constraint: all different, disequality.

constraint arguments: constraint involving set variables.

constraint type: open constraint, soft constraint, value constraint.

filtering: flow.

\textsuperscript{10}An active value corresponds to a value occurring at a position mentioned in the set $S$. 
Arc input(s) | VARIABLES
---|---
Arc generator | CLIQUE→collection(variables1,variables2)
Arc arity | 2
Arc constraint(s) | • variables1.var = variables2.var
• in_set(variables1.key,S)
• in_set(variables2.key,S)

Graph property(ies) | MAX_NSCC≤ 1
Graph class | ONE_SUC

Graph model

We generate a clique with an equality constraint between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed one. Variables for which the corresponding position does not belong to the set $S$ are removed from the final graph by the second and third conditions of the arc-constraint.

Parts (A) and (B) of Figure 5.469 respectively show the initial and final graph associated with the Example slot. Since we use the MAX_NSCC graph property we show one of the largest strongly connected component of the final graph. The open_alldifferent holds since all the strongly connected components have at most one vertex: a value is used at most once.

Figure 5.469: Initial and final graph of the open_alldifferent constraint
### 5.261 open_among

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from among and open_global_cardinality.</td>
</tr>
<tr>
<td>Constraint</td>
<td>open_among(S, NVAR, VARIABLES, VALUES)</td>
</tr>
</tbody>
</table>
| Arguments   | S : svar  
VVAR : dvar  
VARIABLES : collection(var—dvar)  
VALUES : collection(val—int) |
| Restrictions| $S \geq 1$  
$S \leq |\text{VARIABLES}|$  
$\text{NVAR} \geq 0$  
$\text{NVAR} \leq |\text{VARIABLES}|$  
required(\text{VARIABLES}, var)  
required(\text{VALUES}, val)  
\text{distinct}(\text{VALUES}, val) |
| Purpose     | Let $V$ be the variables of the collection VARIABLES for which the corresponding position belongs to the set $S$. Positions are numbered from 1. NVAR is the number of variables of $V$ that take their value in VALUES. |
| Example     | \[
\begin{pmatrix}
{2, 3, 4, 5}, 3, \\
{8, 5, 5, 4, 1}, \\
(1, 5, 8)
\end{pmatrix}
\]  
The \text{open_among} constraint holds since within the last four values (i.e., $S = \{2, 3, 4, 5\}$) of $\langle 8, 5, 5, 4, 1 \rangle$ exactly 3 values belong to the set of values $\{1, 5, 8\}$. |
| Typical     | $\text{NVAR} > 0$  
$\text{NVAR} < |\text{VARIABLES}|$  
$|\text{VARIABLES}| > 1$  
$|\text{VALUES}| > 1$  
$|\text{VARIABLES}| > |\text{VALUES}|$ |
| Symmetries  | • Items of VALUES are permutable.  
• An occurrence of a value of VARIABLES.var that belongs to VALUES.val (resp. does not belong to VALUES.val) can be replaced by any other value in VALUES.val (resp. not in VALUES.val). |
| Arg. properties | • Functional dependency: NVAR determined by S, VARIABLES and VALUES.  
• Suffix-contractible wrt. VARIABLES when NVAR = 0. |
See also

**common keyword:** open_atleast, open_atmost \((open\ constraint, value\ constraint)\),
open_global_cardinality \((open\ constraint, counting\ constraint)\).

**hard version:** among.

**used in graph description:** in_set.

**Keywords**

**constraint arguments:** constraint involving set variables.

**constraint type:** open constraint, value constraint, counting constraint.

**modelling:** functional dependency.
Arc input(s) | VARIABLES
---|---
Arc generator | $SELF \rightarrow \text{collection}(\text{variables})$
Arc arity | 1
Arc constraint(s) | • $\text{in}(\text{variables}.\text{var}, \text{VALUES})$
| • $\text{in}_{\text{set}}(\text{variables}.\text{key}, S)$
Graph property(ies) | NARC = NVAR

Graph model

The arc constraint corresponds to the conjunction of unary constraints $\text{in}(\text{variables}.\text{var}, \text{VALUES})$ and $\text{in}_{\text{set}}(\text{variables}.\text{key}, S)$ defined in this catalogue. Consequently we employ the SELF arc generator in order to produce an initial graph with a single loop on each vertex.

Parts (A) and (B) of Figure 5.470 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the loops of the final graph are stressed in bold.

![Figure 5.470: Initial and final graph of the open_among constraint](image-url)
5.262  open_atleast

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived from <code>atleast</code> and <code>open_global_cardinality</code>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CONSTRAINT</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>open_atleast(S, N, VARIABLES, VALUE)</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ARGUMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>S</code> : <code>svar</code></td>
</tr>
<tr>
<td><code>N</code> : <code>int</code></td>
</tr>
<tr>
<td><code>VARIABLES</code> : <code>collection(var−dvar)</code></td>
</tr>
<tr>
<td><code>VALUE</code> : <code>int</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RESTRICTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S \geq 1 )</td>
</tr>
<tr>
<td>( S \leq</td>
</tr>
<tr>
<td>( N \geq 0 )</td>
</tr>
<tr>
<td>( N \leq</td>
</tr>
<tr>
<td>( \text{required}(VARIABLES, \text{var}) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let ( \mathcal{V} ) be the variables of the collection <code>VARIABLES</code> for which the corresponding position belongs to the set <code>S</code>. Positions are numbered from 1. At least <code>N</code> variables of ( \mathcal{V} ) are assigned value <code>VALUE</code>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EXAMPLE</th>
</tr>
</thead>
</table>
| \[
\begin{pmatrix}
\{2, 3, 4\}, 2, \\
\{4, 2, 4, 4\}, 4
\end{pmatrix}
\]

The `open_atleast` constraint holds since, within the last three (i.e., `S = \{2, 3, 4\}`) values of the collection `{4, 2, 4, 4}`, at least `N = 2` values are equal to value `VALUE = 4`.

<table>
<thead>
<tr>
<th>TYPICAL</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>N &gt; 0</code></td>
</tr>
<tr>
<td>`N &lt;</td>
</tr>
<tr>
<td>`</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SYMMETRIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <code>N</code> can be decreased to any value ( \geq 0 ).</td>
</tr>
<tr>
<td>• An occurrence of a value of <code>VARIABLES,\text{var}</code> that is different from <code>VALUE</code> can be replaced by any other value.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ARG. PROPERTIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suffix-extensible wrt. <code>VARIABLES</code>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SEE ALSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>common keyword: <code>open_among</code>, <code>open_global_cardinality</code> (<code>open constraint, value constraint</code>).</td>
</tr>
<tr>
<td>comparison swapped: <code>open_atmost</code>.</td>
</tr>
<tr>
<td>hard version: <code>atleast</code>.</td>
</tr>
<tr>
<td>used in graph description: <code>in_set</code>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KEYWORDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint arguments: constraint involving set variables.</td>
</tr>
<tr>
<td>constraint type: open constraint, value constraint.</td>
</tr>
<tr>
<td>modelling: at least.</td>
</tr>
</tbody>
</table>
Arc input(s)  VARIABLES
Arc generator  $SELF\rightarrow\text{collection}(\text{variables})$
Arc arity  1
Arc constraint(s)  
- $\text{variables}.\text{var} = \text{VALUE}$
- $\text{in}\_\text{set}(\text{variables}.\text{key}, S)$
Graph property(ies)  $\text{NARC} \geq N$

Graph model
Since each arc constraint involves only one vertex (VALUE is fixed), we employ the $SELF$ arc generator in order to produce a graph with a single loop on each vertex. Variables for which the corresponding position does not belong to the set $S$ are removed from the final graph by the second condition of the arc-constraint.

Parts (A) and (B) of Figure 5.471 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the loops of the final graph are stressed in bold.

![Diagram](A) ![Diagram](B)

Figure 5.471: Initial and final graph of the open_atleast constraint
## 5.263 open_atmost

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Derived from atmost and open_global_cardinality.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Constraint

\[
\text{open_atmost}(S, N, \text{VARIABLES}, \text{VALUE})
\]

### Arguments

- **S** : svar
- **N** : int
- **VARIABLES** : collection(var−dvar)
- **VALUE** : int

### Restrictions

- \(S \geq 1\)
- \(S \leq |\text{VARIABLES}|\)
- \(N \geq 0\)
- \(\text{required}(\text{VARIABLES}, \text{var})\)

### Purpose

Let \(V\) be the variables of the collection \(\text{VARIABLES}\) for which the corresponding position belongs to the set \(S\). Positions are numbered from 1. At most \(N\) variables of \(V\) are assigned value \(\text{VALUE}\).

### Example

\[
\left(\{2, 3, 4\}, 1, \{2, 2, 4, 5\}, 2\right)
\]

The open_atmost constraint holds since, within the last three (i.e., \(S = \{2, 3, 4\}\)) values of the collection \(\{2, 2, 4, 5\}\), at most \(N = 1\) value is equal to value \(\text{VALUE} = 2\).

### Typical

- \(N > 0\)
- \(N < |\text{VARIABLES}|\)
- \(|\text{VARIABLES}| > 1\)

### Symmetries

- \(N\) can be increased.
- An occurrence of a value of \(\text{VARIABLES}.\text{var}\) can be replaced by any other value that is different from \(\text{VALUE}\).

### Arg. properties

Suffix-contractible wrt. \(\text{VARIABLES}\).

### See also

- **common keyword**: open_among, open_global_cardinality (open constraint, value constraint).
- **comparison swapped**: open_atleast.
- **hard version**: atmost.
- **used in graph description**: in_set.

### Keywords

- **constraint arguments**: constraint involving set variables.
- **constraint type**: open constraint, value constraint.
- **modelling**: at most.
### Arc input(s)

- **VARIABLES**

### Arc generator

- $SELF \rightarrow \text{collection(} \text{variables} \text{)}$

### Arc arity

- 1

### Arc constraint(s)

- $\bullet \text{variables.var} = \text{VALUE}$
- $\bullet \text{in.set(} \text{variables.key} , S \text{)}$

### Graph property(ies)

- $\text{NARC} \leq N$

### Graph model

Since each arc constraint involves only one vertex (VALUE is fixed), we employ the $SELF$ arc generator in order to produce a graph with a single loop on each vertex. Variables for which the corresponding position does not belong to the set $S$ are removed from the final graph by the second condition of the arc-constraint.

Parts (A) and (B) of Figure 5.472 respectively show the initial and final graph associated with the Example slot. Since we use the $\text{NARC}$ graph property, the loops of the final graph are stressed in bold.

![Graphs](image-url)

(A)  
(B)

Figure 5.472: Initial and final graph of the open_atmost constraint
### 5.264 open_global_cardinality

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>[402]</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>open_global_cardinality(S, VARIABLES, VALUES)</td>
<td></td>
</tr>
<tr>
<td><strong>Synonyms</strong></td>
<td>open_gcc, ogcc.</td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S : svar</td>
<td>VARIABLES : collection(var−dvar)</td>
<td>VALUES : collection(val−int, noccurrence−dvar)</td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S ≥ 1</td>
<td>S ≤</td>
<td>VARIABLES</td>
</tr>
<tr>
<td>required(VARIABLES, var)</td>
<td>required(VARIABLES, [val, noccurrence])</td>
<td>distinct(VARIABLES, val)</td>
</tr>
<tr>
<td>VALUES.noccurrence ≥ 0</td>
<td>VALUES.noccurrence ≤</td>
<td>VARIABLES</td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Each value VALUES[i].val (1 ≤ i ≤</td>
<td>VALUES</td>
<td>) should be taken by exactly VALUES[i].noccurrence variables of the VARIABLES collection for which the corresponding position belongs to the set S. Positions are numbered from 1.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| \[
\begin{pmatrix}
{2, 3, 4}, \\
{3, 3, 8, 6}, \\
val = 3 \text{ noccurrence = 1}, \\
val = 5 \text{ noccurrence = 0}, \\
val = 6 \text{ noccurrence = 1}
\end{pmatrix}
\] | | |
| The open_global_cardinality constraint holds since: | | |
| • Values 3, 5 and 6 respectively occur 1, 0 and 1 times within the collection \langle 3, 3, 8, 6\rangle (the first item 3 of \langle 3, 3, 8, 6\rangle is ignored since value 1 does not belong to the first argument \(S = \{2, 3, 4\}\) of the open_global_cardinality constraint). | | |
| • No constraint was specified for value 8. | | |
| **Typical** | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| **Symmetries** | | |
| • Items of VALUES are permutable. | | |
| • An occurrence of a value of VARIABLES.var that does not belong to VALUES.val can be replaced by any other value that also does not belong to VALUES.val. | | |
Usage

In their article [402], W.-J. van Hoeve and J.-C. Régin motivate the open global cardinality constraint by the following scheduling problem. Consider a set of activities (where each activity has a fixed duration 1 and a start variable) that can be processed on two factory lines such that all the activities that will be processed on a given line must be pairwise distinct. This can be modelled by using one open global cardinality constraint for each line, involving all the start variables as well as a set variable whose final value specifies the set of activities assigned to that specific factory line.

Note that this can also be directly modelled by one single `diffn` constraint. This is done by introducing an assignment variable for each activity. The initial domain of each assignment variable consists of two values that respectively correspond to the two factory lines.

Remark

In their article [402], W.-J. van Hoeve and J.-C. Régin consider the case where we have no counter variables for the values, but rather some lower and upper bounds (i.e., in fact the open global cardinality low up constraint).

Algorithm

A slight adaptation of the flow model that handles the original global cardinality constraint [322] is described in [402].

See also

**common keyword:** global cardinality low up (assignment, counting constraint), open among (open constraint, counting constraint), open atleast, open atmost (open constraint, value constraint).

**hard version:** global cardinality.

**specialisation:** open all different (each active value should occur at most once), open global cardinality low up (variable replaced by fixed interval).

used in graph description: in set.

Keywords

**application area:** assignment.

**constraint arguments:** constraint involving set variables.

**constraint type:** open constraint, value constraint, counting constraint.

**filtering:** flow.

\[^{11} \text{An active value corresponds to a value occurring at a position mentioned in the set S.}\]
For all items of VALUES:

<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td><code>SELF -&gt; collection(variables)</code></td>
</tr>
<tr>
<td>Arc arity</td>
<td>1</td>
</tr>
</tbody>
</table>
| Arc constraint(s) | • variables.var = VALUES.val  
                  | • `in_set(variables.key, S)`                                              |
| Graph property(ies) | `NVERTEX = VALUES.noccurrence`                                           |

**Graph model**

Since we want to express one unary constraint for each value we use the “For all items of VALUES” iterator. The only difference with the graph model of the `global_cardinality` constraint is the arc constraint where we also specify that the position of the considered variable should belong to the first argument $S$.

Part (A) of Figure 5.473 shows the initial graphs associated with each value 3, 5 and 6 of the VALUES collection of the **Example** slot. Part (B) of Figure 5.473 shows the two corresponding final graphs respectively associated with values 3 and 6 that are both assigned to those variables of the VARIABLES collection for which the index belongs to $S$ (since value 5 is not assigned to any variable of the VARIABLES collection the final graph associated with value 5 is empty). Since we use the `NVERTEX` graph property, the vertices of the final graphs are stressed in bold.

Figure 5.473: Initial and final graph of the open `global_cardinality` constraint
open_global_cardinality_low_up

**DESCRIPTION**

Constraint

open_global_cardinality_low_up(S, VARIABLES, VALUES)

**Arguments**

- **S**: svar
- **VARIABLES**: collection(var–dvar)
- **VALUES**: collection(val–int, omin–int, omx–int)

**Restrictions**

- \(S \geq 1\)
- \(S \leq |\text{VARIABLES}|\)
  - \(\text{required}([\text{VARIABLES}], \text{val})\)
  - \(|\text{VALUES}| > 0\)
  - \(\text{required}([\text{VALUES}], [\text{val}, \text{omin}, \text{omax}])\)
  - \(\text{distinct}([\text{VALUES}], \text{val})\)
  - \(\text{VALUES}.\text{omin} \geq 0\)
  - \(\text{VALUES}.\text{omax} \leq |\text{VARIABLES}|\)
  - \(\text{VALUES}.\text{omin} \leq \text{VALUES}.\text{omax}\)

**Purpose**

Each value \(\text{VALUES}[i].\text{val} (1 \leq i \leq |\text{VALUES}|)\) should be taken by at least \(\text{VALUES}[i].\text{omin}\) and at most \(\text{VALUES}[i].\text{omax}\) variables of the \(\text{VARIABLES}\) collection for which the corresponding position belongs to the set \(S\). Positions are numbered from 1.

**Example**

\[
\begin{pmatrix}
2, 3, 4, \\
3, 3, 8, 6,
\end{pmatrix}
\]

\[
\begin{pmatrix}
\text{val} - 3 & \text{omin} - 1 & \text{omax} - 3, \\
\text{val} - 5 & \text{omin} - 0 & \text{omax} - 1, \\
\text{val} - 6 & \text{omin} - 1 & \text{omax} - 2
\end{pmatrix}
\]

The open_global_cardinality_low_up constraint holds since:

- Values 3, 5 and 6 are respectively used 1 (1 \(\leq 1 \leq 3\), 0 (0 \(\leq 0 \leq 1\) and 1 (1 \(\leq 1 \leq 2\) times within the collection \(3, 3, 8, 6\) (the first item 3 of \(3, 3, 8, 6\) is ignored since value 1 does not belong to the first argument \(S = \{2, 3, 4\}\) of the open_global_cardinality_low_up constraint).
- No constraint was specified for value 8.

**Typical**

- \(|\text{VARIABLES}| > 1\)
- \(\text{range}([\text{VARIABLES}], \text{var}) > 1\)
- \(|\text{VALUES}| > 1\)
- \(\text{VALUES}.\text{omin} \leq |\text{VARIABLES}|\)
- \(\text{VALUES}.\text{omax} > 0\)
- \(\text{VALUES}.\text{omax} \leq |\text{VARIABLES}|\)
- \(|\text{VARIABLES}| > |\text{VALUES}|\)
Symmetries

- Items of VALUES are permutable.
- An occurrence of a value of VARIABLES\_\text{var} that does not belong to VALUES\_\text{val} can be replaced by any other value that also does not belong to VALUES\_\text{val}.

Usage

In their article [402], W.-J. van Hoeve and J.-C. Régin motivate the open\_global\_cardinality\_low\_up constraint by the following scheduling problem. Consider a set of activities (where each activity has a fixed duration 1 and a start variable) that can be processed on two factory lines such that all the activities that will be processed on a given line must be pairwise distinct. This can be modelled by using one open\_global\_cardinality\_low\_up constraint for each line, involving all the start variables as well as a set variable whose final value specifies the set of activities assigned to that specific factory line.

Note that this can also be directly modelled by one single \texttt{diffn} constraint. This is done by introducing an assignment variable for each activity. The initial domain of each assignment variable consists of two values that respectively correspond to the two factory lines.

Algorithm

A slight adaptation of the flow model that handles the original global\_cardinality constraint [322] is described in [402].

See also

- common keyword: global\_cardinality (assignment, counting constraint).
- generalisation: open\_global\_cardinality (fixed interval replaced by variable).
- hard version: global\_cardinality\_low\_up.
- specialisation: open\_alldifferent (each active value\footnote{An active value corresponds to a value occurring at a position mentioned in the set $S$.} should occur at most once).
- used in graph description: in\_set.

Keywords

- application area: assignment.
- constraint arguments: constraint involving set variables.
- constraint type: open constraint, value constraint, counting constraint.
- filtering: flow.
For all items of VALUES:

**Arc input(s)**
VARIABLES

**Arc generator**
SELF \rightarrow collection(variables)

**Arc arity**
1

**Arc constraint(s)**
- variables.var = VALUES.val
- in_set(variables.key, S)

**Graph property(ies)**
- \( N\text{VERTEX} \geq VALUES.\text{omin} \)
- \( N\text{VERTEX} \leq VALUES.\text{omax} \)

**Graph model**
Since we want to express one unary constraint for each value we use the “For all items of VALUES” iterator. The only difference with the graph model of the global_cardinality_low_up constraint is the arc constraint where we also specify that the position of the considered variable should belong to the first argument S.

Part (A) of Figure 5.474 shows the initial graphs associated with each value 3, 5 and 6 of the VALUES collection of the Example slot. Part (B) of Figure 5.474 shows the two corresponding final graphs respectively associated with values 3 and 6 that are both assigned to the variables of the VARIABLES collection (since value 5 is not assigned to any variable of the VARIABLES collection the final graph associated with value 5 is empty). Since we use the \( N\text{VERTEX} \) graph property, the vertices of the final graphs are stressed in bold.

![Graphs](image-url)

Figure 5.474: Initial and final graph of the open_global_cardinality_low_up constraint
5.266 open_maximum

### Description

**Origin**
Derived from maximum

**Constraint**
open_maximum(MAX, VARIABLES)

**Arguments**

MAX : dvar
VARIABLES : collection(var−dvar, bool−dvar)

**Restrictions**

| VARIABLES| > 0
required(VARIABLES, [var, bool])
VARIABLES.bool ≥ 0
VARIABLES.bool ≤ 1

**Purpose**
MAX is the maximum value of the variables VARIABLES[i].var, (1 ≤ i ≤ |VARIABLES|) for which VARIABLES[i].bool = 1 (at least one of the Boolean variables is set to 1).

**Example**

```
( var−3 bool−1,
  var−1 bool−0,  
  5,   var−7 bool−0,  
  var−5 bool−1, 
  var−5 bool−1 )
```

The open_maximum constraint holds since its first argument MAX = 5 is set to the maximum value of values 3, 1, 7, 5, 5 for which the corresponding Boolean 1, 0, 0, 1, 1 is set to 1 (i.e., values 3, 5, 5).

**Typical**

| VARIABLES| > 1
range(VARIABLES.var) > 1

**Symmetries**

- Items of VARIABLES are permutable.
- One and the same constant can be added to MAX as well as to the var attribute of all items of VARIABLES.

**See also**
comparison swapped: open_minimum.
hard version: maximum.
used in graph description: in_set.

**Keywords**
characteristic of a constraint: maximum, automaton, automaton without counters, reified automaton constraint.
constraint network structure: centered cyclic constraint network.
constraint type: order constraint, open constraint, open automaton constraint.
Automaton

Figure 5.475 depicts the automaton associated with the open maximum constraint. Let $\text{VAR}_i, B_i$ be the $i^{th}$ item of the VARIABLES collection. To each triple $(\text{MAX}, \text{VAR}_i, B_i)$ corresponds a signature variable $S_i$ as well as the following signature constraint: 

\[
\begin{align*}
(B_i = 0 \land \text{MAX} < \text{VAR}_i \iff S_i = 0) & \land (B_i = 1 \land \text{MAX} = \text{VAR}_i \iff S_i = 1) \land (B_i = 1 \land \text{MAX} > \text{VAR}_i \iff S_i = 2) \land (B_i = 0 \land \text{MAX} < \text{VAR}_i \iff S_i = 3) \land (B_i = 0 \land \text{MAX} = \text{VAR}_i \iff S_i = 4) \land (B_i = 0 \land \text{MAX} > \text{VAR}_i \iff S_i = 5).
\end{align*}
\]

Figure 5.475: Automaton of the open maximum constraint

Figure 5.476: Hypergraph of the reformulation corresponding to the automaton of the open maximum constraint
### 5.267 open_minimum

<table>
<thead>
<tr>
<th>Origin</th>
<th>Derived from minimum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint</td>
<td>open_minimum(MIN, VARIABLES)</td>
</tr>
<tr>
<td>Arguments</td>
<td>MIN : dvar</td>
</tr>
<tr>
<td></td>
<td>VARIABLES : collection(var−dvar, bool−dvar)</td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>MIN is the minimum value of the variables VARIABLES[i].var, (1 ≤ i ≤</td>
</tr>
<tr>
<td>Example</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Typical</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Symmetries</td>
<td>• Items of VARIABLES are permutable.</td>
</tr>
<tr>
<td></td>
<td>• One and the same constant can be added to MIN as well as to the var attribute of all items of VARIABLES.</td>
</tr>
<tr>
<td>Remark</td>
<td>The open_minimum constraint is used in the reformulation of the tree_range constraint.</td>
</tr>
<tr>
<td>See also</td>
<td>comparison swapped: open_maximum.</td>
</tr>
<tr>
<td></td>
<td>hard version: minimum.</td>
</tr>
<tr>
<td></td>
<td>used in graph description: in_set.</td>
</tr>
<tr>
<td></td>
<td>uses in its reformulation: tree_range.</td>
</tr>
<tr>
<td>Keywords</td>
<td>characteristic of a constraint: minimum, automaton, automaton without counters, reified automaton constraint.</td>
</tr>
<tr>
<td></td>
<td>constraint network structure: centered cyclic(1) constraint network(1).</td>
</tr>
<tr>
<td></td>
<td>constraint type: order constraint, open constraint, open automaton constraint.</td>
</tr>
</tbody>
</table>
Automaton

Figure 5.477 depicts the automaton associated with the open_minimum constraint. Let $\text{VAR}_i, B_i$ be the $i^{th}$ item of the VARIABLES collection. To each triple $(\text{MIN}, \text{VAR}_i, B_i)$ corresponds a signature variable $S_i$ as well as the following signature constraint:

\[
(B_i = 1 \land \text{MIN} < \text{VAR}_i \iff S_i = 0) \land (B_i = 1 \land \text{MIN} = \text{VAR}_i \iff S_i = 1) \land (B_i = 1 \land \text{MIN} > \text{VAR}_i \iff S_i = 2) \land (B_i = 0 \land \text{MIN} < \text{VAR}_i \iff S_i = 3) \land (B_i = 0 \land \text{MIN} = \text{VAR}_i \iff S_i = 4) \land (B_i = 0 \land \text{MIN} > \text{VAR}_i \iff S_i = 5).
\]

Figure 5.477: Automaton of the open_minimum constraint

Figure 5.478: Hypergraph of the reformulation corresponding to the automaton of the open_minimum constraint
## 5.268 opposite_sign

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Arithmetic.</td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>opposite_sign(VAR1, VAR2)</td>
</tr>
</tbody>
</table>
| **Arguments** | VAR1 : dvar  
VAR2 : dvar |
| **Restriction** | |
| **Purpose** | Enforce the fact that the product of the first and second variables is less than or equal to 0. |
| **Example** | (6, −3) |
| The opposite_sign constraint holds since 6 and −3 do not have the same sign. |
| **Typical** | VAR1 \neq 0 |
| **Symmetry** | Arguments are permutable w.r.t. permutation (VAR1, VAR2). |
| **See also** | comparison swapped: same_sign. |
| **Keywords** | constraint arguments: binary constraint.  
constraint type: predefined constraint, arithmetic constraint.  
filtering: arc-consistency. |
### 5.269  \textbf{or}

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Logic</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>or(VAR, VARIABLES)</td>
<td></td>
</tr>
<tr>
<td><strong>Synonym</strong></td>
<td>rel.</td>
<td></td>
</tr>
</tbody>
</table>
| **Arguments** | \begin{align*}
VAR & : \text{dvar} \\
\text{VARIABLES} & : \text{collection}(\text{var}–\text{dvar})
\end{align*} |           |
| **Restrictions** | \begin{align*}
VAR & \geq 0  \\
VAR & \leq 1  \\
|\text{VARIABLES}| & \geq 2  \\
\text{required}(\text{VARIABLES, var})  \\
\text{VARIABLES.var} & \geq 0  \\
\text{VARIABLES.var} & \leq 1
\end{align*} |          |
| **Purpose** | Let VARIABLES be a collection of 0-1 variables \(\text{VAR}_1, \text{VAR}_2, \ldots, \text{VAR}_n\) \((n \geq 2)\). Enforce \(\text{VAR} = \text{VAR}_1 \lor \text{VAR}_2 \lor \ldots \lor \text{VAR}_n\). |          |
| **Example** | \begin{align*}
(0, (0, 0))  \\
(1, (0, 1))  \\
(1, (1, 0))  \\
(1, (1, 1))  \\
(1, (1, 0, 1))
\end{align*} |          |
| **Symmetry** | Items of VARIABLES are permutable. |          |
| **Arg. properties** | \begin{itemize}
\item Functional dependency: VAR determined by VARIABLES.  \\
\item Contractible wrt. VARIABLES when VAR = 0.  \\
\item Extensible wrt. VARIABLES when VAR = 1.  \\
\item Aggregate: VAR(\lor), VARIABLES(union).
\end{itemize} |          |
| **Systems** | reversed in Choco, rel in Gecode, orbool in JaCoP, \#\lor in SICStus. |          |
| **See also** | \begin{itemize}
\item common keyword: and, clause_or, equivalent, imply, nand, nor, xor (Boolean constraint).  \\
\item implies: atleast_nvalue, maximum.
\end{itemize} |          |
Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint arguments: pure functional dependency.

constraint network structure: Berge-acyclic constraint network.

constraint type: Boolean constraint.

filtering: arc-consistency.

modelling: disjunction, functional dependency.
Automaton

Figure 5.479 depicts the automaton associated with the or constraint. To the first argument \( \text{VAR} \) of the or constraint corresponds the first signature variable. To each variable \( \text{VAR}_i \) of the second argument \( \text{VARIABLES} \) of the or constraint corresponds the next signature variable. There is no signature constraint.

![Automaton Diagram]

Figure 5.479: Automaton of the or constraint

![Hypergraph Diagram]

Figure 5.480: Hypergraph of the reformulation corresponding to the automaton of the or constraint
5.270 orchard

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
</table>

Origin

[206]

Constraint

orchard(NROW, TREES)

Arguments

NROW : dvar
TREES : collection(index=int, x=dvar, y=dvar)

Restrictions

NROW ≥ 0
TREES.index ≥ 1
TREES.index ≤ |TREES|
required(TREES,[index,x,y])
distinct(TREES,index)
TREES.x ≥ 0
TREES.y ≥ 0

Purpose

Orchard problem [206]:

“Your aid I want, Nine trees to plant, In rows just half a score, And let there be, In each row, three—Solve this: I ask no more!”

Example

The 10 alignments of 3 trees correspond to the following triples of trees: (1, 2, 3), (1, 4, 8), (1, 5, 9), (2, 4, 7), (2, 5, 8), (2, 6, 9), (3, 5, 7), (3, 6, 8), (4, 5, 6), (7, 8, 9).

Figure 5.481 shows the 9 trees and the 10 alignments corresponding to the example.

Typical

NROW > 0
|TREES| > 3

Symmetries

- Items of TREES are permutable.
- Attributes of TREES are permutable w.r.t. permutation (index) (x,y) (permutation applied to all items).
- One and the same constant can be added to the x attribute of all items of TREES.
- One and the same constant can be added to the y attribute of all items of TREES.
Functional dependency: NROW determined by TREES.

Keywords:
- characteristic of a constraint: hypergraph.
- constraint arguments: pure functional dependency.
- geometry: geometrical constraint, alignment.
Figure 5.481: Nine trees with 10 alignments of 3 trees
Arc input(s)  TRES
Arc generator  CLIQUE(<) \rightarrow \text{collection}(\text{trees1, trees2, trees3})
Arc arity  3
Arc constraint(s)  
\[
\begin{align*}
\sum \left( & \text{trees1.x} \times \text{trees2.y} - \text{trees1.x} \times \text{trees3.y}, \\
& \text{trees1.y} \times \text{trees3.x} - \text{trees1.y} \times \text{trees2.x}, \\
& \text{trees2.x} \times \text{trees3.y} - \text{trees2.y} \times \text{trees3.x} \right) = 0
\end{align*}
\]
Graph property(ies)  NARC = NROW

Graph model
The arc generator CLIQUE(<) with an arity of three is used in order to generate all the arcs of the directed hypergraph. Each arc is an ordered triple of trees. We use the restriction < in order to generate one single arc for each set of three trees. This is required, since otherwise we would count more than once a given alignment of three trees. The formula used within the arc constraint expresses the fact that the three points of respective coordinates \((\text{trees1.x, trees1.y}), (\text{trees2.x, trees2.y})\) and \((\text{trees3.x, trees3.y})\) are aligned. It corresponds to the development of the expression:

\[
\begin{array}{ccc}
\text{trees1.x} & \text{trees2.y} & 1 \\
\text{trees2.x} & \text{trees2.y} & 1 \\
\text{trees3.x} & \text{trees3.y} & 1 \\
\end{array}
= 0
\]
5.271 ordered_atleast_nvector

**DESCRIPTION**

Conjoin atleast_nvector and lex_chainlesseq.

**LINKS**

ordered_atleast_nvector(NVEC, VECTORS)

**GRAPH**

ordered_atleast_nvectors, ordered_atleast_npoint, ordered_atleast_npoints.

**Type**

VECTOR : collection(var−dvar)

**Arguments**

NVEC : dvar
VECTORS : collection(vec − VECTOR)

**Restrictions**

|VECTOR| ≥ 1

NVEC ≥ 0

NVEC ≤ |VECTORS|

required(VECTORS, vec)

same_size(VECTORS, vec)

Enforces the following two conditions:

1. The number of distinct tuples of values taken by the vectors of the collection VECTORS is greater than or equal to NVEC. Two tuples of values ⟨A1, A2, . . . , Am⟩ and ⟨B1, B2, . . . , Bm⟩ are distinct if and only if there exist an integer i ∈ [1, m] such that Ai ≠ Bi.

2. For each pair of consecutive vectors VECTORi and VECTORi+1 of the VECTORS collection we have that VECTORi is lexicographically less than or equal to VECTORi+1. Given two vectors, X and Y of n components, ⟨X0, . . . , Xn−1⟩ and ⟨Y0, . . . , Yn−1⟩, X is lexicographically less than or equal to Y if and only if n = 0 or X0 < Y0 or X0 = Y0 and ⟨X1, . . . , Xn−1⟩ is lexicographically less than or equal to ⟨Y1, . . . , Yn−1⟩.

**Example**

\[
\begin{pmatrix}
2, \\
\text{vec} - (5, 6), \\
\text{vec} - (5, 6), \\
\text{vec} - (9, 3), \\
\text{vec} - (9, 4)
\end{pmatrix}
\]

The ordered_atleast_nvector constraint holds since:

1. The collection VECTORS involves at least 2 distinct tuples of values (i.e., in fact the 3 distinct tuples ⟨5, 6⟩, ⟨9, 3⟩ and ⟨9, 4⟩).

2. The vectors of the collection VECTORS are sorted in increasing lexicographical order.
Typical

|VECTOR| > 1
NVEC > 0
NVEC < |VECTORS|
|VECTORS| > 1

Symmetry

NVEC can be decreased to any value ≥ 0.

Reformulation

The ordered_atleast_nvector constraint can be reformulated as a conjunction of a atleast_nvector and a lex_chain_lesseq constraints.

See also

- common keyword: nvector (vector).
- comparison swapped: ordered_atmost_nvector.
- implied by: ordered_nvector (≥ NVEC replaced by = NVEC).
- implies: atleast_nvector, lex_chain_lesseq(NVEC of constraint ordered_atleast_nvector removed).
- used in graph description: lex_less, lex_lesseq.

Keywords

- characteristic of a constraint: vector.
- constraint type: counting constraint, order constraint.
- symmetry: symmetry.
Arc input(s) | VECTORS
---|---
Arc generator | \(PATH \rightarrow \text{collection}(\text{vectors1}, \text{vectors2})\)
Arc arity | 2
Arc constraint(s) | \(\text{lex,lesseq}(\text{vectors1}.\text{vec}, \text{vectors2}.\text{vec})\)
Graph property(ies) | \(\text{NARC} = |\text{VECTORS}| - 1\)

| Arc input(s) | VECTORS
---|---
Arc generator | \(PATH \rightarrow \text{collection}(\text{vectors1}, \text{vectors2})\)
Arc arity | 2
Arc constraint(s) | \(\text{lex,less}(\text{vectors1}.\text{vec}, \text{vectors2}.\text{vec})\)
Graph property(ies) | \(\text{NCC} \geq \text{NVEC}\)

**Graph model**

Parts (A) and (B) of Figure 5.482 respectively show the initial and final graph of the second graph constraint associated with the Example slot. Since we use the NCC graph property in this second graph constraint, we show the different connected components of the final graph. Each strongly connected component corresponds to a tuple of values that is assigned to some vectors of the VECTORS collection. The 3 following tuple of values \(\langle 5, 6 \rangle\), \(\langle 9, 3 \rangle\) and \(\langle 9, 4 \rangle\) are used by the vectors of the VECTORS collection.

![Graph Model](image)

Figure 5.482: Initial and final graph of the ordered_atleast_nvector constraint
### 5.272  ordered_atmost_nvector

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Conjoin <code>atmost_nvector</code> and <code>lex_chain_lesseq</code>.</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td><code>ordered_atmost_nvector(NVEC, VECTORS)</code></td>
<td></td>
</tr>
<tr>
<td><strong>Synonyms</strong></td>
<td><code>ordered_atmost_nvectors</code>, <code>ordered_atmost_npoint</code>, <code>ordered_atmost_npoints</code>.</td>
<td></td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td><code>VECTOR</code> : <code>collection(var=dvar)</code></td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td><code>NVEC</code> : <code>dvar</code> \ <code>VECTORS</code> : <code>collection(vec − VECTOR)</code></td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(</td>
<td>\text{VECTOR}</td>
<td>\geq 1)</td>
</tr>
<tr>
<td>`NVEC \geq \min(1,</td>
<td>\text{VECTORS}</td>
<td>)`</td>
</tr>
<tr>
<td><code>required(\text{VECTORS}.vec)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>same_size(\text{VECTORS}.vec)</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Enforces the following two conditions:

1. The number of distinct tuples of values taken by the vectors of the collection `VECTORS` is less than or equal to `NVEC`. Two tuples of values \(\langle A_1, A_2, \ldots, A_m \rangle\) and \(\langle B_1, B_2, \ldots, B_m \rangle\) are distinct if and only if there exist an integer \(i \in [1, m]\) such that \(A_i \neq B_i\).

2. For each pair of consecutive vectors `VECTOR_i` and `VECTOR_{i+1}` of the `VECTORS` collection we have that `VECTOR_i` is lexicographically less than or equal to `VECTOR_{i+1}`. Given two vectors, \(\vec{X}\) and \(\vec{Y}\) of \(n\) components, \(\langle X_0, \ldots, X_{n-1} \rangle\) and \(\langle Y_0, \ldots, Y_{n-1} \rangle\), \(\vec{X}\) is lexicographically less than or equal to \(\vec{Y}\) if and only if \(n = 0\) or \(X_0 < Y_0\) or \(X_0 = Y_0\) and \(\langle X_1, \ldots, X_{n-1} \rangle\) is lexicographically less than or equal to \(\langle Y_1, \ldots, Y_{n-1} \rangle\).

#### Example

\[
\begin{pmatrix}
\text{vec} - (5, 6), \\
\text{vec} - (5, 6), \\
3, (\text{vec} - (5, 6)), \\
\text{vec} - (9, 3), \\
\text{vec} - (9, 3)
\end{pmatrix}
\]

The `ordered_atmost_nvector` constraint holds since:

1. The collection `VECTORS` involves at most 3 distinct tuples of values (i.e., in fact the 2 distinct tuples \((5, 6)\) and \((9, 3)\)).

2. The vectors of the collection `VECTORS` are sorted in increasing lexicographical order.

#### Typical

\(|\text{VECTOR}| > 1\) \\
`NVEC > 1` \\
`NVEC < |\text{VECTORS}|` \\
\(|\text{VECTORS}| > 1`
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symmetry</td>
<td>NVEC can be increased.</td>
</tr>
<tr>
<td>Arg. properties</td>
<td>Contractible wrt. VECTORS.</td>
</tr>
<tr>
<td>Reformulation</td>
<td>The \texttt{ordered_atmost_nvector} constraint can be reformulated as a conjunction of a \texttt{atmost_nvector} and a \texttt{lex_chain_lesseq} constraints.</td>
</tr>
<tr>
<td>See also</td>
<td>common keyword: \texttt{nvector} (\texttt{vector}). comparison swapped: \texttt{ordered_atleast_nvector}. implied by: \texttt{ordered_nvector} ($\leq \text{NVEC replaced by} = \text{NVEC}$). implies: \texttt{atmost_nvector}, \texttt{lex_chain_lesseq} (NVEC of constraint \texttt{ordered_atmost_nvector} removed). used in graph description: \texttt{lex_less}, \texttt{lex_lesseq}.</td>
</tr>
<tr>
<td>Keywords</td>
<td>characteristic of a constraint: \texttt{vector}. constraint type: counting constraint, order constraint. symmetry: symmetry.</td>
</tr>
<tr>
<td>Arc input(s)</td>
<td>VECTORS</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>Arc generator</td>
<td>$PATH \rightarrow \text{collection}(\text{vectors1}, \text{vectors2})$</td>
</tr>
<tr>
<td>Arc arity</td>
<td>2</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>$\text{lex;lesseq}(\text{vectors1.vec}, \text{vectors2.vec})$</td>
</tr>
<tr>
<td>Graph property(ies)</td>
<td>$\text{NARC} =</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VECTORS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>$PATH \rightarrow \text{collection}(\text{vectors1}, \text{vectors2})$</td>
</tr>
<tr>
<td>Arc arity</td>
<td>2</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>$\text{lex;less}(\text{vectors1.vec}, \text{vectors2.vec})$</td>
</tr>
<tr>
<td>Graph property(ies)</td>
<td>$\text{NCC} \leq \text{NVEC}$</td>
</tr>
</tbody>
</table>

**Graph model**

Parts (A) and (B) of Figure 5.483 respectively show the initial and final graph of the second graph constraint associated with the **Example** slot. Since we use the **NCC** graph property in this second graph constraint, we show the different connected components of the final graph. Each strongly connected component corresponds to a tuple of values that is assigned to some vectors of the **VECTORS** collection. The 2 following tuple of values $(5, 6)$ and $(9, 3)$ are used by the vectors of the **VECTORS** collection.

![Graph Diagram](image)

Figure 5.483: Initial and final graph of the **ordered_atmost_nvector** constraint
5.273 ordered_global_cardinality

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[292]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>ordered_global_cardinality(VARIABLES, VALUES)</td>
<td></td>
</tr>
<tr>
<td>Usual name</td>
<td>order gcc</td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td>order gcc</td>
<td></td>
</tr>
</tbody>
</table>
| Arguments | VARIABLES : collection(var−dvar)  
VALUES : collection(val−int, omax−int) | |
| Restrictions | required(VARIABLES, var)  
| | VALUES > 0  
| | required(VARIABLES, [val, omax])  
| | increasing_seq(VARIABLES, [val])  
| | VALUES.omax ≥ 0  
| | VALUES.omax ≤ | VARIABLES| |
| Purpose | For each \(i \in [1, |VALUES|]\), the values of the corresponding set of values VALUES[j].val \((i \leq j \leq |VALUES|)\) should be taken by at most VALUES[i].omax variables of the VARIABLES collection.  
From that previous definition, the omax attributes are decreasing. | |
| Example | \[
\begin{pmatrix}
(2, 0, 1, 0, 0), \\
\text{val} − 0 \text{ omax} − 5, \\
\text{val} − 1 \text{ omax} − 3, \\
\text{val} − 2 \text{ omax} − 1
\end{pmatrix}
\] | |
| Symmetry | Items of VARIABLES are permutable. | |
| Arg. properties | Contractible wrt. VALUES. | |
| Usage | The ordered_global_cardinality can be used in order to restrict the way we assign the values of the VALUES collection to the variables of the VARIABLES collection. It expresses the fact that, when we use a value \(v\), we implicitly also use all values that are less than or equal to \(v\). As depicted by Figure 5.484 this is for instance the case for a soft cumulative constraint where we want to control the shape of cumulative profile by providing for each instant \(i\) a variable \(h_i\) that gives the height of the cumulative profile at instant \(i\). These variables \(h_i\) are passed as the first argument of the ordered_global_cardinality | |
constraint. Then the \( o_{\text{max}} \) attribute of the \( j \)-th item of the VALUES collection gives the maximum number of instants for which the height of the cumulative profile is greater than or equal to value VALUES\[j\].val. In Figure 5.484 we should have:

- no more than 1 height variable greater than or equal to 2,
- no more than 3 height variables greater than or equal to 1,
- no more than 5 height variables greater than or equal to 0.

\[ \begin{array}{cccc}
2 & 0 & 1 & 0 \\
\hline
h_1 & h_2 & h_3 & h_4 \\
\end{array} \hspace{1cm} \begin{array}{cccc}
2 & 1 & 0 & 0 \\
\hline
h_1 & h_2 & h_3 & h_4 \\
\end{array} \]

(A) (B)

Figure 5.484: (A) Cumulative profile and (B) corresponding height variables

Remark

The original definition of the ordered\_global\_cardinality constraint mentions a third argument, namely the minimum number of occurrences of the smallest value. We omit it since it is redundant.

An other closely related constraint, the cost\_ordered\_global\_cardinality constraint was introduced in [292] in order to model the fact that overloads costs may depend of the instant where they occur.

Algorithm

A filtering algorithm achieving arc-consistency in \( O(|\text{VARIABLES}| + |\text{VALUES}|) \) is described in [292]. It is based on the equivalence between the following two statements:

1. the ordered\_global\_cardinality constraint has a solution,
2. all variables of the VARIABLES collection assigned to their respective minimum value correspond to a solution of the ordered\_global\_cardinality constraint.

Reformulation

The ordered\_global\_cardinality\((\text{var} - V_1, \text{var} - V_2, \ldots, \text{var} - V_{|\text{VARIABLES}|})\), \((\text{val} - v_1 \text{omax} - o_1, \text{val} - v_2 \text{omax} - o_2, \ldots, \text{val} - v_{|\text{VALUES}|} \text{omax} - o_{|\text{VALUES}|})\) constraint can be reformulated into a global\_cardinality\((\text{var} - V_1, \text{var} - V_2, \ldots, \text{var} - V_{|\text{VARIABLES}|})\), \((\text{val} - v_1 \text{noccurrence} - N_1, \text{val} - v_2 \text{noccurrence} - N_2, \ldots, \text{val} - v_{|\text{VALUES}|} \text{noccurrence} - N_{|\text{VALUES}|})\) and \(|\text{VALUES}|\) sliding linear inequalities constraints of the form:

\[ \begin{align*}
N_1 + N_2 + \ldots + N_{|\text{VALUES}|} & \leq o_1, \\
N_2 + \ldots + N_{|\text{VALUES}|} & \leq o_2, \\
& \ldots \ldots \ldots \\
N_{|\text{VALUES}|} & \leq o_{|\text{VALUES}|}.
\end{align*} \]

However, with the next example, T. Petit and J.-C. Régis have shown that this reformulation hinders propagation:

1. \( V_1 \in \{0, 1\}, V_2 \in \{0, 1\}, V_3 \in \{0, 1, 2\}, V_4 \in \{2, 3\}, V_5 \in \{2, 3\}. \)
2. \texttt{global_cardinality}((V_1, V_2, V_3, V_4, V_5), \langle \text{val - 1 nocurrence} - N_1, \\
\text{val - 2 nocurrence} - N_2, \text{val - 3 nocurrence} - N_3 \rangle),

3. \( N_1 + N_2 + N_3 \leq 3 \land N_2 + N_3 \leq 2 \land N_3 \leq 2 \).

The previous reformulation does not remove value 2 from the domain of variable \( V_3 \).

\textbf{See also} \texttt{related: cumulative} (controlling the shape of the cumulative profile for breaking symmetry), \texttt{global_cardinality_low_up, increasing_global_cardinality} (the order is imposed on the main variables, and not on the count variables).

\texttt{root concept: global_cardinality}.

\textbf{Keywords}

\texttt{application area: assignment.}

\texttt{constraint type: value constraint, order constraint.}

\texttt{filtering: arc-consistency.}
For all items of VALUES:

<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>(SELF \leftrightarrow collection(\text{variables}))</td>
</tr>
<tr>
<td>Arc arity</td>
<td>1</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>(\text{variables.var} \geq \text{VALUES.val})</td>
</tr>
<tr>
<td>Graph property(ies)</td>
<td>(\text{NVERTEX} \leq \text{VALUES.\text{omax}})</td>
</tr>
</tbody>
</table>

**Graph model**

Since we want to express one unary constraint for each value we use the “For all items of VALUES” iterator. Part (A) of Figure 5.485 shows the initial graphs associated with each value 0, 1 and 2 of the VALUES collection of the Example slot. Part (B) of Figure 5.485 shows the corresponding final graph associated with value 0. Since we use the \(\text{NVERTEX}\) graph property, the vertices of the final graph is stressed in bold.

Figure 5.485: Initial and final graph of the \texttt{ordered\_global\_cardinality} constraint
### 5.274 **ordered_nvector**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from <code>nvector</code>.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td><code>ordered_nvector(NVEC, VECTORS)</code></td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td><code>ordered_nvectors, ordered_npoint, ordered_npoints</code>.</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td><code>VECTOR : collection(var − dvar)</code></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td><code>NVEC : dvar</code>&lt;br&gt;<code>VECTORS : collection(vec − VECTOR)</code></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>`</td>
<td>VECTOR</td>
</tr>
</tbody>
</table>

Enforces the following two conditions:

1. **NVEC** is the number of distinct tuples of values assigned to the vectors of the collection **VECTORS**. Two tuples of values \(\langle A_1, A_2, \ldots, A_m \rangle\) and \(\langle B_1, B_2, \ldots, B_m \rangle\) are distinct if and only if there exist an integer \(i \in [1, m]\) such that \(A_i \neq B_i\).

2. For each pair of consecutive vectors **VECTOR** and **VECTOR**\(_{i+1}\) of the **VECTORS** collection we have that **VECTOR** is lexicographically less than or equal to **VECTOR**\(_{i+1}\). Given two vectors, \(X\) and \(Y\) of \(n\) components, \(\langle X_0, \ldots, X_{n-1} \rangle\) and \(\langle Y_0, \ldots, Y_{n-1} \rangle\), \(X\) is lexicographically less than or equal to \(Y\) if and only if \(n = 0\) or \(X_0 < Y_0\) or \(X_0 = Y_0\) and \(\langle X_1, \ldots, X_{n-1} \rangle\) is lexicographically less than or equal to \(\langle Y_1, \ldots, Y_{n-1} \rangle\).

**Example**

\[
\begin{pmatrix}
\text{vec} = (5, 6), \\
\text{vec} = (5, 6), \\
2, \\
\text{vec} = (5, 6), \\
\text{vec} = (9, 3), \\
\text{vec} = (9, 3)
\end{pmatrix}
\]

The `ordered_nvector` constraint holds since:

1. Its first argument `NVEC = 2` is set to the number of distinct tuples of values (i.e., tuples \(\langle 5, 6 \rangle\) and \(\langle 9, 3 \rangle\)) occurring within the collection **VECTORS**.

2. The vectors of the collection **VECTORS** are sorted in increasing lexicographical order.
Typical

- |VECTOR| > 1
- NVEC > 1
- NVEC < |VECTORS|
- |VECTORS| > 1

Arg. properties

- **Functional dependency**: NVEC determined by VECTORS.
- **Contractible** wrt. VECTORS when NVEC = 1 and |VECTORS| > 0.
- **Contractible** wrt. VECTORS when NVEC = |VECTORS|.

Reformulation

The ordered_nvector constraint can be reformulated as a conjunction of a nvector and a lex_chain_lesseq constraints.

See also

- **implies**: lex_chain_lesseq(NVEC of constraint ordered_nvector removed), nvector, ordered_atleast_nvector (= NVEC replaced by ≥ NVEC), ordered_atmost_nvector (= NVEC replaced by ≤ NVEC).
- **related**: increasing_nvalue_chain.
- **root concept**: increasing_nvalue.
- used in graph description: lex_less, lex_lesseq.

Keywords

- **characteristic of a constraint**: vector.
- **constraint type**: counting constraint, order constraint.
- **modelling**: functional dependency.
- **symmetry**: symmetry.
Arc input(s) \( \text{VECTORS} \)
Arc generator \( PATH \rightarrow \text{collection}(\text{vectors1}, \text{vectors2}) \)
Arc arity 2
Arc constraint(s) \( \text{lex,lesseq} (\text{vectors1}.\text{vec}, \text{vectors2}.\text{vec}) \)
Graph property(ies) \( \text{NARC} = |\text{VECTORS}| - 1 \)

Arc input(s) \( \text{VECTORS} \)
Arc generator \( PATH \rightarrow \text{collection}(\text{vectors1}, \text{vectors2}) \)
Arc arity 2
Arc constraint(s) \( \text{lex.less} (\text{vectors1}.\text{vec}, \text{vectors2}.\text{vec}) \)
Graph property(ies) \( \text{NCC} = \text{NVEC} \)

Graph model

Parts (A) and (B) of Figure 5.486 respectively show the initial and final graph of the second graph constraint associated with the Example slot. Since we use the NCC graph property in this second graph constraint, we show the different connected components of the final graph. Each strongly connected component corresponds to a tuple of values that is assigned to some vectors of the VECTORS collection. The following tuple of values \( \langle 5, 6 \rangle \) and \( \langle 9, 3 \rangle \) are used by the vectors of the VECTORS collection.

![Graphs](A) and (B) showing initial and final graph of the ordered nvector constraint.

Figure 5.486: Initial and final graph of the ordered nvector constraint
5.275 orth_link_ori_siz_end

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Used by several constraints between orthotopes</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>orth_link_ori_siz_end(ORTHOTOPE)</td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>ORTHOTOPE : collection(ori−dvar, siz−dvar, end−dvar)</td>
<td></td>
</tr>
</tbody>
</table>
| Restrictions| \(|ORTHOTOPE| > 0 \)
|             | require_at_least(2, ORTHOTOPE, [ori, siz]) |
|             | ORTHOTOPE.siz ≥ 0 |
|             | ORTHOTOPE.ori ≤ ORTHOTOPE.end |
| Purpose     | Enforce for each item of the ORTHOTOPE collection the constraint ori + siz = end. |
| Example     | \((ori − 2 siz − 2 end − 4, ori − 1 siz − 3 end − 4)\) |

The orth_link_ori_siz_end constraint holds since the two items \((ori − 2 siz − 2 end − 4)\) and \((ori − 1 siz − 3 end − 4)\) respectively verify the conditions \(2 + 2 = 4\) and \(1 + 3 = 4\).

| Typical     | \(|ORTHOTOPE| > 1 \)
|             | ORTHOTOPE.siz > 0 |
| Symmetries  | • Items of ORTHOTOPE are permutable. |
|             | • One and the same constant can be added to the ori and end attributes of all items of ORTHOTOPE. |
|             | • One and the same constant can be added to the siz and end attributes of all items of ORTHOTOPE. |
| Arg. properties | • Functional dependency: ORTHOTOPE.ori determined by ORTHOTOPE.siz and ORTHOTOPE.end. |
|             | • Functional dependency: ORTHOTOPE.siz determined by ORTHOTOPE.ori and ORTHOTOPE.end. |
|             | • Functional dependency: ORTHOTOPE.end determined by ORTHOTOPE.ori and ORTHOTOPE.siz. |
|             | • Contractible wrt. ORTHOTOPE. |
| Usage       | Used in the Arc constraint(s) slot for defining some constraints like diffn, place_in_pyramid or orths_are_connected. |
| Used in     | diffn, orth_on_the_ground, orth_on_top_of_orth, orths_are_connected, two_orth_are_in_contact, two_orth_column, two_orth_do_not_overlap, two_orth_include. |
Keywords

- **constraint arguments**: pure functional dependency.
- **constraint type**: decomposition.
- **geometry**: orthotope.
- **modelling**: functional dependency.
### Arc input(s)
- **ORTHOTOPE**

### Arc generator
- **SELF** \(\rightarrow\) **collection(orthotope)**

### Arc arity
- 1

### Arc constraint(s)
- \(\text{orthotope.ori} + \text{orthotope.siz} = \text{orthotope.end}\)

### Graph property(ies)
- \(\text{NARC} = |\text{ORTHOTOPE}|\)

### Graph model
Parts (A) and (B) of Figure 5.487 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the loops of the final graph are stressed in bold.

![Graph Model](image)

**Figure 5.487:** Initial and final graph of the **orth_link_ori_siz_end** constraint

### Signature
Since we use the **SELF** arc generator on the **ORTHOTOPE** collection the number of arcs of the initial graph is equal to \(|\text{ORTHOTOPE}|\). Therefore the maximum number of arcs of the final graph is also equal to \(|\text{ORTHOTOPE}|\). For this reason we can rewrite the graph property \(\text{NARC} = |\text{ORTHOTOPE}|\) to \(\text{NARC} \geq |\text{ORTHOTOPE}|\) and simplify **NARC** to **NARC**.
5.276 orth_on_the_ground

**DESCRIPTION**

**Constraint**

$\text{orth_on_the_ground}(\text{ORTHOTOPE}, \text{VERTICAL\_DIM})$

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORTHOTOPE</td>
<td>collection(ori-dvar, siz-dvar, end-dvar)</td>
</tr>
<tr>
<td>VERTICAL_DIM</td>
<td>int</td>
</tr>
</tbody>
</table>

**Restrictions**

- $|\text{ORTHOTOPE}| > 0$
- $\text{require\_at\_least}(2, \text{ORTHOTOPE}, [\text{ori}, \text{siz}, \text{end}])$
- ORTHOTOPE.siz $\geq 0$
- ORTHOTOPE.ori $\leq$ ORTHOTOPE.end
- VERTICAL\_DIM $\geq 1$
- VERTICAL\_DIM $\leq |\text{ORTHOTOPE}|$
- $\text{orth\_link\_ori\_siz\_end}(\text{ORTHOTOPE})$

**Purpose**

The ori attribute of the $\text{VERTICAL\_DIM}^{th}$ item of the ORTHOTOPEs collection should be fixed to one.

**Example**

$$\langle \text{ori} - 1, \text{siz} - 2, \text{end} - 3, \text{ori} - 2, \text{siz} - 3, \text{end} - 5 \rangle, 1$$

The $\text{orth_on_the_ground}$ constraint holds since the ori attribute of its $1^{th}$ item $\langle \text{ori} - 1, \text{siz} - 2, \text{end} - 3 \rangle$ (i.e., $1^{th}$ item since VERTICAL\_DIM = 1) is set to one.

**Typical**

- $|\text{ORTHOTOPE}| > 1$
- ORTHOTOPE.siz $> 0$

**Used in**

place_in_pyramid.

**Keywords**

geometry: geometrical constraint, orthotope.
Arc input(s) | ORTHOTOPE
---|---
Arc generator | SELF→collection(orthotope)
Arc arity | 1
Arc constraint(s) | • orthotope.key = VERTICAL_DIM
• orthotope.ori = 1
Graph property(ies) | NARC = 1

Graph model

Parts (A) and (B) of Figure 5.488 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the loop of the final graph is stressed in bold.

![Diagram](A) ![Diagram](B)

Figure 5.488: Initial and final graph of the orth.on.the.ground constraint

Signature

Since all the key attributes of the ORTHOTOPE collection are distinct, because of the first condition of the arc constraint, and since we use the SELF arc generator the final graph contains at most one arc. Therefore we can rewrite the graph property NARC = 1 to NARC ≥ 1 and simplify NARC to NARC.
5.277 orth_on_top_of_orth

**DESCRIPTION**

Origin

Used for defining place_in_pyramid.

Constraint

orth_on_top_of_orth(ORTHOTOPE1, ORTHOTOPE2, VERTICAL_DIM)

**LINKS**

Type

ORTHOTOPE : collection(ori-dvar, siz-dvar, end-dvar)

Arguments

ORTHOTOPE1 : ORTHOTOPE
ORTHOTOPE2 : ORTHOTOPE
VERTICAL_DIM : int

**GRAPH**

Restrictions

|ORTHOTOPE| > 0
require_at_least(2, ORTHOTOPE, [ori, siz, end])
ORTHOTOPE.ori ≤ ORTHOTOPE.end
ORTHOTOPE.siz ≥ 0
ORTHOTOPE.siz ≥ 0
ORTHOTOPE.ori ≤ ORTHOTOPE.end
|ORTHOTOPE1| = |ORTHOTOPE2|
VERTICAL_DIM ≥ 1
VERTICAL_DIM ≤ |ORTHOTOPE1|
ORTH_LINK.ori_siz_end(ORTHOTOPE1)
ORTH_LINK.ori_siz_end(ORTHOTOPE2)

**Purpose**

ORTHOTOPE1 is located on top of ORTHOTOPE2 which concretely means:

- In each dimension different from VERTICAL_DIM the projection of ORTHOTOPE1 is included in the projection of ORTHOTOPE2.
- In the dimension VERTICAL_DIM the origin of ORTHOTOPE1 coincide with the end of ORTHOTOPE2.

**Example**

\[
\begin{pmatrix}
(ori - 5, siz - 2, end - 7, ori - 3, siz - 3, end - 6), \\
(ori - 3, siz - 5, end - 8, ori - 1, siz - 2, end - 3), 2
\end{pmatrix}
\]

As illustrated by Figure 5.489 the orthotope ORTHOTOPE1 (rectangle R1 coloured in pink) is on top of ORTHOTOPE2 (rectangle R2 coloured in blue) according to the hypothesis that the vertical dimension corresponds to dimension 2 (i.e., VERTICAL_DIM = 2). This stands from the fact that the following conditions hold:


Consequently, the orth_on_top_of_orth constraint holds.

**Typical**

|ORTHOTOPE| > 1
ORTHOTOPE.siz > 0
Used in: place_in_pyramid.

Keywords: constraint type: logic.
geometry: geometrical constraint, non-overlapping, orthotope.
Figure 5.489: Illustration of the relation on top of
Arc input(s)  ORTHOTOPE1 ORTHOTOPE2
Arc generator  \( PRODUCT(=) \rightarrow collection(orthotope1,orthotope2) \)
Arc arity  2
Arc constraint(s)  
  \* orthotope1.key \neq VERTICAL_DIM  
  \* orthotope2.ori \leq orthotope1.ori  
  \* orthotope1.end \leq orthotope2.end  
Graph property(ies)  \( \text{NARC} = |ORTHOTOPE1| - 1 \)

Arc input(s)  ORTHOTOPE1 ORTHOTOPE2
Arc generator  \( PRODUCT(=) \rightarrow collection(orthotope1,orthotope2) \)
Arc arity  2
Arc constraint(s)  
  \* orthotope1.key = VERTICAL_DIM  
  \* orthotope1.ori = orthotope2.end  
Graph property(ies)  \( \text{NARC} = 1 \)

Graph model
The first and second graph constraints respectively express the first and second conditions stated in the **Purpose** slot defining the orth on _top_of_ orth constraint.

Parts (A) and (B) of Figure 5.490 respectively show the initial and final graph associated with the second graph constraint of the **Example** slot. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold.

![Figure 5.490: Initial and final graph of the orth on _top_of_ orth constraint](image)

**Signature**
Consider the second graph constraint. Since all the key attributes of the ORTHOTOPE1 collection are distinct, because of the arc constraint orthotope1.key = VERTICAL_DIM, and since we use the \( PRODUCT(=) \) arc generator the final graph contains at most one arc. Therefore we can rewrite the graph property \( \text{NARC} = 1 \) to \( \text{NARC} \geq 1 \) and simplify \( \text{NARC} \) to \( \text{NARC} \).
### 5.278 orths_are_connected

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>N. Beldiceanu</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>orths_are_connected(ORTHOTOPES)</td>
<td></td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>ORTHOTOPE : collection(ori−dvar,siz−dvar,end−dvar)</td>
<td></td>
</tr>
<tr>
<td><strong>Argument</strong></td>
<td>ORTHOTOPES : collection(orth−ORTHOTOPE)</td>
<td></td>
</tr>
</tbody>
</table>
| **Restrictions** | | [ORTHOTOPE] > 0  
 require_at_least(2, ORTHOTOPE, [ori,siz,end])  
 ORTHOTOPE.siz > 0  
 ORTHOTOPE.ori ≤ ORTHOTOPE.end  
 required(ORTHOTOPES, orth)  
 same_size(ORTHOTOPES, orth) | |
| **Purpose** | There should be one single group of connected orthotopes. Two orthotopes touch each other (i.e., are connected) if they overlap in all dimensions except one, and if, for the dimension where they do not overlap, the distance between the two orthotopes is equal to 0. | |
| **Example** | \[
\begin{align*}
\text{orth} & - \langle \text{ori} - 2, \text{siz} - 4, \text{end} - 6, \rangle \\
\text{orth} & - \langle \text{ori} - 1, \text{siz} - 2, \text{end} - 3, \rangle \\
\text{orth} & - \langle \text{ori} - 4, \text{siz} - 3, \text{end} - 7, \rangle \\
\text{orth} & - \langle \text{ori} - 7, \text{siz} - 4, \text{end} - 11, \rangle \\
\text{orth} & - \langle \text{ori} - 1, \text{siz} - 2, \text{end} - 3, \rangle \\
\text{orth} & - \langle \text{ori} - 6, \text{siz} - 2, \text{end} - 8, \rangle \\
\text{orth} & - \langle \text{ori} - 3, \text{siz} - 2, \text{end} - 5, \rangle \\
\end{align*}
\] | Figure 5.491 shows the rectangles associated with the example. One can note that:  
- Rectangle 2 touch rectangle 1,  
- Rectangle 1 touch rectangle 2 and rectangle 4,  
- Rectangle 4 touch rectangle 1 and rectangle 3,  
- Rectangle 3 touch rectangle 4.  
Consequently, since we have one single group of connected rectangles, the orths_are_connected constraint holds. |
| **Typical** | [ORTHOTOPE] > 1  
 [ORTHOTOPES] > 1 | |
Symmetries

- Items of ORTHOTOPES are permutable.
- Items of ORTHOTOPES.orth are permutable (same permutation used).
- One and the same constant can be added to the ori and end attributes of all items of ORTHOTOPES.orth.

Usage

In floor planning problem there is a typical constraint, that states that one should be able to access every room from any room.

See also

implies: diffn.

used in graph description: orth_link_ori.siz.end, two_orth_are_in_contact.

Keywords

geometry: geometrical constraint, touch, contact, non-overlapping, orthotope.

Figure 5.491: Four connected rectangles
Arc input(s) \hspace{1cm} \text{ORTHOTOPES}
Arc generator \hspace{1cm} \text{SELF} \rightarrow \text{collection(orthotopes)}
Arc arity \hspace{1cm} 1
Arc constraint(s) \hspace{1cm} \text{orth_link_ori_siz_end(orthotopes.orth)}
Graph property(ies) \hspace{1cm} \text{NARC} = |\text{ORTHOTOPES}|

Arc input(s) \hspace{1cm} \text{ORTHOTOPES}
Arc generator \hspace{1cm} \text{CLIQUE}(\neq) \rightarrow \text{collection(orthotopes1,orthotopes2)}
Arc arity \hspace{1cm} 2
Arc constraint(s) \hspace{1cm} \text{two_orth_are_in_contact(orthotopes1.orth,orthotopes2.orth)}
Graph property(ies) \hspace{1cm} \begin{align*}
\bullet \text{ NVERTEX} & = |\text{ORTHOTOPES}| \\
\bullet \text{ NCC} & = 1
\end{align*}

Graph model

Parts (A) and (B) of Figure 5.492 respectively show the initial and final graph associated with the Example slot. Since we use the NVERTEX graph property the vertices of the final graph are stressed in bold. Since we also use the NCC graph property we show the unique connected component of the final graph. An arc between two vertices indicates that two rectangles are in contact.

Figure 5.492: Initial and final graph of the orths_are_connected constraint

Signature

Since the first graph constraint uses the SELF arc generator on the ORTHOTOPES collection the corresponding initial graph contains $|\text{ORTHOTOPES}|$ arcs. Therefore the final graph of the first graph constraint contains at most $|\text{ORTHOTOPES}|$ arcs and we can rewrite $\text{NARC} = |\text{ORTHOTOPES}|$ to $\text{NARC} \geq |\text{ORTHOTOPES}|$. So we can simplify $\text{NARC}$ to $\text{NARC}$.

Consider now the second graph constraint. Since its corresponding initial graph contains $|\text{ORTHOTOPES}|$ vertices, its final graph has a maximum number of vertices also
equal to $|\text{ORTHOTOPES}|$. Therefore we can rewrite $N_{\text{VERTEX}} = |\text{ORTHOTOPES}|$ to $N_{\text{VERTEX}} \geq |\text{ORTHOTOPES}|$ and simplify $N_{\text{VERTEX}}$ to $N_{\text{VERTEX}}$. From the graph property $N_{\text{VERTEX}} = |\text{ORTHOTOPES}|$ and from the restriction $|\text{ORTHOTOPES}| > 0$ the final graph is not empty. Therefore it contains at least one connected component. So we can rewrite $N_{CC} = 1$ to $N_{CC} \leq 1$ and simplify $N_{CC}$ to $N_{CC}$. 
5.279  overlap_sboxes

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>LOGIC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometry, derived from [318]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>overlap_sboxes(K, DIMS, OBJECTS, SBOXES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Synonym</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>overlap.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Types</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARIABLES : collection(v−dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTEGERS : collection(v−int)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POSITIVES : collection(v−int)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K : int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIMS : sint</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OBJECTS : collection(oid−int,sid−int,x − VARIABLES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBOXES : collection(sid−int,t − INTEGERS,l − POSITIVES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| \[ | VARIABLES | \geq | 1 | \\
| INTEGERS | \geq | 1 | \\
| POSITIVES | \geq | 1 | \\
| required(VARIABLES,v) | | | |
| \[ | VARIABLES | = | K | \\
| INTEGERS | = | K | \\
| POSITIVES | = | K | \\
| POSITIVES.v | > | 0 | \\
| K | > | 0 | \\
| DIMS | \geq | 0 | \\
| DIMS | < | K | \\
| increasing_seq(OBJECTS,[oid]) | | | |
| required(OBJECTS,[oid,sid,x]) | | | |
| OBJECTS.oid | \geq | 1 | \\
| OBJECTS.oid | \leq | | \text{OBJECTS} | \\
| OBJECTS.sid | \geq | 1 | \\
| OBJECTS.sid | \leq | | \text{SBOXES} | \\
| SBOXES | \geq | 1 | \\
| required(SBOXES,[sid,t,l]) | | | |
| SBOXES.sid | \geq | 1 | \\
| SBOXES.sid | \leq | | \text{SBOXES} | \\
| do_not_overlap(SBOXES) | | | |
Holds if, for each pair of objects \((O_i, O_j)\), \(i < j\), \(O_i\) overlaps \(O_j\) with respect to a set of dimensions depicted by \(\text{DIMS}\). \(O_i\) and \(O_j\) are objects that take a shape among a set of shapes. Each \textit{shape} is defined as a finite set of shifted boxes, where each shifted box is described by a box in a \(K\)-dimensional space at a given offset (from the origin of the shape) with given sizes. More precisely, a \textit{shifted box} is an entity defined by its shape \(\text{id}\), shift offset \(\text{t}\), and sizes \(l\). Then, a shape is defined as the union of shifted boxes sharing the same shape id. An \textit{object} is an entity defined by its unique object identifier \(\text{oid}\), shape \(\text{id}\) \(\text{sid}\) and origin \(x\).

An object \(O_i\) \textit{overlaps} an object \(O_j\) with respect to a set of dimensions depicted by \(\text{DIMS}\) if and only if, there exists a shifted box \(s_i\) associated with \(O_i\) and there exists a shifted box \(s_j\) associated with \(O_j\), such that (1) there exists a dimension \(d \in \text{DIMS}\) where the end of \(O_i\) in dimension \(d\) is strictly greater than the start of \(O_j\) in dimension \(d\), and (2) the end of \(O_j\) in dimension \(d\) is strictly greater than the start of \(O_i\) in dimension \(d\).

**Example**

\[
\begin{pmatrix}
2, \{0, 1\}, \\
\langle \text{oid} - 1, \text{sid} - 1 \rangle \langle x - 1, 1 \rangle, \\
\langle \text{oid} - 2, \text{sid} - 2 \rangle \langle x - 3, 2 \rangle, \\
\langle \text{oid} - 3, \text{sid} - 3 \rangle \langle x - 2, 4 \rangle, \\
\langle \text{sid} - 1 \rangle \langle \text{t} - 0, 0 \rangle \langle l - 4, 5 \rangle, \\
\langle \text{sid} - 2 \rangle \langle \text{t} - 0, 0 \rangle \langle l - 3, 3 \rangle, \\
\langle \text{sid} - 3 \rangle \langle \text{t} - 0, 0 \rangle \langle l - 2, 1 \rangle
\end{pmatrix}
\]

Figure 5.493 shows the objects of the example. Since \(O_1\) overlaps both \(O_2\) and \(O_3\), and since \(O_2\) overlaps \(O_3\), the overlap_sboxes constraint holds.

Figure 5.493: The three objects of the example

\(\left|\text{OBJECTS}\right| > 1\)
Symmetries

- Items of OBJECTS are **permutable**.
- Items of SBOXES are **permutable**.
- Items of OBJECTS.x, SBOXES.t and SBOXES.1 are **permutable** (same permutation used).
- SBOXES.1.v can be **increased**.

Arg. properties

**Suffix-contractible** wrt. OBJECTS.

Remark

One of the eight relations of the *Region Connection Calculus* [318].

See also

**common keyword**: contains_sboxes, coveredby_sboxes, covers_sboxes, disjoint_sboxes, equal_sboxes, inside_sboxes, meet_sboxes (rcc8), non_overlap_sboxes (*geometrical constraint, logic*).

Keywords

**constraint type**: logic.

**geometry**: geometrical constraint, rcc8.
Logic

- **origin**$(O_1, S_1, D) \ \overset{\text{def}}{=} \ O_1.x(D) + S_1.t(D)$
- **end**$(O_1, S_1, D) \ \overset{\text{def}}{=} \ O_1.x(D) + S_1.t(D) + S_1.1(D)$
- **overlap_sboxes**$(\text{Dims}, O_1, S_1, O_2, S_2) \ \overset{\text{def}}{=} \ \\
\forall D \in \text{Dims} \\
\left( \begin{array}{c}
\text{end}(O_1, S_1, D) > \\
\text{origin}(O_2, S_2, D) \\
\text{end}(O_2, S_2, D) > \\
\text{origin}(O_1, S_1, D)
\end{array} \right)$
- **overlap_objects**$(\text{Dims}, O_1, O_2) \ \overset{\text{def}}{=} \ \\
\forall S_1 \in \text{sboxes}([O_1.\text{sid}]) \\
\exists S_2 \in \text{sboxes}(\ [O_2.\text{sid} \ ] ) \\
\text{overlap_sboxes}(\ Dims, \ O_1, \ S_1, \ O_2, \ S_2)$
- **all_overlap**$(\text{Dims}, \text{OIDS}) \ \overset{\text{def}}{=} \ \\
\forall O_1 \in \text{objects(OIDS)} \\
\forall O_2 \in \text{objects(OIDS)} \\
O_1.\text{oid} < \Rightarrow \\
O_2.\text{oid} \\
\text{overlap_objects}(\ Dims, \ O_1, \ O_2)$
- **all_overlap**$(\text{DIMENSIONS}, \text{OIDS})$
5.280 path

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from binary.tree.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>path(NPATH, NODES)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>NPATH : dvar</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES : collection(index-int, succ-dvar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>NPATH ≥ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NPATH ≤</td>
<td>NODES</td>
</tr>
<tr>
<td></td>
<td>required(NODES, [index, succ])</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NODES</td>
</tr>
<tr>
<td></td>
<td>NODES.index ≥ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.index ≤</td>
<td>NODES</td>
</tr>
<tr>
<td></td>
<td>distinct(NODES, index)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.succ ≥ 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NODES.succ ≤</td>
<td>NODES</td>
</tr>
</tbody>
</table>

Purpose

Cover the digraph G described by the NODES collection with NPATH paths in such a way that each vertex of G belongs to exactly one path.

Example

The path constraint holds since its second argument corresponds to the 3 (i.e., the first argument of the path constraint) paths depicted by Figure 5.494.

```
3, {
  index - 1 succ - 1,
  index - 2 succ - 3,
  index - 3 succ - 5,
  index - 4 succ - 7,
  index - 5 succ - 1,
  index - 6 succ - 6,
  index - 7 succ - 7,
  index - 8 succ - 6
}
```

Figure 5.494: The three paths corresponding to the Example slot
Typical

\[ \text{NPATH} < |\text{NODES}| \]
\[ |\text{NODES}| > 1 \]

Symmetry

Items of NODES are permutable.

Arg. properties

Functional dependency: NPATH determined by NODES.

Reformulation

The path constraint can be expressed in terms of (1) a set of $|\text{NODES}|^2$ reified constraints for avoiding circuit between more than one node and of (2) $|\text{NODES}|$ reified constraints and of one sum constraint for counting the paths and of (3) a set of $|\text{NODES}|$ reified constraints and of $|\text{NODES}|$ inequalities constraints for enforcing the fact that each vertex has at most two children.

1. For each vertex NODES[i] ($i \in [1, |\text{NODES}|]$) of the NODES collection we create a variable $R_i$ that takes its value within interval [1, |NODES|]. This variable represents the rank of vertex NODES[i] within a solution. It is used to prevent the creation of circuit involving more than one vertex as explained now. For each pair of vertices NODES[i], NODES[j] ($i, j \in [1, |\text{NODES}|]$) of the NODES collection we create a reified constraint of the form $\text{NODES}[i].\text{succ} = \text{NODES}[j].\text{index} \land i \neq j \Rightarrow R_i < R_j$. The purpose of this constraint is to express the fact that, if there is an arc from vertex NODES[i] to another vertex NODES[j], then $R_i$ should be strictly less than $R_j$.

2. For each vertex NODES[i] ($i \in [1, |\text{NODES}|]$) of the NODES collection we create a 0-1 variable $B_i$ and state the following reified constraint \( \text{NODES}[i].\text{succ} = \text{NODES}[i].\text{index} \iff B_i \) in order to force variable $B_i$ to be set to value 1 if and only if there is a loop on vertex NODES[i]. Finally we create a constraint $\text{NPATH} = B_1 + B_2 + \ldots + B_{|\text{NODES}|}$ for stating the fact that the number of paths is equal to the number of loops of the graph.

3. For each pair of vertices NODES[i], NODES[j] ($i, j \in [1, |\text{NODES}|]$) of the NODES collection we create a 0-1 variable $B_{ij}$ and state the following reified constraint \( \text{NODES}[i].\text{succ} = \text{NODES}[j].\text{index} \iff B_{ij} \). Variable $B_{ij}$ is set to value 1 if and only if there is an arc from NODES[i] to NODES[j]. Then for each vertex NODES[j] ($j \in [1, |\text{NODES}|]$) we create a constraint of the form $B_{1j} + B_{2j} + \ldots + B_{|\text{NODES}|j} \leq 2$.

See also

- common keyword: circuit (graph partitioning constraint, one succ), dom_reachability (path), path_from_to (path, select an induced subgraph so that there is a path from a given vertex to an other given vertex).
- generalisation: binary_tree (at most one child replaced by at most two children), temporal_path (vertices are located in time, and to each arc corresponds a precedence constraint), tree (at most one child replaced by no limit on the number of children).
- implies: binary_tree.
- related: balance_path (counting number of paths versus controlling how balanced the paths are).

Keywords

- combinatorial object: path.
- constraint type: graph constraint, graph partitioning constraint.
- filtering: DFS-bottleneck.
- final graph structure: connected component, tree, one_suc.
Arc input(s) | NODES
---|---
Arc generator | CLIQUE"→collection(nodes1, nodes2)
Arc arity | 2
Arc constraint(s) | nodes1.succ = nodes2.index
Graph property(ies) | • MAX_NSCC ≤ 1
 | • NCC = NPATH
 | • MAX_ID ≤ 1
Graph class | ONE_SUCC

Graph model

We use the same graph constraint as for the binary_tree constraint, except that we replace
the graph property \( \text{MAX.ID} \leq 2 \), which constraints the maximum in-degree of the final
graph to not exceed 2 by \( \text{MAX.ID} \leq 1 \). MAX_ID does not consider loops: This is why
we do not have any problem with the final node of each path.

Parts (A) and (B) of Figure 5.495 respectively show the initial and final graph associated
with the Example slot. Since we use the NCC graph property, we display the three
connected components of the final graph. Each of them corresponds to a path. Since we
use the MAX_ID graph property, we also show with a double circle a vertex that has a
maximum number of predecessors.

The path constraint holds since all strongly connected components of the final graph have
no more than one vertex, since \( \text{NPATH} = \text{NCC} = 3 \) and since \( \text{MAX.ID} = 1 \).
Figure 5.495: Initial and final graph of the path constraint
### 5.281 path_from_to

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[4]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>path_from_to(FROM, TO, NODES)</td>
<td></td>
</tr>
<tr>
<td>Usual name</td>
<td>path</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>FROM : int, TO : int, NODES : collection(index=int, succ=svar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>FROM $\geq 1$, FROM $\leq</td>
<td>NODES</td>
</tr>
<tr>
<td>Purpose</td>
<td>Select some arcs of a digraph $G$ so that there is still a path between two given vertices of $G$.</td>
<td></td>
</tr>
</tbody>
</table>
| Example     | \[
\begin{align*}
\text{index} - 1 & : \text{succ} - \emptyset, \\
\text{index} - 2 & : \text{succ} - \emptyset, \\
4, 3 & : \text{succ} - \{5\}, \\
\text{index} - 4 & : \text{succ} - \{5\}, \\
\text{index} - 5 & : \text{succ} - \{2, 3\}
\end{align*}
\] |       |
| Typical     | FROM $\neq$ TO, | | |
| Symmetry    | Items of NODES are permutable. | | |
| See also    | common keyword: dom_reachability(path), link_set_to_booleans(constraint involving set variables), path, temporal_path(path). | used in graph description: in_set. |
Keywords

combinatorial object: path.
constraint arguments: constraint involving set variables.
constraint type: graph constraint.
filtering: linear programming.
**Arc input(s)**

NODES

**Arc generator**

$\text{CLIQUE} \rightarrow \text{collection}(\text{nodes}_1, \text{nodes}_2)$

**Arc arity**

2

**Arc constraint(s)**

in_set(\text{nodes}_2.\text{index}, \text{nodes}_1.\text{succ})

**Graph property(ies)**

$\text{PATH\_FROM\_TO}(\text{index}, \text{FROM}, \text{TO}) = 1$

**Graph model**

Within the context of the **Example** slot, part (A) of Figure 5.496 shows the initial graph from which we choose to start. It is derived from the set associated with each vertex. Each set describes the potential values of the $\text{succ}$ attribute of a given vertex. Part (B) of Figure 5.496 gives the final graph associated with the **Example** slot. Since we use the $\text{PATH\_FROM\_TO}$ graph property we show on the final graph the following information:

- The vertices that respectively correspond to the start and the end of the required path are stressed in bold.
- The arcs on the required path are also stressed in bold.

The $\text{PATH\_FROM\_TO}$ constraint holds since there is a path from vertex 4 to vertex 3 (4 and 3 refer to the $\text{index}$ attribute of a vertex).

![Graph Model](image)

**Signature**

Since the maximum value returned by the graph property $\text{PATH\_FROM\_TO}$ is equal to 1 we can rewrite $\text{PATH\_FROM\_TO}(\text{index}, \text{FROM}, \text{TO}) = 1$ to $\text{PATH\_FROM\_TO}(\text{index}, \text{FROM}, \text{TO}) \geq 1$. Therefore we simplify $\text{PATH\_FROM\_TO}$ to $\text{PATH\_FROM\_TO}$.
5.282 pattern

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[78]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>pattern(VARIABLES, PATTERNS)</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>PATTERN : collection(var-int)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VARIABLES : collection(var-dvar)</td>
<td>PATTERNS : collection(pat - PATTERN)</td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(PATTERN.var)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PATTERN.var ≥ 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>change(0, PATTERN, =)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[PATTERN] &gt; 1</td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES.var)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>required(PATTERNS.pat)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[PATTERNS] &gt; 0</td>
</tr>
<tr>
<td></td>
<td>same_size(PATTERNS.pat)</td>
<td></td>
</tr>
</tbody>
</table>

We quote the definition from the original article [78, page 157] introducing the pattern constraint:

“We call a k-pattern (k > 1) any sequence of k elements such that no two successive elements have the same value. Consider a set \( V = \{v_1, v_2, \ldots, v_n\} \) and a sequence \( s = s_1 s_2 \ldots s_n \) of elements of \( V \). In this context, a stretch is a maximum subsequence of variables of \( s \) which all have the same value. Consider now the sequence \( v_i v_{i+1} \ldots v_{i+k-1} \) of the types of the successive stretches that appear in \( s \). Let \( P \) be a set of k-patterns. \( s \) satisfies \( P \) if and only if every subsequence of k elements in \( v_i v_{i+1} \ldots v_{i+k-1} \) belongs to \( P \).”

Example

\[
\begin{pmatrix}
\text{var - 1,} \\
\text{var - 1,} \\
\text{var - 2,} \\
\text{var - 2,} \\
\text{var - 2,} \\
\text{var - 1,} \\
\text{var - 3,} \\
\text{var - 3} \\
\text{pat - (1,2,1),} \\
\text{pat - (1,2,3),} \\
\text{pat - (2,1,3)}
\end{pmatrix}
\]

The pattern constraint holds since, as depicted by Figure 5.497, all its sequences of three consecutive stretches correspond to one of the 3-pattern given in the PATTERNS collection.
Typical

\[ |\text{VARIABLES}| > 2 \]
\[ \text{range(\text{VARIABLES}.\text{var})} > 1 \]

Symmetries

- Items of \text{PATTERNS} are permutable.
- Items of \text{VARIABLES} and \text{PATTERNS}.\text{pat} are simultaneously reversible.
- All occurrences of two distinct tuples of values in \text{VARIABLES}.\text{var} or \text{PATTERNS}.\text{pat}.\text{var} can be swapped; all occurrences of a tuple of values in \text{VARIABLES}.\text{var} or \text{PATTERNS}.\text{pat}.\text{var} can be renamed to any unused tuple of values.

Arg. properties

- Prefix-contractible wrt. \text{VARIABLES}.
- Suffix-contractible wrt. \text{VARIABLES}.

Usage

The pattern constraint was originally introduced within the context of staff scheduling. In this context, the value of the \(i^{th}\) variable of the \text{VARIABLES} collection corresponds to the type of shift performed by a person on the \(i^{th}\) day. A stretch is a maximum sequence of consecutive variables that are all assigned to the same value. The pattern constraint imposes that each sequence of \(k\) consecutive stretches belongs to a given list of patterns.

Remark

A generalisation of the pattern constraint to the regular constraint enforcing the fact that a sequence of variables corresponds to a regular expression is presented in [286].

See also

- **common keyword:** group (timetabling constraint), sliding_distribution (sliding sequence constraint), stretch_circuit, stretch_path (sliding sequence constraint, timetabling constraint), stretch_path_partition (sliding sequence constraint).

Keywords

- characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.
- constraint network structure: Berge-acyclic constraint network.
- constraint type: timetabling constraint, sliding sequence constraint,
- filtering: arc-consistency.
Figure 5.497: The sequence of the Example slot, its four stretches and the corresponding two 3-patterns 1 2 1 and 2 1 3
Automaton

Taking advantage of the fact that all $k$-patterns have the same length $k$, it is straightforward to construct an automaton that only accepts solutions of the pattern constraint. Figure 5.498 depicts the automaton associated with the pattern constraint of the Example slot. The construction can be done in three steps:

- First, build a prefix tree of all the $k$-patterns. In the context of our example, this gives all arcs of Figure 5.498, except self loops and the arc from $s_3$ to $s_7$.
- Second, find out the transitions that exit a leave of the tree. For this purpose we remove the first symbol of the corresponding $k$-pattern, add at the end of the remaining $k$-pattern a symbol corresponding to a stretch value, and check whether the new pattern belongs or not to the set of $k$-patterns of the pattern constraint. When the new pattern belongs to the set of $k$-patterns we add a corresponding transition. For instance, in the context of our example, consider the leave $s_3$ that is associated with the 3-pattern $1,2,1$. We remove the first symbol 1 and get 2, 1. We then try to successively add the stretch values 1, 2 and 3 to the end of 2, 1 and check if the corresponding patterns $2,1,1$, $2,1,2$, and $2,1,3$ belong or not to our set of 3-patterns. Since only 2, 1, 3 is a 3-pattern we add a new transition between the corresponding leaves of the prefix tree (i.e., a transition from $s_3$ to $s_7$).
- Third, in order to take into account the fact that each value of a $k$-pattern corresponds in fact to a given stretch value (i.e., several consecutive values that are assigned the same value), we add a self loop to all non-source states with a transition label that corresponds to the transition label of their entering arc.

![Automaton of the pattern constraint of the Example slot](image)

Figure 5.498: Automaton of the pattern constraint of the Example slot
## 5.283 peak

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from <em>inflexion</em>.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>( \text{peak}(N, \text{VARIABLES}) )</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>( N : \text{dvar} ) ( \text{VARIABLES} : \text{collection}(\text{var} - \text{dvar}) )</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>( N \geq 0 ) ( 2 \times N \leq \max(</td>
<td>\text{VARIABLES}</td>
</tr>
</tbody>
</table>

### Purpose

A variable \( V_k (1 < k < m) \) of the sequence of variables \( \text{VARIABLES} = V_1, \ldots, V_m \) is a peak if and only if there exists an \( i (1 < i \leq k) \) such that \( V_{i-1} < V_i \) and \( V_i = V_{i+1} = \ldots = V_k \) and \( V_k > V_{k+1} \). \( N \) is the total number of peaks of the sequence of variables \( \text{VARIABLES} \).

### Example

\[
\left( \begin{array}{c}
\text{var} - 1, \\
\text{var} - 1, \\
\text{var} - 4, \\
\text{var} - 8, \\
\text{var} - 6, \\
\text{var} - 2, \\
\text{var} - 7, \\
\text{var} - 1
\end{array} \right)
\]

The peak constraint holds since the sequence 1 1 4 8 6 2 7 1 contains two peaks that respectively correspond to the variables that are assigned to values 8 and 7.

![Graph showing the sequence and its two peaks](image-url)

**Figure 5.499:** The sequence and its two peaks
Typical
\[ |\text{VARIABLES}| > 2 \]
\[ \text{range}(\text{VARIABLES}.\text{var}) > 1 \]

Symmetries
- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties
Contractible wrt. VARIABLES when \( N = 0 \).

Usage
Useful for constraining the number of peaks of a sequence of domain variables.

Remark
Since the arity of the arc constraint is not fixed, the peak constraint cannot be currently described. However, this would not hold anymore if we were introducing a slot that specifies how to merge adjacent vertices of the final graph.

See also
common keyword: highest_peak, inflexion (sequence).
comparison swapped: valley.
related: no_valley.
specialisation: no_peak (the variable counting the number of peaks is set to 0 and removed).

Keywords
characteristic of a constraint: automaton, automaton with counters.
combinatorial object: sequence.
constraint network structure: sliding cyclic(1) constraint network(2).
Automaton

Figure 5.500 depicts the automaton associated with the peak constraint. To each pair of consecutive variables \((\text{VAR}_i, \text{VAR}_{i+1})\) of the collection VARIABLES corresponds a signature variable \(S_i\). The following signature constraint links \(\text{VAR}_i\), \(\text{VAR}_{i+1}\) and \(S_i\): \((\text{VAR}_i > \text{VAR}_{i+1} \Leftrightarrow S_i = 0) \land (\text{VAR}_i = \text{VAR}_{i+1} \Leftrightarrow S_i = 1) \land (\text{VAR}_i < \text{VAR}_{i+1} \Leftrightarrow S_i = 2)\).

Figure 5.500: Automaton of the peak constraint

Figure 5.501: Hypergraph of the reformulation corresponding to the automaton of the peak constraint
5.284 period

**DESCRIPTION**

**Constraint**

period(PERIOD, VARIABLES, CTR)

**Arguments**

<table>
<thead>
<tr>
<th>PERIOD</th>
<th>dvar</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARIABLES</td>
<td>collection(var−dvar)</td>
</tr>
<tr>
<td>CTR</td>
<td>atom</td>
</tr>
</tbody>
</table>

**Restrictions**

- PERIOD ≥ 1
- PERIOD ≤ |VARIABLES|
- required(VARIABLES, var)
- CTR ∈ [=, ≠, <, ≥, >, ≤]

**Purpose**

Let us note $V_0, V_1, \ldots, V_{m-1}$ the variables of the VARIABLES collection. **PERIOD** is the period of the sequence $V_0 V_1 \ldots V_{m-1}$ according to constraint **CTR**. This means that **PERIOD** is the smallest natural number such that $V_i \text{CTR} V_{i+\text{PERIOD}}$ holds for all $i ∈ 0, 1, \ldots, m − \text{PERIOD} − 1$.

**Example**

\[
\begin{pmatrix}
\text{var} - 1, \\
\text{var} - 1, \\
\text{var} - 4, \\
\text{var} - 1, \\
\text{var} - 1, \\
\text{var} - 4, \\
\text{var} - 1, \\
\text{var} - 1 \\
\end{pmatrix}, =
\]

The period constraint holds since, as depicted by Figure 5.502, its first argument PERIOD = 3 is equal (i.e., since **CTR** is set to =) to the period of the sequence 1 1 4 1 1 4 1 1.

Figure 5.502: A sequence that has a period of 3

**Typical**

- PERIOD > 1
- PERIOD < |VARIABLES|
- |VARIABLES| > 2
- range(VARIABLES.var) > 1
- CTR ∈ [=]
Symmetries

- Items of VARIABLES can be reversed.
- Items of VARIABLES can be shifted.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

Arg. properties

- Functional dependency: PERIOD determined by VARIABLES and CTR.
- Contractible wrt. VARIABLES when CTR ∈ [=] and PERIOD = 1.
- Prefix-contractible wrt. VARIABLES.
- Suffix-contractible wrt. VARIABLES.

Algorithm

When CTR corresponds to the equality constraint, a potentially incomplete filtering algorithm based on 13 deductions rules is described in [52]. The generalisation of these rules to the case where CTR is not the equality constraint is discussed.

See also

- generalisation: period_vectors (variable replaced by vector).
- implies: period_except_0.
- soft variant: period_except_0 (value 0 can match any other value).

Keywords

- combinatorial object: periodic, sequence.
- constraint arguments: pure functional dependency.
- constraint type: predefined constraint, timetabling constraint, scheduling constraint.
- filtering: border.
5.285  **period_except_0**

### DESCRIPTION

**Origin**
Derived from `period`.

**Constraint**
`period_except_0(PERIOD, VARIABLES, CTR)`

**Arguments**
- **PERIOD**: `dvar`
- **VARIABLES**: `collection(var−dvar)`
- **CTR**: `atom`

**Restrictions**
- `PERIOD ≥ 1`
- `PERIOD ≤ |VARIABLES|`
- `required(VARIABLES, var)`
- `CTR ∈ [=, ≠, <, ≥, >, ≤]`

### Purpose

Let us note \( V_0, V_1, \ldots, V_{m−1} \) the variables of the `VARIABLES` collection. `PERIOD` is the period of the sequence \( V_0, V_1, \ldots, V_{m−1} \) according to constraint `CTR`. This means that `PERIOD` is the smallest natural number such that \( V_i \ CTR V_{i+PERIOD} \lor V_i = 0 \lor V_{i+PERIOD} = 0 \) holds for all \( i ∈ 0, 1, \ldots, m − PERIOD − 1 \).

### Example

\[
\begin{pmatrix}
  \text{var} - 1, \\
  \text{var} - 1, \\
  \text{var} - 4, \\
  \text{var} - 1, \\
  \text{var} - 1, \\
  \text{var} - 0, \\
  \text{var} - 1, \\
  \text{var} - 1
\end{pmatrix}, =
\]

The `period_except_0` constraint holds since, as depicted by Figure 5.503, its first argument `PERIOD = 3` is equal (i.e., since `CTR` is set to `=`) to the period of the sequence `1 1 4 1 1 0 1 1`; value 0 is assumed to be equal to any other value.

![Figure 5.503: A sequence that has a period of 3 when we assume that value 0 can be equal to any other value](image)

### Typical

- `PERIOD > 1`
- `PERIOD < |VARIABLES|`
- `|VARIABLES| > 2`
- `range(VARIABLES.var) > 1`
- `atleast(1, VARIABLES, 0)`
- `CTR ∈ [=]`
Symmetries

- Items of VARIABLES can be reversed.
- Items of VARIABLES can be shifted.
- All occurrences of two distinct values of VARIABLES.var that are both different from 0 can be swapped; all occurrences of a value of VARIABLES.var that is different from 0 can be renamed to any unused value that is also different from 0.

Arg. properties

- Functional dependency: PERIOD determined by VARIABLES and CTR.
- Contractible wrt. VARIABLES when CTR ∈ [=] and PERIOD = 1.
- Prefix-contractible wrt. VARIABLES.
- Suffix-contractible wrt. VARIABLES.

Usage

Useful for timetabling problems where a person should repeat some work pattern over an over except when he is unavailable for some reason. The value 0 represents the fact that he is unavailable, while the other values are used in the work pattern.

Algorithm

See [52].

See also

hard version: period.
implied by: period.

Keywords

characteristic of a constraint: joker value.
combinatorial object: periodic, sequence.
constraint arguments: pure functional dependency.
constraint type: predefined constraint, timetabling constraint, scheduling constraint.
modelling: functional dependency.
5.286  period_vectors

DESCRIPTION

Origin  Derived from period

Constraint  period_vectors(PERIOD, VECTORS, CTRS)

Types  
  VECTOR : collection(var=dvar)
  CTR : atom

Arguments  
  PERIOD : dvar
  VECTORS : collection(vec=VECTOR)
  CTRS : collection(ctr=CTR)

Restrictions  
  |VECTOR| ≥ 1
  required(VECTOR, var)
  CTR ∈ [=, ≠, <, ≥, >, ≤]
  PERIOD ≥ 1
  PERIOD ≤ |VECTORS|
  required(VECTORS, vec)
  same_size(VECTORS, vec)
  required(CTRS, ctr)
  |CTRS| = |VECTOR|

Purpose  
Let us note VECTOR₀, VECTOR₁,...,VECTORₙ₋₁ the vectors of the VECTORS collection, and d the number of components of each vector (all vectors have the same size). PERIOD is the period of the sequence of vectors VECTOR₀, VECTOR₁,...,VECTORₙ₋₁ according to constraints CTRS. This means that PERIOD is the smallest natural number such that ∀i ∈ [0, n − PERIOD − 1], ∀j ∈ [0, d − 1] : VECTORᵢ, vec[j] CTRS[j] VECTORᵢ₊₁, vec[j].

Example  

The period_vectors constraint holds since its first argument PERIOD = 3 is equal (i.e., since CTRS is set to ⟨=, =⟩) to the period of the sequence vec − ⟨1, 0⟩, vec − ⟨1, 5⟩, vec − ⟨4, 4⟩, vec − ⟨1, 0⟩, vec − ⟨1, 5⟩, vec − ⟨4, 4⟩, vec − ⟨1, 0⟩, vec − ⟨1, 5⟩.
Typical

| CTR ∈ [\] |
| VECTOR| > 1 |
| PERIOD > 1 |
| PERIOD < |VECTORS| |
| VECTORS| > 2 |

Symmetry

Items of VECTORS can be reversed.

Arg. properties

- Functional dependency: PERIOD determined by VECTORS and CTRS.
- Prefix-contractible wrt. VECTORS.
- Suffix-contractible wrt. VECTORS.

See also

specialisation: period (vector replaced by variable).

Keywords

characteristic of a constraint: vector.
combinatorial object: periodic, sequence.
constraint arguments: pure functional dependency.
constraint type: predefined constraint.
modelling: functional dependency.
5.287 permutation

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from <code>alldifferent_consecutive_values</code>.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td><code>permutation(VARIABLES)</code></td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td><code>VARIABLES : collection(var−dvar)</code></td>
<td></td>
</tr>
</tbody>
</table>
| Restrictions| `required(VARIABLES,var)`  
               `minval(VARIABLES,var) = 1`  
               `maxval(VARIABLES,var) = |VARIABLES|` |       |
| Purpose     | Enforce all variables of the collection VARIABLES to take distinct values between 1 and the total number of variables. |       |
| Example     | `(⟨3, 2, 1, 4⟩)` |       |
|             | The permutation constraint holds since all the values 3, 2, 1 and 4 are distinct, and since they all belong to interval [1, 4] where 4 is the total number of variables. |       |
| Typical     | `|VARIABLES| > 2` |       |
| Symmetries  | • Items of VARIABLES are permutable.  
               • Two distinct values of VARIABLES.var can be swapped. |       |
| Usage       | See Usage slof of `alldifferent`. |       |
| Algorithm   | See Algorithm slof of `alldifferent`. |       |
| See also    | `implies: alldifferent_consecutive_values.` |       |
| Keywords    | characteristic of a constraint: all different, disequality, sort based reformulation.  
               combinatorial object: permutation.  
               constraint type: value constraint.  
               final graph structure: one_succ. |       |
Arc input(s) | VARIABLES
---|---
Arc generator | $\text{CLIQUE} \rightarrow \text{collection}(\text{variables1}, \text{variables2})$
Arc arity | 2
Arc constraint(s) | $\text{variables1}.\text{var} = \text{variables2}.\text{var}$
Graph property(ies) | $\text{MAX}_{\text{NSCC}} \leq 1$
Graph class | $\text{ONE}_{\text{SUCC}}$

**Graph model**

We generate a *clique* with an *equality* constraint between each pair of vertices (including a vertex and itself) and state that the *size* of the largest strongly connected component should not exceed one. Finally the restrictions express the fact that all values are between 1 and the total number of variables.

Parts (A) and (B) of Figure 5.504 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_{NSCC}** graph property we show one of the largest strongly connected component of the final graph. The permutation holds since all the strongly connected components have at most one vertex: a value is used at most once.

![Graph model diagram](image)

Figure 5.504: Initial and final graph of the permutation constraint
5.288

place_in_pyramid

DESCRIPTION

Origin
N. Beldiceanu

Constraint
place_in_pyramid(ORTHOTOPES, VERTICAL_DIM)

Type
ORTHOTOPE : collection(ori−dvar, siz−dvar, end−dvar)

Arguments
ORTHOTOPES : collection(orth − ORTHOTOPE)
VERTICAL_DIM : int

Restrictions

|ORTHOTOPE| > 0
require_at_least(2, ORTHOTOPE, [ori, siz, end])
ORTHOTOPE.siz ≥ 0
ORTHOTOPE ori ≤ ORTHOTOPE.end
required(ORTHOTOPE. orth)
same_size(ORTHOTOPES, orth)
VERTICAL_DIM ≥ 1
diffn(ORTHOTOPES)

Purpose
For each pair of orthotopes \((O_1, O_2)\) of the collection ORTHOTOPES, \(O_1\) and \(O_2\) do not overlap (two orthotopes do not overlap if there exists at least one dimension where their projections do not overlap). In addition, each orthotope of the collection ORTHOTOPES should be supported by one other orthotope or by the ground. The vertical dimension is given by the parameter VERTICAL_DIM.

Example

\[
\begin{pmatrix}
\langle \text{orth} \rangle & \langle \text{ori}−1, \text{siz}−3, \text{end}−4 \rangle, \\
\langle \text{orth} \rangle & \langle \text{ori}−1, \text{siz}−2, \text{end}−3 \rangle, \\
\langle \text{orth} \rangle & \langle \text{ori}−3, \text{siz}−3, \text{end}−6 \rangle, \\
\langle \text{orth} \rangle & \langle \text{ori}−5, \text{siz}−6, \text{end}−11 \rangle, \\
\langle \text{orth} \rangle & \langle \text{ori}−1, \text{siz}−2, \text{end}−3 \rangle, \\
\langle \text{orth} \rangle & \langle \text{ori}−5, \text{siz}−2, \text{end}−7 \rangle, \\
\langle \text{orth} \rangle & \langle \text{ori}−3, \text{siz}−2, \text{end}−5 \rangle, \\
\langle \text{orth} \rangle & \langle \text{ori}−8, \text{siz}−3, \text{end}−11 \rangle, \\
\langle \text{orth} \rangle & \langle \text{ori}−3, \text{siz}−2, \text{end}−5 \rangle, \\
\langle \text{orth} \rangle & \langle \text{ori}−8, \text{siz}−2, \text{end}−10 \rangle, \\
\langle \text{orth} \rangle & \langle \text{ori}−5, \text{siz}−2, \text{end}−7 \rangle \\
\end{pmatrix}, 2
\]

Figure 5.505 depicts the placement associated with the example, where the \(i^{th}\) item of the ORTHOTOPES collection is represented by the rectangle \(R_i\). The place_in_pyramid constraint holds since the rectangles do not overlap and since rectangles \(R_1, R_2, R_3, R_4, R_5\), and \(R_6\) are respectively supported by the ground, \(R_1\), the ground, \(R_3\), \(R_3\), and \(R_5\).
Typical

|ORTHOTOPE| > 1
|ORTHOTOPE.siz| > 0
|ORTHOTOPES| > 1

Symmetry

Items of ORTHOTOPES are permutable.

Usage

The diffn constraint is not enough if one wants to produce a placement where no orthotope floats in the air. This constraint is usually handled with a heuristic during the enumeration phase.

See also

used in graph description: orth_on_the_ground, orth_on_top_of_orth.

Keywords

constraint type: logic.
geometry: geometrical constraint, non-overlapping, orthotope.

Figure 5.505: Solution corresponding to the example
Arc input(s)  ORTHOTOPES
Arc generator  $\text{CLIQUE} \rightarrow \text{collection}(\text{orthotopes1, orthotopes2})$
Arc arity  2
Arc constraint(s)  
\[
\bigwedge \left( \begin{array}{c}
\text{orthotopes1.key} = \text{orthotopes2.key}, \\
\text{orth_on_the_ground}(\text{orthotopes1.orth, VERTICAL_DIM})
\end{array} \right),
\bigvee \left( \begin{array}{c}
\text{orthotopes1.key} \neq \text{orthotopes2.key}, \\
\text{orth_on_top_of_orth}
\end{array} \right)
\]
Graph property(ies)  NARC = |ORTHOTOPES|

Graph model

The arc constraint of the graph constraint enforces one of the following conditions:

- If the arc connects the same orthotope $O$ then the ground directly supports $O$,
- Otherwise, if we have an arc from an orthotope $O_1$ to a distinct orthotope $O_2$, the condition is: $O_1$ is on top of $O_2$ (i.e., in all dimensions, except dimension VERTICAL_DIM, the projection of $O_1$ is included in the projection of $O_2$, while in dimension VERTICAL_DIM the projection of $O_1$ is located after the projection of $O_2$).

Parts (A) and (B) of Figure 5.506 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

![Graph](image)

Figure 5.506: Initial and final graph of the place_in_pyramid constraint
### 5.289 polyomino

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Inspired by ([180]).</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>polyomino(CELLS)</td>
<td></td>
</tr>
<tr>
<td><strong>Argument</strong></td>
<td>CELLS : (\text{collection}) ((\text{index} - \text{int}, \text{right} - \text{dvar}, \text{left} - \text{dvar}, \text{up} - \text{dvar}, \text{down} - \text{dvar}))</td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>CELLS.index (\geq 1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CELLS.index (\leq</td>
<td>\text{CELLS}</td>
</tr>
<tr>
<td></td>
<td>(</td>
<td>\text{CELLS}</td>
</tr>
<tr>
<td></td>
<td>(\text{required}(\text{CELLS}, \text{index, right, left, up, down}))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{distinct}(\text{CELLS}, \text{index}))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CELLS.right (\geq 0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CELLS.right (\leq</td>
<td>\text{CELLS}</td>
</tr>
<tr>
<td></td>
<td>CELLS.left (\geq 0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CELLS.left (\leq</td>
<td>\text{CELLS}</td>
</tr>
<tr>
<td></td>
<td>CELLS.up (\geq 0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CELLS.up (\leq</td>
<td>\text{CELLS}</td>
</tr>
<tr>
<td></td>
<td>CELLS.down (\geq 0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CELLS.down (\leq</td>
<td>\text{CELLS}</td>
</tr>
</tbody>
</table>

Enforce all cells of the collection CELLS to be connected and to form one single block. Each cell is defined by the following attributes:

1. The **index** attribute of the cell, which is an integer between 1 and the total number of cells, is unique for each cell.
2. The **right** attribute that is the index of the cell located immediately to the right of that cell (or 0 if no such cell exists).
3. The **left** attribute that is the index of the cell located immediately to the left of that cell (or 0 if no such cell exists).
4. The **up** attribute that is the index of the cell located immediately on top of that cell (or 0 if no such cell exists).
5. The **down** attribute that is the index of the cell located immediately above that cell (or 0 if no such cell exists).

This corresponds to a polyomino \([181]\).
The polyomino constraint holds since all the cells corresponding to the items of the CELLS collection form one single group of connected cells: the $i^{th}$ ($i \in [1, 4]$) cell is connected to the $(i + 1)^{th}$ cell. Figure 5.507 shows the corresponding polyomino.

![Polyomino example](image)

Figure 5.507: Polyomino corresponding to the example

### Symmetries
- Items of CELLS are permutable.
- Attributes of CELLS are permutable w.r.t. permutation (index) (right, left) (up, down) (permutation applied to all items).
- Attributes of CELLS are permutable w.r.t. permutation (index) (right) (left) (up, down) (permutation applied to all items).
- Attributes of CELLS are permutable w.r.t. permutation (index) (up, left, down, right) (permutation applied to all items).

### Usage
- Enumeration of polyominoes.

### Keywords
- **combinatorial object**: pentomino.
- **final graph structure**: strongly connected component.
- **geometry**: geometrical constraint.
- **puzzles**: pentomino.
**Arc input(s)**

CELLS

**Arc generator**

\[\text{CLIQUE}(\neq) \mapsto \text{collection}(\text{cells1}, \text{cells2})\]

**Arc arity**

2

**Arc constraint(s)**

\[
\begin{align*}
\text{cells1.right} &= \text{cells2.index} \land \text{cells2.left} = \text{cells1.index}, \\
\text{cells1.left} &= \text{cells2.index} \land \text{cells2.right} = \text{cells1.index}, \\
\text{cells1.up} &= \text{cells2.index} \land \text{cells2.down} = \text{cells1.index}, \\
\text{cells1.down} &= \text{cells2.index} \land \text{cells2.up} = \text{cells1.index}
\end{align*}
\]

**Graph property(ies)**

- \(\text{NVERTEX} = |\text{CELLS}|\)
- \(\text{NCC} = 1\)

**Graph model**

The graph constraint models the fact that all the cells are connected. We use the \(\text{CLIQUE}(\neq)\) arc generator in order to only consider connections between two distinct cells. The first graph property \(\text{NVERTEX} = |\text{CELLS}|\) avoid the case isolated cells, while the second graph property \(\text{NCC} = 1\) enforces to have one single group of connected cells.

Parts (A) and (B) of Figure 5.508 respectively show the initial and final graph associated with the Example slot. Since we use the \(\text{NVERTEX}\) graph property the vertices of the final graph are stressed in bold. Since we also use the \(\text{NCC}\) graph property we show the unique connected component of the final graph. An arc between two vertices indicates that two cells are directly connected.

![Graph model](image)

**Signature**

From the graph property \(\text{NVERTEX} = |\text{CELLS}|\) and from the restriction \(|\text{CELLS}| \geq 1\)
we have that the final graph is not empty. Therefore it contains at least one connected component. So we can rewrite $\text{NCC} = 1$ to $\text{NCC} \leq 1$ and simplify $\text{NCC}$ to $\text{NCC}$. 
5.290  power

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[126]</td>
</tr>
<tr>
<td>Constraint</td>
<td>power((X, N, Y))</td>
</tr>
<tr>
<td>Synonym</td>
<td>xexpyeqz.</td>
</tr>
<tr>
<td>Arguments</td>
<td>(X : dvar) (N : dvar) (Y : dvar)</td>
</tr>
<tr>
<td>Restrictions</td>
<td>(X \geq 0) (N \geq 0) (Y \geq 0)</td>
</tr>
<tr>
<td>Purpose</td>
<td>Enforce the fact that (Y) is equal to (X^N).</td>
</tr>
<tr>
<td>Example</td>
<td>((2, 3, 8))</td>
</tr>
<tr>
<td>Typical</td>
<td>(X &gt; 1) (N &gt; 1) (Y &gt; 1)</td>
</tr>
<tr>
<td>Arg. properties</td>
<td>Functional dependency: (Y) determined by (X) and (N).</td>
</tr>
<tr>
<td>Algorithm</td>
<td>In [126] a filtering algorithm for the power constraint was automatically derived from the algorithm that multiplies (X) by itself (N) times by using constructive disjunction and abstract interpretation in order to approximate the behaviour of the while loop of that algorithm.</td>
</tr>
<tr>
<td>Systems</td>
<td>xexpyeqz in JaCoP.</td>
</tr>
<tr>
<td>See also</td>
<td>common keyword: gcd (abstract interpretation).</td>
</tr>
</tbody>
</table>
### 5.291 precedence

<table>
<thead>
<tr>
<th><strong>DESCRIPTION</strong></th>
<th><strong>LINKS</strong></th>
<th><strong>GRAPH</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Scheduling</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>precedence(TASKS)</td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>TASKS : collection(origin−dvar,duration−dvar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(TASKS,[origin,duration]) TASKS.duration ≥ 0</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>All consecutive pairs of tasks of the collection TASKS should be ordered (i.e., the end of the first task of a pair should be less than or equal to the start of the second task of the same pair).</td>
<td></td>
</tr>
</tbody>
</table>
| Example        | \[
|               | (origin − 1 duration − 3,
|               | (origin − 4 duration − 0,
|               | (origin − 5 duration − 2,
|               | (origin − 8 duration − 1))\] |           |
|               | Since the tasks are ordered (i.e., \(1 + 3 \leq 4, 4 + 0 \leq 5, 5 + 2 \leq 8\)) the precedence constraint holds. |           |
| Typical        | |TASKS| > 1
|               | TASKS.duration ≥ 1 |           |
| Symmetries     | • TASKS.duration can be decreased to any value ≥ 0. |           |
|               | • One and the same constant can be added to the origin attribute of all items of TASKS. |           |
| Arg. properties| Contractible wrt. TASKS. |           |
| See also       | common keyword: increasing (order constraint). implies: disjunctive. |           |
| Keywords       | constraint type: decomposition, order constraint. filtering: arc-consistency. |           |
Arc input(s) | TASKS
---|---
Arc generator | $PATH \mapsto \text{collection}(\text{tasks1, tasks2})$
Arc arity | 2
Arc constraint(s) | tasks1.origin + tasks1.duration $\leq$ tasks2.origin
Graph property(ies) | $\text{NARC} = |\text{TASKS}| - 1$

Graph model
Since we are only interested by the constraints linking two consecutive items of the collection TASKS we use $PATH$ to generate the arcs of the initial graph.

Parts (A) and (B) of Figure 5.509 respectively show the initial and final graph of the first example of the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

![Figure 5.509: Initial and final graph of the precedence constraint](image)
5.292 product\_ctr

**Origin**
Arithmetic constraint.

**Constraint**
product\_ctr(VARIABLES, CTR, VAR)

**Arguments**
- VARIABLES: collection(var–dvar)
- CTR: atom
- VAR: dvar

**Restrictions**
required(VARIABLES, var)
CTR ∈ [=, ≠, <, ≥, >, ≤]

**Purpose**
Constraint the product of a set of domain variables. More precisely, let \( P \) denote the product of the variables of the VARIABLES collection. Enforce the following constraint to hold: \( P \text{ CTR VAR} \).

**Example**
\[(2, 1, 4), =, 8\]

The product\_ctr constraint holds since its last argument VAR = 8 is equal (i.e., CTR is set to =) to \( 2 \cdot 1 \cdot 4 \).

**Typical**
- \(|\text{VARIABLES}| > 1\)
- \(|\text{VARIABLES}| < 10\)
- range(\text{VARIABLES.var}) > 1
- VARIABLES.var ≠ 0
- CTR ∈ [=, <, ≥, >, ≤]
- VAR ≠ 0

**Symmetry**
Items of VARIABLES are permutable.

**Arg. properties**
- **Contractible** wrt. VARIABLES when CTR ∈ \([<, ≤]\) and minval(VARIABLES.var) > 0.
- **Aggregate**: VARIABLES(union), CTR(id), VAR(*) when CTR ∈ [=].

**Used in**
cumulative\_product.

**See also**
common keyword: range\_ctr, sum\_ctr (arithmetic constraint).

**Keywords**
- characteristic of a constraint: product.
- constraint type: arithmetic constraint.
Arc input(s) | VARIABLES
---|---
Arc generator | SELF→collection(variables)
Arc arity | 1
Arc constraint(s) | TRUE
Graph property(ies) | PROD(VARIABLES, var) CTR VAR

Graph model

Since we want to keep all the vertices of the initial graph we use the SELF arc generator together with the TRUE arc constraint. This predefined arc constraint always holds.

Parts (A) and (B) of Figure 5.510 respectively show the initial and final graph associated with the Example slot. Since we use the TRUE arc constraint both graphs are identical.

![Graphs](image-url)

Figure 5.510: Initial and final graph of the product_ctr constraint
5.293 proper_forest

**DESCRIPTION**

**LINKS**

**GRAPH**

**Origin**

Derived from tree,[44].

**Constraint**

proper_forest(NTREES, NODES)

**Arguments**

<table>
<thead>
<tr>
<th>NTREES</th>
<th>dvar</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODES</td>
<td>collection(index−int, neighbour−svar)</td>
</tr>
</tbody>
</table>

**Restrictions**

- NTREES $\geq 0$
- required(NODES.[index, neighbour])
- $|NODES| \text{ mod } 2 = 0$
- NODES.index $\geq 1$
- NODES.index $\leq |NODES|$
- distinct(NODES.index)
- NODES.neighbour $\geq 1$
- NODES.neighbour $\leq |NODES|$
- NODES.neighbour $\neq$ NODES.index

**Purpose**

Cover an undirected graph $G$ by a set of NTREES trees (i.e., a tree is a connected graph without cycles that contains at least two vertices [100]) in such a way that each vertex of $G$ belongs to one distinct tree.

**Example**

The proper_forest constraint holds since the undirected graph associated with the items of the NODES collection corresponds to a forest containing NTREES = 3 trees: each tree respectively involves the vertices \{1, 3, 5, 6, 7\}, \{2, 4, 9\} and \{8, 10\}.

**Typical**

- NTREES $> 0$
- $|NODES| > 1$

**Symmetry**

Items of NODES are permutable.

**Arg. properties**

Functional dependency: NTREES determined by NODES.
Algorithm

A filtering algorithm for the proper_forest constraint was proposed by N. Beldiceanu et al. in [44]. It achieves hybrid-consistency and its running time is dominated by the complexity of finding all edges that do not belong to any maximum cardinality matching in an undirected \( n \)-vertex, \( m \)-edge graph, i.e., \( O(m \cdot n) \).

Systems

tree in Choco.

See also

common keyword: tree (connected component, tree).

Keywords

characteristic of a constraint: undirected graph.

constraint arguments: constraint involving set variables.

constraint type: graph constraint.

filtering: hybrid-consistency.

final graph structure: connected component, tree, no cycle, symmetric.

modelling: functional dependency.
Arc input(s) | NODES
--- | ---
Arc generator | $\text{CLIQUE}(\neq) \rightarrow \text{collection}(\text{nodes1}, \text{nodes2})$
Arc arity | 2
Arc constraint(s) | $\text{in}_\text{set}(\text{nodes2.index}, \text{nodes1.neighbour})$
Graph property(ies) | • $\text{NVERTEX} = (\text{NARC} + 2 \times \text{NTREES}) / 2$
| • $\text{NCC} = \text{NTREES}$
| • $\text{NVERTEX} = |\text{NODES}|$
Graph class | SYMMETRIC

Graph model

The graph constraint enforces the following conditions:

- Each connected component of the final graph has $n$ vertices and $2 \cdot (n - 1)$ arcs. This is equivalent to the fact that each connected component has not any cycle.
- Since we use the $\text{CLIQUE}(\neq)$ arc-generator and since, by definition, the final graph does not contain any isolated vertex, each connected component of the final graph involves more than one vertex.
- The number of connected components of the final graph is equal to $\text{NCC}$.
- All the vertices of the initial graph belong to the final graph.
- The final graph is symmetric.

Parts (A) and (B) of Figure 5.511 respectively show the initial and final graph associated with the Example slot. For each connected component we display its number of arcs as well as its number of vertices. The proper forest constraint holds since the final graph has $\text{NTREES} = \text{NCC} = 3$ connected components and no cycle.
Figure 5.511: Initial and final graph of the proper forest constraint
5.294  range_ctr

**Description**

Arithmetic constraint.

**Constraint**

`range_ctr(VARIABLES, CTR, R)`

**Arguments**

- `VARIABLES` : `collection(var−dvar)`
- `CTR` : `atom`
- `R` : `dvar`

**Restrictions**

- `|VARIABLES| > 0`
- `required(VARIABLES, var)`
- `CTR ∈ [=, ≠, <, ≥, >, ≤]`

**Purpose**

Constraint the difference between the maximum value and the minimum value of a set of domain variables. More precisely, let `RANGE` denote the difference between the largest and the smallest variables of the `VARIABLES` collection plus one. Enforce the following constraint to hold: `RANGE CTR R`.

**Example**

`(⟨1, 9, 4⟩, =, 9)`

The `range_ctr` constraint holds since `max(1, 9, 4) − min(1, 9, 4) + 1` is equal (i.e., `CTR` is set to `=`) to its last argument `R = 9`.

**Typical**

- `|VARIABLES| > 1`
- `range(VARIABLES, var) > 1`
- `CTR ∈ [=, <, ≥, >, ≤]`

**Symmetries**

- Items of `VARIABLES` are permutable.
- All occurrences of two distinct values of `VARIABLES.var` can be swapped.
- One and the same constant can be added to the `var` attribute of all items of `VARIABLES`.

**Arg. properties**

- **Contractible** wrt. `VARIABLES` when `CTR ∈ [≤, ≤]`.
- **Extensible** wrt. `VARIABLES` when `CTR ∈ [≥, >]`.

**Used in**

`shift`.

**See also**

- **common keyword**: `product_ctr, sum_ctr` (*arithmetic constraint*).

**Keywords**

- **characteristic of a constraint**: `range`.
- **constraint type**: `arithmetic constraint`. 
Figure 5.512: Illustration of the example: three variables respectively fixed to values 1, 9 and 4, and their corresponding range $R = 9$. Values: minimum value = 1, maximum value = 9. Variables: $R = 9 - 1 + 1$. The diagram shows the range $R$ for the variables.
Arc input(s)  VARIABLES
Arc generator  $SELF \rightarrow \text{collection}(\text{variables})$
Arc arity  1
Arc constraint(s)  TRUE
Graph property(ies)  $\text{RANGE}(\text{VARIABLES}, \text{var}) \text{ CTR R}$

Graph model
Since we want to keep all the vertices of the initial graph we use the $SELF$ arc generator together with the TRUE arc constraint. This predefined arc constraint always holds.

Parts (A) and (B) of Figure 5.513 respectively show the initial and final graph associated with the Example slot. Since we use the TRUE arc constraint both graphs are identical.

Figure 5.513: Initial and final graph of the range_ctr constraint
### 5.295 relaxed_sliding_sum

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>CHIP</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>relaxed_sliding_sum(ATLEAST, ATMOST, LOW, UP, SEQ, VARIABLES)</td>
<td></td>
</tr>
</tbody>
</table>

#### Arguments
- ATLEAST : int
- ATMOST : int
- LOW : int
- UP : int
- SEQ : int
- VARIABLES : collection(var-dvar)

#### Restrictions
- ATLEAST ≥ 0
- ATMOST ≥ ATLEAST
- ATMOST ≤ |VARIABLES| − SEQ + 1
- UP ≥ LOW
- SEQ > 0
- SEQ ≤ |VARIABLES| − required(VARIABLES, var)

#### Purpose
There are between ATLEAST and ATMOST sequences of SEQ consecutive variables of the collection VARIABLES such that the sum of the variables of the sequence is in [LOW, UP].

#### Example

\[
\begin{pmatrix}
\text{var} - 2, \\
\text{var} - 4, \\
3, 4, 3, 7, 4, \\
\text{var} - 2, \\
\text{var} - 0, \\
\text{var} - 4, \\
\text{var} - 3, \\
\text{var} - 4
\end{pmatrix}
\]

Within the sequence 2 4 2 0 0 3 4 we have exactly 3 subsequences of SEQ = 4 consecutive values such that their sum is located within the interval [LOW, UP] = [3, 7]: subsequences 4 2 0 0, 2 0 0 3 and 0 0 3 4. Consequently the relaxed_sliding_sum constraint holds since the number of such subsequences is located within the interval [ATLEAST, ATMOST] = [3, 4].

#### Typical
- SEQ > 1
- SEQ < |VARIABLES|
- range(VARIABLES.var) > 1
- ATLEAST > 0 ∨ ATMOST < |VARIABLES| − SEQ + 1

#### Symmetries
- ATLEAST can be decreased to any value ≥ 0.
- ATMOST can be increased to any value ≤ |VARIABLES| − SEQ + 1.
- Items of VARIABLES can be reversed.
Algorithm [29].

See also hard version: sliding_sum.
used in graph description: sum_ctr (the sliding constraint).

Keywords characteristic of a constraint: hypergraph.
combinatorial object: sequence.
constraint type: sliding sequence constraint, soft constraint, relaxation.
Arc input(s) | VARIABLES
---|---
Arc generator | \( PATH \rightarrow \text{collection} \)
Arc arity | SEQ
Arc constraint(s) | • \( \text{sum}_\text{ctr}(\text{collection}, \geq, \text{LOW}) \)
 | • \( \text{sum}_\text{ctr}(\text{collection}, \leq, \text{UP}) \)
Graph property(ies) | • \( \text{NARC} \geq \text{ATLEAST} \)
 | • \( \text{NARC} \leq \text{ATMOST} \)

Graph model

Within the context of the Example slot, the corresponding final directed hypergraph is given by Figure 5.514. For each vertex of the graph we show its corresponding position within the collection of variables. The constraint associated with each arc corresponds to a conjunction of two \( \text{sum}_\text{ctr} \) constraints involving 4 consecutive variables. We did not put vertex 1 since the single arc constraint that mentions vertex 1 does not hold (i.e., the sum \( 2 + 4 + 2 + 0 = 8 \) is not located in interval \([3, 7]\)). However, the directed hypergraph contains 3 arcs, so the relaxed sliding \( \text{sum} \) constraint is satisfied since it was requested to have between 3 and 4 arcs.

![Figure 5.514: Final directed hypergraph associated with the example](image-url)
## 5.296 remainder

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
</table>

### Origin
Arithmetic.

### Constraint
\( \text{remainder}(Q, D, R) \)

### Synonyms
modulo, mod.

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Q )</td>
<td>dvar</td>
</tr>
<tr>
<td>( D )</td>
<td>dvar</td>
</tr>
<tr>
<td>( R )</td>
<td>dvar</td>
</tr>
</tbody>
</table>

### Restrictions
\[
\begin{align*}
Q & \geq 0 \\
D & > 0 \\
R & \geq 0 \\
R & < D
\end{align*}
\]

### Purpose
Enforce \( R \) to be equal to the remainder of the division of \( Q \) by \( D \).

### Example
\[ (15, 2, 1) \]

The remainder constraint holds since 1 is the rest of the division of 15 by 2.

### Arg. properties

<table>
<thead>
<tr>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional dependency: ( R ) determined by ( Q ) and ( D ).</td>
</tr>
</tbody>
</table>

### Keywords

<table>
<thead>
<tr>
<th>Constraint arguments:</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary constraint, pure functional dependency.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constraint type:</th>
</tr>
</thead>
<tbody>
<tr>
<td>predefined constraint, arithmetic constraint.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Modelling:</th>
</tr>
</thead>
<tbody>
<tr>
<td>functional dependency.</td>
</tr>
</tbody>
</table>
5.297 roots

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[59]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>roots(S, T, VARIABLES)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>S : svar</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T : svar</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>S ≤</td>
<td>VARIABLES</td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES, var)</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>S is the set of indices of the variables in the collection VARIABLES taking their values in T; S = {i</td>
<td>VARIABLES[i].var ∈ T}. Positions are numbered from 1.</td>
</tr>
</tbody>
</table>
| Example     | \(\begin{pmatrix}
{2, 4, 5},
{2, 3, 8},
{1, 3, 1, 2, 3}
\end{pmatrix}\) |       |
|             | The roots constraint holds since values 2 and 3 in T occur in the collection \(\{1, 3, 1, 2, 3\}\) only at positions S = \{2, 4, 5\}. The value 8 ∈ T does not occur within the collection \(\{1, 3, 1, 2, 3\}\). |       |
| Typical     | \(|\text{VARIABLES}| > 1 |       |
|             | range(\text{VARIABLES}.var) > 1 |       |
| Usage       | Bessière et al. showed [59] that many counting and occurrence constraints can be specified with two global primitives: roots and range. For instance, the count constraint can be decomposed into one roots constraint: \(\text{count}(\text{VAL}, \text{VARS}, \text{OP}, \text{NVAR})\) \iff roots(S, \text{VAL}, \text{VARS}) \land |S| |
|             | \(|\text{VARIABLES}| \subset T, \forall i \in S \Rightarrow variables[i].var \subset T and variables[i].var \Rightarrow T \land i \in S. Enforcing bound consistency on the decomposition achieves bound consistency on roots. Enforcing hybrid consistency on the decomposition achieves at least bound consistency on roots, until hybrid consistency in some special cases: |       |
|             | \(\text{dom}(\text{VARIABLES}[i].var) \subset T, \forall i \in S\). |       |
| Algorithm   | In [62], Bessière et al. shows that enforcing hybrid-consistency on roots is NP-hard. They consider the decomposition of roots into a network of ternary constraints: \(\forall i, i \in S \Rightarrow variables[i].var \in T and variables[i].var \Rightarrow T \land i \in S. Enforcing bound consistency on the decomposition achieves bound consistency on roots. Enforcing hybrid consistency on the decomposition achieves at least bound consistency on roots, until hybrid consistency in some special cases: |       |
• $\text{dom}(\text{VARIABLES}[i].\text{var}) \cap T = \emptyset$, $\forall i \notin S$.
• VARIABLES are ground.
• $T$ is ground.

Enforcing hybrid consistency on the decomposition can be done in $O(nd)$ with $n = |\text{VARIABLES}|$ and $d$ the maximum domain size of $\text{VARIABLES}[i].\text{var}$ and $T$.

**Systems**
roots in Gecode, roots in MiniZinc.

**See also**
common keyword: link_set_to_booleans (constraint involving set variables).
related: among (can be expressed with roots), assign_and_nvalues (can be expressed with roots and range), atleast, atmost (can be expressed with roots), common (can be expressed with roots and range), count (can be expressed with roots), domain_constraint, global_cardinality, global_contiguity (can be expressed with roots), symmetric_alldifferent, uses (can be expressed with roots and range).

**Keywords**
characteristic of a constraint: disequality.
constraint arguments: constraint involving set variables.
constraint type: counting constraint, value constraint, decomposition.
filtering: hybrid-consistency.
Derived Collection

\[ \text{col}(\text{SETS}\times\text{collection}(\text{s\_var},\text{t\_var}),|\text{item}(\text{s},\text{S},\text{t},\text{T})|) \]

Arc input(s)

\text{SETS VARIABLES}

Arc generator

\[ \text{PRODUCT} \rightarrow \text{collection} \]

Arc arity

2

Arc constraint(s)

\[ \text{in\_set}(\text{variables.key},\text{sets.s}) \leftrightarrow \text{in\_set}(\text{variables.var},\text{sets.t}) \]

Graph property(ies)

\[ \text{NARC} = |\text{VARIABLES}| \]

Figure 5.515: Initial and final graph of the \textit{roots} constraint

Graph model
### 5.298 same

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>N. Beldiceanu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>\texttt{same(VARIABLES1, VARIABLES2)}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>\texttt{VARIABLES1} : \texttt{collection(var–dvar)} [\texttt{VARIABLES2} : \texttt{collection(var–dvar)}]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>(</td>
<td>\texttt{VARIABLES1}</td>
<td>=</td>
</tr>
<tr>
<td>Purpose</td>
<td>The variables of the \texttt{VARIABLES2} collection correspond to the variables of the \texttt{VARIABLES1} collection according to a permutation.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{pmatrix}
\text{var} - 1, \\
\text{var} - 9, \\
\text{var} - 1, \\
\text{var} - 5, \\
\text{var} - 2, \\
\text{var} - 1, \\
\text{var} - 9, \\
\text{var} - 1, \\
\text{var} - 1, \\
\text{var} - 1, \\
\text{var} - 2, \\
\text{var} - 5
\end{pmatrix}
\]

The \texttt{same} constraint holds since values 1, 2, 5 and 9 have the same number of occurrences within both collections \langle 1, 9, 1, 5, 2, 1 \rangle and \langle 9, 1, 1, 1, 2, 5 \rangle. Figure 5.516 illustrates this correspondence.

![Graph](image)

**Figure 5.516**: Correspondence between collection \langle 1, 9, 1, 5, 2, 1 \rangle and collection \langle 9, 1, 1, 1, 2, 5 \rangle

**Typical**

\[|\texttt{VARIABLES1}| > 1\] \[\texttt{range(VARIABLES1.var)} > 1\] \[\texttt{range(VARIABLES2.var)} > 1\]
Symmetries

- Arguments are permutable w.r.t. permutation \((\text{VARIABLES1}, \text{VARIABLES2})\).
- Items of \text{VARIABLES1} are permutable.
- Items of \text{VARIABLES2} are permutable.
- All occurrences of two distinct values in \text{VARIABLES1}.var or \text{VARIABLES2}.var can be swapped; all occurrences of a value in \text{VARIABLES1}.var or \text{VARIABLES2}.var can be renamed to any unused value.

Arg. properties

\text{Aggregate:} \text{VARIABLES1(union)}, \text{VARIABLES2(union)}.

Usage

The same constraint can be used in the following contexts:

- Pairing problems taken from [46]. The organisation Doctors Without Borders has a list of doctors and a list of nurses, each of whom volunteered to go on one mission in the next year. Each volunteer specifies a list of possible dates and each mission involves one doctor and one nurse. The task is to produce a list of pairs such that each pair includes a doctor and a nurse who are available at the same date and each volunteer appears in exactly one pair. The problem is modelled by a same\((D = d_1, d_2, \ldots, d_m, N = n_1, n_2, \ldots, n_m)\) constraint where each doctor is represented by a domain variable in \(D\) and each nurse by a domain variable in \(N\). For a given doctor or nurse the corresponding domain variable gives the dates when the person is available. When the number of nurses is different from the number of doctors we replace the same constraint by a used_by constraint.

- Timetabling problems where we wish to produce fair schedules for different persons is a second use of the same constraint. Assume we need to generate a plan over a period of \(D\) consecutive days for \(P\) persons. For each day \(d\) and each person \(p\) we need to decide whether person \(p\) works in the morning shift, in the afternoon shift, in the night shift or does not work at all on day \(d\). In a fair schedule, the number of morning shifts should be the same for all the persons. The same condition holds for the afternoon and the night shifts as well as for the days off. We create for each person \(p\) the sequence of variables \(v_{p,1}, v_{p,2}, \ldots, v_{p,D}\) is equal to one of 0, 1, 2 and 3, depending on whether person \(p\) does not work, works in the morning, in the afternoon or during the night on day \(d\). We can use \(P − 1\) same constraints to express the fact that \(v_{1,1}, v_{1,2}, \ldots, v_{1,D}\) should be a permutation of \(v_{p,1}, v_{p,2}, \ldots, v_{p,D}\) for each \((1 < p \leq P)\).

- The same constraint can also be used as a channelling constraint for modelling the following recurring pattern: given the number of 1s in each line and each column of a 0-1 matrix \(M\) with \(n\) rows and \(m\) columns, reconstruct the matrix. This pattern usually occurs with additional constraints about compatible positions of the 1s, or about the overall shape reconstituted from all the 1’s (e.g., convexity, connectivity). If we restrict ourselves to the basic pattern there is an \(O(mn)\) algorithm for reconstructing a \(m \cdot n\) matrix from its horizontal and vertical directions [165]. We show how to model this pattern with the same constraint. Let \(l_i\) (\(1 \leq i \leq n\)) and \(c_j\) (\(1 \leq j \leq m\)) denote respectively, the required number of 1s in the \(i\)th row and the \(j\)th column of \(M\). We number the entries of the matrix as shown in the left-hand side of 5.517. For row \(i\) we create \(l_i\) domain variables \(v_{i,k}\) where \(k \in [1, l_i]\). Similarly, for each column \(j\) we create \(c_j\) domain variables \(u_{j,k}\) where \(k \in [1, c_j]\). The domain of each variable contains the set of entries that belong to the row or column that the variable corresponds to. Thus, each domain variable represents a 1 that appears in
the designated row or column. Let \( V \) be the set of variables corresponding to rows and \( U \) be the set of variables corresponding to columns. To make sure that each 1 is placed in a different entry, we impose the constraint \( \text{alldifferent}(U) \). In addition, the constraint \( \text{same}(U, V) \) enforces that the 1s exactly coincide on the rows and the columns. A solution is shown on the right-hand side of Figure 5.517. Note that the \text{same} and \text{global_cardinality} constraint allows to model the matrix reconstruction problem without the additional \text{alldifferent} constraint.

Remark

The \text{same} constraint is a relaxed version of the \text{sort} constraint introduced in [277]. We do not enforce the second collection of variables to be sorted in increasing order.

If we interpret the collections \text{VARIABLES1} and \text{VARIABLES2} as two multisets variables [222], the \text{same} constraint can be considered as an equality constraint between two multisets variables.

The \text{same} constraint can be modelled by two \text{global_cardinality} constraints. For instance, the \text{same} constraint

\[
\text{same} \left( \left\{ \text{var} \dashv x_1, \text{var} \dashv x_2 \right\}, \left\{ \text{var} \dashv y_1, \text{var} \dashv y_2 \right\} \right)
\]

where the union of the domains of the different variables is \( \{1, 2, 3, 4\} \) corresponds to the conjunction of the following two \text{global_cardinality} constraints:

\[
\text{global_cardinality} \left( \left\{ \text{var} \dashv x_1, \text{var} \dashv x_2 \right\}, \begin{cases} \text{val} \dashv 1 & \text{noccurrence} \dashv c_1 \\ \text{val} \dashv 2 & \text{noccurrence} \dashv c_2 \\ \text{val} \dashv 3 & \text{noccurrence} \dashv c_3 \\ \text{val} \dashv 4 & \text{noccurrence} \dashv c_4 \end{cases} \right)
\]

\[
\text{global_cardinality} \left( \left\{ \text{var} \dashv y_1, \text{var} \dashv y_2 \right\}, \begin{cases} \text{val} \dashv 1 & \text{noccurrence} \dashv c_1 \\ \text{val} \dashv 2 & \text{noccurrence} \dashv c_2 \\ \text{val} \dashv 3 & \text{noccurrence} \dashv c_3 \\ \text{val} \dashv 4 & \text{noccurrence} \dashv c_4 \end{cases} \right)
\]
As shown by the next example, the consistency for all variables of the two 
*global_cardinality* constraints does not imply consistency for the corresponding 
*same* constraint. This is for instance the case when the domains of $x_1$, $x_2$, $y_1$ and $y_2$ 
is respectively equal to $\{1, 2\}$, $\{3, 4\}$, $\{1, 2, 3, 4\}$ and $\{3, 4\}$. The conjunction of the two 
*global_cardinality* constraints does not remove values 3 and 4 from $y_1$.

In his PhD thesis, W.-J. van Hoeve introduces a soft version of the *same* constraint where 
the cost is the minimum number of variables to unassign in order to get back to a solution [398, page 78]. In the context of the *same* constraint this violation cost corresponds 
to the difference between the number of variables in VARIABLES1 and the number of 
values that both occur in VARIABLES1 and in VARIABLES2 (provided that one value of 
VARIABLES1 matches at most one value of VARIABLES2).

**Algorithm**

In [45, 46, 47, 213], it is shown how to model this constraint by a flow network that enables 
to compute arc-consistency and bound-consistency. Unlike the networks used for 
*alldifferent* and *global_cardinality*, the network now has three sets of nodes, so the algorithms are more complex, in particular the efficient bound-consistency algorithm.

**Reformulation**

The *same*(VARIABLES1, VARIABLES2) constraint can be reformulated as the conjunction 
$\text{sort}($VARIABLES1, SORTED_VARIABLES$) \land \text{sort}($VARIABLES2, SORTED_VARIABLES$)$.

**Used in**

$k$ same.

**See also**

- generalisation: correspondence (PERMUTATION parameter added), 
  *same_interval* (variable replaced by variable/constant), 
  *same_modulo* (variable replaced by variable $\mod$ constant), 
  *same_partition* (variable replaced by variable $\in$ partition).
- implied by: *lex_equal*, *same_and_global_cardinality*, 
  *same_and_global_cardinality_low_up*, *sort*.
- implies: *same_intersection*, used by.
- related to a common problem: *colored_matrix* (matrix reconstruction problem).
- soft variant: *soft_same_var* (variable-based violation measure).
- system of constraints: *k_same*.
- used in reformulation: *sort*.

**Keywords**

- characteristic of a constraint: sort based reformulation, automaton, 
  automaton with array of counters.
- combinatorial object: permutation, multiset.
- constraint arguments: constraint between two collections of variables.
- filtering: flow, arc-consistency, bound-consistency, DFS-bottleneck.
- modelling: channelling constraint, equality between multisets.
Arc input(s)  VARIABLES1 VARIABLES2
Arc generator  \textit{PRODUCT} \rightarrow \textit{collection}(\textit{variables1},\textit{variables2})
Arc arity  2
Arc constraint(s)  \textit{variables1}.\textit{var} = \textit{variables2}.\textit{var}
Graph property(ies)
\begin{itemize}
  \item for all connected components: \textit{NSOURCE} = \textit{NSINK}
  \item \textit{NSOURCE} = |\textit{VARIABLES1}|
  \item \textit{NSINK} = |\textit{VARIABLES2}|
\end{itemize}

Graph model
Parts (A) and (B) of Figure 5.518 respectively show the initial and final graph associated with the \textbf{Example} slot. Since we use the \textit{NSOURCE} and \textit{NSINK} graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. The same constraint holds since:
\begin{itemize}
  \item Each connected component of the final graph has the same number of sources and of sinks.
  \item The number of sources of the final graph is equal to |\textit{VARIABLES1}|.
  \item The number of sinks of the final graph is equal to |\textit{VARIABLES2}|.
\end{itemize}

Signature
Since the initial graph contains only sources and sinks, and since isolated vertices are eliminated from the final graph, we make the following observations:
\begin{itemize}
  \item Sources of the initial graph cannot become sinks of the final graph,
  \item Sinks of the initial graph cannot become sources of the final graph.
\end{itemize}

From the previous observations and since we use the \textit{PRODUCT} arc generator on the collections \textit{VARIABLES1} and \textit{VARIABLES2}, we have that the maximum number of sources and sinks of the final graph is respectively equal to |\textit{VARIABLES1}| and |\textit{VARIABLES2}|. Therefore we can rewrite \textit{NSOURCE} = |\textit{VARIABLES1}| to \textit{NSOURCE} \geq |\textit{VARIABLES1}| and simplify \textit{NSINK} to \textit{NSINK}. In a similar way, we can rewrite \textit{NSINK} = |\textit{VARIABLES2}| to \textit{NSINK} \geq |\textit{VARIABLES2}| and simplify \textit{NSINK} to \textit{NSINK}.
Figure 5.518: Initial and final graph of the same constraint
Automaton

To each item of the collection VARIABLES1 corresponds a signature variable $S_i$ that is equal to 0. To each item of the collection VARIABLES2 corresponds a signature variable $S_{i+|\text{VARIABLES}_1|}$ that is equal to 1.

Figure 5.519: Automaton of the same constraint
5.299 same_and_global_cardinality

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Conjoin same and global_cardinality</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>same_and_global_cardinality(VARIABLES1, VARIABLES2, VALUES)</td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>sgcc, same_gcc, same_and_gcc, swc, same_with_cardinalities.</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VARIABLES1 : collection(var−dvar)</td>
<td>VARIABLES2 : collection(var−dvar)</td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>The variables of the VARIABLES2 collection correspond to the variables of the VARIABLES1 collection according to a permutation. In addition, each value VALUES[i].val (with $i \in [1,</td>
<td>VALUES</td>
</tr>
</tbody>
</table>

Example

$$\langle \text{var}−1, \text{var}−9, \text{var}−1, \text{var}−5, \text{var}−2, \text{var}−1, \text{var}−9, \text{var}−1, \text{var}−1, \text{var}−2, \text{var}−5, \text{val}−1 \text{ noccurrence}−3, \text{val}−2 \text{ noccurrence}−1, \text{val}−5 \text{ noccurrence}−1, \text{val}−7 \text{ noccurrence}−0, \text{val}−9 \text{ noccurrence}−1 \rangle$$

The same_and_global_cardinality constraint holds since:
● The values 1, 9, 1, 5, 2, 1 assigned to VARIABLES1 correspond to a permutation of the values 9, 1, 1, 1, 2, 5 assigned to VARIABLES2.

● The values 1, 2, 5, 7 and 6 are respectively used 3, 1, 1, 0 and 1 times.

**Typical**

\[
\begin{align*}
|VARIABLES1| & > 1 \\
\text{range}(VARIABLES1.var) & > 1 \\
\text{range}(VARIABLES2.var) & > 1 \\
|VALUES| & > 1 \\
\text{range}(VALUES.noccurrence) & > 1 \\
|VARIABLES1| & > |VALUES|
\end{align*}
\]

**Symmetries**

● Arguments are permutable w.r.t. permutation (VARIABLES1, VARIABLES2) (VALUES).

● Items of VARIABLES1 are permutable.

● Items of VARIABLES2 are permutable.

● Items of VALUES are permutable.

● An occurrence of a value of VARIABLES1.var or VARIABLES2.var that does not belong to VALUES.val can be replaced by any other value that also does not belong to VALUES.val.

● All occurrences of two distinct values in VARIABLES1.var, VARIABLES2.var or VALUES.val can be swapped; all occurrences of a value in VARIABLES1.var, VARIABLES2.var or VALUES.val can be renamed to any unused value.

**Arg. properties**

Contractible wrt. VALUES.

**Usage**

See the same and global cardinality low up constraint.

**Algorithm**

The filtering algorithm presented in [48] can be reused for pruning the variables of the VARIABLES1 and the VARIABLES2 collection. This algorithm does not restrict the noccurrence variables of the VALUES collection.

**See also**

implies: global_cardinality, same.

related: k_alldifferent (two overlapping alldifferent plus restriction on values).

specialisation: same_and_global_cardinality_low_up (variable replaced by fixed interval).

**Keywords**

application area: assignment.

combinatorial object: permutation, multiset.

constraint arguments: constraint between two collections of variables.

constraint type: value constraint.

filtering: flow.

modelling: equality between multisets.

problems: demand profile.
Arc input(s)  VARIABLES1 VARIABLES2
Arc generator  PRODUCT→collection(variables1,variables2)
Arc arity  2
Arc constraint(s)  variables1.var = variables2.var
Graph property(ies)
- for all connected components: \( \text{NSOURCE} = \text{NSINK} \)
- \( \text{NSOURCE} = |\text{VARIABLES1}| \)
- \( \text{NSINK} = |\text{VARIABLES2}| \)

For all items of VALUES:

Arc input(s)  VARIABLES1
Arc generator  SELF→collection(variables)
Arc arity  1
Arc constraint(s)  variables.var = VALUES.val
Graph property(ies)  \( \text{NVERTEX} = \text{VALUES.noccurrence} \)

Graph model
Parts (A) and (B) of Figure 5.520 respectively show the initial and final graph associated with the first graph constraint of the Example slot. Since we use the \( \text{NSOURCE} \) and \( \text{NSINK} \) graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint.
Figure 5.520: Initial and final graph of the same_and_global_cardinality constraint
5.300  same_and_global_cardinality_low_up

**Description**
- Derived from same and global_cardinality_low_up

**Constraint**
- same_and_global_cardinality_low_up(VARIABLES1, VARIABLES2, VALUES)

**Arguments**
- VARIABLES1: collection(var−dvar)
- VARIABLES2: collection(var−dvar)
- VALUES: collection(val−int, omin−int, omax−int)

**Restrictions**
- |VARIABLES1| = |VARIABLES2|
- required(VARIABLES1, var)
- required(VARIABLES2, var)
- required(VALUES, [val, omin, omax])
- distinct(VALUES, val)
- VALUES.omin ≥ 0
- VALUES.omax ≤ |VARIABLES1|
- VALUES.omin ≤ VALUES.omax

The variables of the VARIABLES2 collection correspond to the variables of the VARIABLES1 collection according to a permutation. In addition, each value VALUES[i].val (with i ∈ [1, |VALUES|]) should be taken by at least VALUES[i].omin and at most VALUES[i].omax variables of the VARIABLES1 collection. Finally, each variable of VARIABLES1 should be assigned a value of VALUES[i].val (with i ∈ [1, |VALUES|]).

**Example**

\[
\begin{pmatrix}
\text{var} - 1, \\
\text{var} - 9, \\
\text{var} - 1, \\
\text{var} - 5, \\
\text{var} - 2, \\
\text{var} - 1, \\
\text{var} - 9, \\
\text{var} - 1, \\
\text{var} - 1, \\
\text{var} - 2, \\
\text{var} - 5 \\
\end{pmatrix}
\begin{pmatrix}
\text{val} - 1 \omin - 2 \Om - 3, \\
\text{val} - 2 \omin - 1 \Om - 1, \\
\text{val} - 5 \omin - 1 \Om - 1, \\
\text{val} - 7 \omin - 0 \Om - 2, \\
\text{val} - 9 \omin - 1 \Om - 1 \\
\end{pmatrix}
\]

The same_and_global_cardinality_low_up constraint holds since:
- The values 1, 9, 1, 5, 2, 1 assigned to |VARIABLES1| correspond to a permutation of the values 9, 1, 1, 1, 2, 5 assigned to |VARIABLES2|. 
- The values 1, 2, 5, 6 and 7 are respectively used 3 \((2 \leq 3 \leq 3)\), 1 \((1 \leq 1 \leq 1)\), 1 \((1 \leq 1 \leq 1)\), 0 \((0 \leq 0 \leq 2)\) and 1 \((1 \leq 1 \leq 1)\) times.

### Typical

\[
|\text{VARIABLES1}| > 1 \\
\text{range}(\text{VARIABLES1}.\text{var}) > 1 \\
\text{range}(\text{VARIABLES2}.\text{var}) > 1 \\
|\text{VALUES}| > 1 \\
\text{VALUES}.\text{omin} \leq |\text{VARIABLES1}| \\
\text{VALUES}.\text{omax} > 0 \\
\text{VALUES}.\text{omax} < |\text{VARIABLES1}| \\
|\text{VARIABLES1}| > |\text{VALUES}|
\]

### Symmetries

- Arguments are permutable w.r.t. permutation \((\text{VARIABLES1}, \text{VARIABLES2})(\text{VALUES})\).
- Items of \text{VARIABLES1} are permutable.
- Items of \text{VARIABLES2} are permutable.
- An occurrence of a value of \text{VARIABLES1}.\text{var} or \text{VARIABLES2}.\text{var} that does not belong to \text{VALUES}.\text{val} can be replaced by any other value that also does not belong to \text{VALUES}.\text{val}.
- Items of \text{VALUES} are permutable.
- \text{VALUES}.\text{omin} can be decreased to any value \(\geq 0\).
- \text{VALUES}.\text{omax} can be increased to any value \(\leq |\text{VARIABLES1}|\).
- All occurrences of two distinct values in \text{VARIABLES1}.\text{var}, \text{VARIABLES2}.\text{var} or \text{VALUES}.\text{val} can be swapped; all occurrences of a value in \text{VARIABLES1}.\text{var}, \text{VARIABLES2}.\text{var} or \text{VALUES}.\text{val} can be renamed to any unused value.

### Arg. properties

Contractible wrt. \text{VALUES}.

### Usage

The same and global cardinality_low_up constraint can be used for modelling the following assignment problem with one single constraint. The organisation Doctors Without Borders has a list of doctors and a list of nurses, each of whom volunteered to go on one rescue mission. Each volunteer specifies a list of possible dates and each mission should include one doctor and one nurse. In addition we have for each date the minimum and maximum number of missions that should be effectively done. The task is to produce a list of pairs such that each pair includes a doctor and a nurse who are available on the same date and each volunteer appears in exactly one pair so that for each day we build the required number of missions.

### Algorithm

In [48], the flow network that was used to model the same constraint [45, 46] is extended to support the cardinalities. Then, algorithms are developed to compute arc-consistency and bound-consistency.

### See also

**generalisation:** same and global cardinality (fixed interval replaced by variable).  
**implies:** global cardinality_low_up, global cardinality_low_up_no_loop, same.
Keywords

application area: assignment.
combinatorial object: permutation, multiset.
constraint arguments: constraint between two collections of variables.
constraint type: value constraint.
filtering: bound-consistency, arc-consistency, flow.
modelling: equality between multisets.
problems: demand profile.
<table>
<thead>
<tr>
<th><strong>Arc input(s)</strong></th>
<th>VARIABLES1 VARIABLES2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arc generator</strong></td>
<td>( \text{PRODUCT} \rightarrow \text{collection}(\text{variables1}, \text{variables2}) )</td>
</tr>
<tr>
<td><strong>Arc arity</strong></td>
<td>2</td>
</tr>
<tr>
<td><strong>Arc constraint(s)</strong></td>
<td>\text{variables1}.\text{var} = \text{variables2}.\text{var}</td>
</tr>
<tr>
<td><strong>Graph property(ies)</strong></td>
<td></td>
</tr>
</tbody>
</table>
|                   | • for all connected components: \( \text{NSOURCE} = \text{NSINK} \)  
|                   | • \( \text{NSOURCE} = |\text{variables1}| \)  
|                   | • \( \text{NSINK} = |\text{variables2}| \) |

For all items of \( \text{VALUES} \):

<table>
<thead>
<tr>
<th><strong>Arc input(s)</strong></th>
<th>VARIABLES1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arc generator</strong></td>
<td>( \text{SELF} \rightarrow \text{collection}(\text{variables}) )</td>
</tr>
<tr>
<td><strong>Arc arity</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Arc constraint(s)</strong></td>
<td>\text{variables}.\text{var} = \text{VALUES}.\text{val}</td>
</tr>
<tr>
<td><strong>Graph property(ies)</strong></td>
<td></td>
</tr>
</tbody>
</table>
|                   | • \( \text{NVERTEX} \geq \text{VALUES}.\text{omin} \)  
|                   | • \( \text{NVERTEX} \leq \text{VALUES}.\text{omax} \) |

**Graph model**

Parts (A) and (B) of Figure 5.521 respectively show the initial and final graph associated with the first graph constraint of the **Example** slot. Since we use the \( \text{NSOURCE} \) and \( \text{NSINK} \) graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint.
Figure 5.521: Initial and final graph of the same_and_global_cardinality_low_up constraint
### 5.301 same_intersection

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from <code>same</code> and <code>common</code>.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td><code>same_intersection(VARIABLES1, VARIABLES2)</code></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td><code>VARIABLES1 : collection(var−dvar)</code></td>
<td><code>VARIABLES2 : collection(var−dvar)</code></td>
</tr>
<tr>
<td>Restrictions</td>
<td><code>required(VARIABLES1.var)</code></td>
<td><code>required(VARIABLES2.var)</code></td>
</tr>
<tr>
<td>Purpose</td>
<td>Each value, which occurs both in the <code>VARIABLES1</code> and in the <code>VARIABLES2</code> collections, has the same number of occurrences in <code>VARIABLES1</code> as well as in <code>VARIABLES2</code>.</td>
<td></td>
</tr>
</tbody>
</table>

![Example](image)

First note that the values, which occur both in `VARIABLES1 = ⟨1, 9, 1, 5, 2, 1⟩` as well as in `VARIABLES2 = ⟨9, 1, 1, 1, 3, 5, 8⟩` correspond to values 1, 5, and 9. Consequently, the `same_intersection` constraint holds since these values 1, 5, and 9 have the same number of occurrences in both collections (i.e., they respectively occur 3, 1, and 1 times within `VARIABLES1` and `VARIABLES2`).

**Typical**

- `|VARIABLES1| > 1`
- `range(VARIABLES1.var) > 1`
- `|VARIABLES2| > 1`
- `range(VARIABLES2.var) > 1`

**Symmetries**

- Arguments are permutable w.r.t. permutation `(VARIABLES1, VARIABLES2)`.  
- Items of `VARIABLES1` are permutable.  
- Items of `VARIABLES2` are permutable.  
- All occurrences of two distinct values in `VARIABLES1.var` or `VARIABLES2.var` can be swapped; all occurrences of a value in `VARIABLES1.var` or `VARIABLES2.var` can be renamed to any unused value.
See also

- common keyword: common, nvalue_on_intersection (constraint on the intersection).
- implied by: alldifferent_on_intersection, same.

Keywords

- constraint arguments: constraint between two collections of variables.
- constraint type: constraint on the intersection.
Arc input(s) VARIABLES1 VARIABLES2
Arc generator \( \text{PRODUCT} \rightarrow \text{collection}(\text{variables1, variables2}) \)
Arc arity 2
Arc constraint(s) \( \text{variables1.var} = \text{variables2.var} \)
Graph property(ies) for all connected components: \( \text{NSOURCE} = \text{NSINK} \)

Graph model

Parts (A) and (B) of Figure 5.522 respectively show the initial and final graph associated with the Example slot. The same_intersection constraint holds since each connected component of the final graph has the same number of sources and sinks. Note that all the vertices corresponding to the variables that take values 2, 3 or 8 were removed from the final graph since there is no arc for which the associated equality constraint holds.

![Graph model](image)

Figure 5.522: Initial and final graph of the same_intersection constraint
5.302 same_interval

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from same.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>same_interval(VARIABLES1, VARIABLES2, SIZE_INTERVAL)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VARIABLES1 : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES2 : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SIZE_INTERVAL : int</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Typical</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Let \( N_i \) (respectively \( M_i \)) denote the number of variables of the collection \( \text{VARIABLES1} \) (respectively \( \text{VARIABLES2} \)) that take a value in the interval \([\text{SIZE_INTERVAL} \cdot i, \text{SIZE_INTERVAL} \cdot i + \text{SIZE_INTERVAL} - 1]\). For all integer \( i \) we have \( N_i = M_i \).

In the example, the third argument \( \text{SIZE_INTERVAL} = 3 \) defines the following family of intervals \([3 \cdot k, 3 \cdot k + 2]\), where \( k \) is an integer. Consequently the values of the collection \((1, 7, 6, 0, 1, 7)\) are respectively located within intervals \([0, 2], [6, 8], [6, 8], [0, 2], [0, 2], [6, 8]\). Therefore intervals \([0, 2]\) and \([6, 8]\) are respectively used 3 and 3 times. Similarly, the values of the collection \((8, 8, 8, 0, 1, 2)\) are respectively located within intervals \([6, 8], [6, 8], [6, 8], [0, 2], [0, 2], [0, 2]\). As before intervals \([0, 2]\) and \([6, 8]\) are respectively used 3 and 3 times. Consequently the same_interval constraint holds. Figure 5.523 illustrates this correspondence.
Symmetries

- Arguments are **permutable** w.r.t. permutation (VAR1, VAR2) (SIZE_INTERVAL).
- Items of VAR1 are **permutable**.
- Items of VAR2 are **permutable**.
- An occurrence of a value of VAR that belongs to the $k$-th interval, of size SIZE_INTERVAL, can be **replaced** by any other value of the same interval.

Arg. properties

- Aggregate: VAR1(union), VAR2(union), SIZE_INTERVAL(id).

Algorithm

See algorithm of the **same** constraint.

Used in

- $k$ **same** interval.

See also

- **implies**: used_by_interval.
- **soft variant**: soft_same_interval_var (**variable-based violation measure**).
- **specialisation**: same (**variable/constant replaced by variable**).
- **system of constraints**: k_same_interval.

Keywords

- **characteristic of a constraint**: sort based reformulation.
- **combinatorial object**: permutation.
- **constraint arguments**: constraint between two collections of variables.
- **modelling**: interval.

---

Figure 5.523: Correspondence between the intervals associated with collection \( \{1, 7, 6, 0, 1, 7\} \) and with collection \( \{8, 8, 0, 1, 2\} \)
Arc input(s)  VARIABLES1 VARIABLES2
Arc generator  \( \text{PRODUCT} \rightarrow \text{collection}(\text{variables1,variables2}) \)
Arc arity  2
Arc constraint(s)  \( \text{variables1}.\text{var}/\text{SIZE\_INTERVAL} = \text{variables2}.\text{var}/\text{SIZE\_INTERVAL} \)
Graph property(ies)  • for all connected components: \( \text{NSOURCE} = \text{NSINK} \)
• \( \text{NSOURCE} = |\text{VARIABLES1}| \)
• \( \text{NSINK} = |\text{VARIABLES2}| \)

Graph model

Parts (A) and (B) of Figure 5.524 respectively show the initial and final graph associated with the Example slot. Since we use the \text{NSOURCE} and \text{NSINK} graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. The same interval constraint holds since:

• Each connected component of the final graph has the same number of sources and of sinks.
• The number of sources of the final graph is equal to \( |\text{VARIABLES1}| \).
• The number of sinks of the final graph is equal to \( |\text{VARIABLES2}| \).

Signature

Since the initial graph contains only sources and sinks, and since isolated vertices are eliminated from the final graph, we make the following observations:

• Sources of the initial graph cannot become sinks of the final graph,
• Sinks of the initial graph cannot become sources of the final graph.

From the previous observations and since we use the \text{PRODUCT} arc generator on the collections \text{VARIABLES1} and \text{VARIABLES2}, we have that the maximum number of sources and sinks of the final graph is respectively equal to \( |\text{VARIABLES1}| \) and \( |\text{VARIABLES2}| \). Therefore we can rewrite \( \text{NSOURCE} = |\text{VARIABLES1}| \) to \( \text{NSOURCE} \geq |\text{VARIABLES1}| \) and simplify \( \text{NSOURCE} \) to \( \text{NSINK} \). In a similar way, we can rewrite \( \text{NSINK} = |\text{VARIABLES2}| \) to \( \text{NSINK} \geq |\text{VARIABLES2}| \) and simplify \( \text{NSINK} \) to \( \text{NSINK} \).
Figure 5.524: Initial and final graph of the same_interval constraint
5.303 same modulo

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from same.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>same modulo(VARIABLES1, VARIABLES2, M)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VARIABLES1 : collection(var–dvar)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES2 : collection(var–dvar)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M : int</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>For each integer R in [0, M − 1], let N1R (respectively N2R) denote the number of variables of VARIABLES1 (respectively VARIABLES2) that have R as a rest when divided by M. For all R in [0, M − 1] we have that N1R = N2R.</td>
<td></td>
</tr>
</tbody>
</table>

The values of the first collection \(\langle \text{var}−1, \text{var}−9, \text{var}−1, \text{var}−5, \text{var}−2, \text{var}−1, \text{var}−6, \text{var}−4, \text{var}−1, \text{var}−1, \text{var}−5, \text{var}−5 \rangle\) are respectively associated with the equivalence classes 1 mod 3 = 1, 9 mod 3 = 0, 1 mod 3 = 1, 5 mod 3 = 2, 2 mod 3 = 2, 1 mod 3 = 1. Therefore the equivalence classes 0, 1, and 2 are respectively used 1, 3, and 2 times. Similarly, the values of the second collection \(\langle 6,4,1,1,5,5 \rangle\) are respectively associated with the equivalence classes 6 mod 3 = 0, 4 mod 3 = 1, 1 mod 3 = 1, 1 mod 3 = 1, 5 mod 3 = 2, 5 mod 3 = 2. Therefore the equivalence classes 0, 1, and 2 are respectively used 1, 3, and 2 times. Consequently the same modulo constraint holds. Figure 5.525 illustrates this correspondence.
Symmetries

- Arguments are permutable w.r.t. permutation \( (\text{VARIABLES1}, \text{VARIABLES2}) \mod M \).
- Items of \( \text{VARIABLES1} \) are permutable.
- Items of \( \text{VARIABLES2} \) are permutable.
- An occurrence of a value \( u \) of \( \text{VARIABLES}.\text{var} \) can be replaced by any other value \( v \) such that \( v \) is congruent to \( u \) modulo \( M \).

Arg. properties

Aggregate: \( \text{VARIABLES1}(\text{union}), \text{VARIABLES2}(\text{union}), M(\text{id}) \).

Used in

\( k\_same\_modulo \).

See also

implies: \( used\_by\_modulo \).
soft variant: \( soft\_same\_modulo\_var \) (\( variable\_based\_violation\_measure \)).
specialisation: \( same \) (\( variable \mod constant \ replaced \ by \ variable \)).
system of constraints: \( k\_same\_modulo \).

Keywords

characteristic of a constraint: sort based reformulation, modulo.
combinatorial object: permutation.
constraint arguments: constraint between two collections of variables.

---

### Figure 5.525: Correspondence between the equivalence classes associated with collection \( \{1, 9, 1, 5, 2, 1\} \) and with collection \( \{6, 4, 1, 1, 5, 5\} \)
Arc input(s) | VARIABLES1, VARIABLES2
---|---
Arc generator | \( \text{PRODUCT} \rightarrow \text{collection}(\text{variables1, variables2}) \)
Arc arity | 2
Arc constraint(s) | variables1.var mod M = variables2.var mod M
Graph property(ies) | • for all connected components: NSOURCE = NSINK  
• NSOURCE = |VARIABLES1|  
• NSINK = |VARIABLES2|

**Graph model**

Parts (A) and (B) of Figure 5.526 respectively show the initial and final graph associated with the Example slot. Since we use the NSOURCE and NSINK graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. The same modulo constraint holds since:

• Each connected component of the final graph has the same number of sources and of sinks.
• The number of sources of the final graph is equal to |VARIABLES1|.
• The number of sinks of the final graph is equal to |VARIABLES2|.

**Signature**

Since the initial graph contains only sources and sinks, and since isolated vertices are eliminated from the final graph, we make the following observations:

• Sources of the initial graph cannot become sinks of the final graph,
• Sinks of the initial graph cannot become sources of the final graph.

From the previous observations and since we use the PRODUCT arc generator on the collections VARIABLES1 and VARIABLES2, we have that the maximum number of sources and sinks of the final graph is respectively equal to |VARIABLES1| and |VARIABLES2|. Therefore we can rewrite NSOURCE = |VARIABLES1| to NSOURCE \( \geq |\text{VARIABLES1}| \) and simplify NSINK to NSINK. In a similar way, we can rewrite NSINK = |VARIABLES2| to NSINK \( \geq |\text{VARIABLES2}| \) and simplify NSINK to NSINK.
Figure 5.526: Initial and final graph of the same_modulo constraint
5.304 same_partition

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from same.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>same_partition(VARIABLES1, VARIABLES2, PARTITIONS)</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>VALUES : collection(val–int)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VARIABLES1 : collection(var–dvar)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES2 : collection(var–dvar)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PARTITIONS : collection(p–VALUES)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>For each integer i in [1,</td>
<td>PARTITIONS</td>
</tr>
</tbody>
</table>
| Example     | \[
\begin{pmatrix}
  \text{var} - 1, \\
  \text{var} - 2, \\
  \text{var} - 6, \\
  \text{var} - 3, \\
  \text{var} - 1, \\
  \text{var} - 2, \\
  \text{var} - 6, \\
  \text{var} - 3, \\
  \text{var} - 1, \\
  \text{var} - 3 \\
\end{pmatrix},
\]

The different values of the collection \( \langle 1, 2, 6, 3, 1, 2 \rangle \) are respectively associated with the partitions \( p = \langle 1, 3 \rangle, p = \langle 2, 6 \rangle, p = \langle 2, 6 \rangle, p = \langle 1, 3 \rangle, p = \langle 1, 3 \rangle, \) and \( p = \langle 2, 6 \rangle \). Therefore partitions \( p = \langle 1, 3 \rangle \) and \( p = \langle 2, 6 \rangle \) are respectively used 3 and 3 times. Similarly,
the different values of the collection \(\{6, 6, 2, 3, 1, 3\}\) are respectively associated with the partitions \(p - \langle 2, 6 \rangle, p - \langle 2, 6 \rangle, p - \langle 2, 6 \rangle, p - \langle 1, 3 \rangle, p - \langle 1, 3 \rangle, \) and \(p - \langle 1, 3 \rangle\). As before partitions \(p - \langle 1, 3 \rangle\) and \(p - \langle 2, 6 \rangle\) are respectively used 3 and 3 times. Consequently the same partition constraint holds. Figure 5.527 illustrates this correspondence.

![Figure 5.527: Correspondence between the partitions associated with collection \(\{1, 2, 6, 3, 1, 2\}\) and with collection \(\{6, 6, 2, 3, 1, 3\}\)](image)

**Typical**

- \(|\text{VARIABLES1}| > 1\)
- \(\text{range}(\text{VARIABLES1}.\text{var}) > 1\)
- \(\text{range}(\text{VARIABLES2}.\text{var}) > 1\)
- \(|\text{VARIABLES1}| > |\text{PARTITIONS}|\)
- \(|\text{VARIABLES2}| > |\text{PARTITIONS}|\)

**Symmetries**

- Arguments are *permutable* w.r.t. permutation \((\text{VARIABLES1}, \text{VARIABLES2})\) \((\text{PARTITIONS})\).
- Items of \(\text{VARIABLES1}\) are *permutable*.
- Items of \(\text{VARIABLES2}\) are *permutable*.
- Items of \(\text{PARTITIONS}\) are *permutable*.
- Items of \(\text{PARTITIONS}.p\) are *permutable*.
- An occurrence of a value of \(\text{VARIABLES}.\text{var}\) can be replaced by any other value that also belongs to the same partition of \(\text{PARTITIONS}\).

**Arg. properties**

Aggregate: \(\text{VARIABLES1}(\text{union}), \text{VARIABLES2}(\text{union}), \text{PARTITIONS}(\text{id})\).

**Used in**

- \(k._{\text{same partition}}\).

**See also**

- implies: \(\text{used by partition}\).
- soft variant: \(\text{soft same partition var} (\text{variable-based violation measure})\).
- specialisation: \(\text{same (variable ∈ partition replaced by variable)}\).
- system of constraints: \(k._{\text{same partition}}\).
- used in graph description: \(\text{in same partition}\).

**Keywords**

- characteristic of a constraint: sort based reformulation, partition.
- combinatorial object: permutation.
- constraint arguments: constraint between two collections of variables.
**Arc input(s)**

VARIABLES1 VARIABLE2

**Arc generator**

\( \text{PRODUCT} \rightarrow \text{collection}(\text{variables1}, \text{variables2}) \)

**Arc arity**

2

**Arc constraint(s)**

in_same_partition(\text{variables1.var}, \text{variables2.var}, \text{PARTITIONS})

**Graph property(ies)**

- for all connected components: NSOURCE = NSINK
- NSOURCE = |VARIABLES1|
- NSINK = |VARIABLES2|

**Graph model**

Parts (A) and (B) of Figure 5.528 respectively show the initial and final graph associated with the Example slot. Since we use the NSOURCE and NSINK graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. The same partition constraint holds since:

- Each connected component of the final graph has the same number of sources and of sinks.
- The number of sources of the final graph is equal to |VARIABLES1|.
- The number of sinks of the final graph is equal to |VARIABLES2|.

**Signature**

Since the initial graph contains only sources and sinks, and since isolated vertices are eliminated from the final graph, we make the following observations:

- Sources of the initial graph cannot become sinks of the final graph,
- Sinks of the initial graph cannot become sources of the final graph.

From the previous observations and since we use the PRODUCT arc generator on the collections VARIABLES1 and VARIABLES2, we have that the maximum number of sources and sinks of the final graph is respectively equal to |VARIABLES1| and |VARIABLES2|. Therefore we can rewrite NSOURCE = |VARIABLES1| to NSOURCE ≥ |VARIABLES1| and simplify NSOURCE to NSINK. In a similar way, we can rewrite NSINK = |VARIABLES2| to NSINK ≥ |VARIABLES2| and simplify NSINK to NSINK.
Figure 5.528: Initial and final graph of the same partition constraint
### 5.305 **same_sign**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Arithmetic.</td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>same_sign(VAR1, VAR2)</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>VAR1: dvar, VAR2: dvar</td>
</tr>
<tr>
<td><strong>Restriction</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>Enforce the fact that the product of the first and second variables is greater than or equal to 0.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>(7, 1)</td>
</tr>
</tbody>
</table>

The `same_sign` constraint holds since 7 and 1 have the same sign.

<table>
<thead>
<tr>
<th><strong>Typical</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>VAR1 ≠ 0</td>
<td></td>
</tr>
<tr>
<td>VAR2 ≠ 0</td>
<td></td>
</tr>
</tbody>
</table>

| **Symmetry** | Arguments are permutable w.r.t. permutation (VAR1, VAR2). |
| **See also** | comparison swapped: opposite_sign, implied by: divisible, eq. |
| **Keywords** | constraint arguments: binary constraint, constraint type: predefined constraint, arithmetic constraint, filtering: arc-consistency. |
5.306 scalar_product

DESCRIPTION

Origin
Arithmetic constraint.

Constraint
scalar_product(LINEARTERM, CTR, VAL)

Synonyms
equation, linear, sum_weight, weightedSum.

Arguments
LINEARTERM : collection(coef−int, var−dvar)
CTR : atom
VAL : dvar

Restrictions
required(LINEARTERM, [coef, var])
CTR ∈ [=, ≠, <, ≥, >, ≤]

Purpose
Constraint a linear term defined as the sum of products of coefficients and variables. More precisely, let $S$ denote the sum of the product between a coefficient and its variable of the different items of the LINEARTERM collection. Enforce the following constraint to hold: $S$ CTR VAL.

Example

\[
\begin{pmatrix}
\text{coeff} - 1 & \text{var} - 1, \\
\text{coeff} - 3 & \text{var} - 1, \\
\text{coeff} - 4 & \text{var} - 4
\end{pmatrix}
\]

The scalar_product constraint holds since the condition $1 \cdot 1 + 3 \cdot 1 + 1 \cdot 4 = 8$ is satisfied.

Typical

|LINEARTERM| > 1
range(LINEARTERM.coeff) > 1
range(LINEARTERM.var) > 1
CTR ∈ [=, <, ≥, >, ≤]

Symmetries

- Items of LINEARTERM are permutable.
- Attributes of LINEARTERM are permutable w.r.t. permutation (coef, var) (permutation not necessarily applied to all items).

Arg. properties

- Contractible w.r.t. LINEARTERM when CTR ∈ [<, ≤], minval(LINEARTERM.coeff) ≥ 0 and minval(LINEARTERM.var) ≥ 0.
- Extensible w.r.t. LINEARTERM when CTR ∈ [≥, >], minval(LINEARTERM.coeff) ≥ 0 and minval(LINEARTERM.var) ≥ 0.
- Aggregate: LINEARTERM(union), CTR(id), VAL(+).
Remark

The scalar_product constraint is called linear in Gecode (http://www.gecode.org/). It is called sum_weight in JaCoP (http://www.jacop.eu/). In the 2008 CSP solver competition the scalar_product constraint was called weightedSum and required VAL to be fixed.

Algorithm

Most filtering algorithms first merge multiple occurrences of identical variables in order to potentially make more deductions. When CTR corresponds to the less than or equal to constraint, a filtering algorithm achieving bound-consistency for the scalar_product constraint with large numbers of variables is described in [188].

Systems

equation in Choco, linear in Gecode, sumweight in JaCoP, scalar_product in SICStus.

See also

specialisation: sum_ctr (arithmetic constraint where all coefficients are equal to 1).

Keywords

characteristic of a constraint: sum.
constraint type: predefined constraint, arithmetic constraint.
filtering: duplicated variables.
5.307  sequence_folding

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>J. Pearson</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>sequence_folding(LETTERS)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>LETTERS : collection(index-int, next-dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>Express the fact that a sequence is folded in a way that no crossing occurs. A sequence is modelled by a collection of letters. For each letter $l_1$ of a sequence, we indicate the next letter $l_2$ located after $l_1$ that is directly in contact with $l_1$ ($l_1$ itself if such a letter does not exist).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Typical     | $|\text{LETTERS}| > 2$  
$\text{range}(|\text{LETTERS}\_\text{next}|) > 1$ |
| Usage       | Motivated by RNA folding [154]. |
| Keywords    | application area: bioinformatics.  
characteristic of a constraint: automaton.  
reified automaton constraint.  
combinatorial object: sequence.  
constraint type: decomposition.  
geometry: geometrical constraint. |
Figure 5.529: Folded sequence associated with the example
Graph model

Parts (A) and (B) of Figure 5.530 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

![Diagrams](image-url)

(A) (B)

Figure 5.530: Initial and final graph of the sequence_folding constraint

Signature

Consider the first graph constraint. Since we use the SELF arc generator on the LETTERS collection the maximum number of arcs of the final graph is equal to \(|\text{LETTERS}|\). Therefore
we can rewrite the graph property $\text{NARC} = |\text{LETTERS}|$ to \( \text{NARC} \geq |\text{LETTERS}| \) and simplify $\text{NARC}$ to $\text{NARC}$.

Consider now the second graph constraint. Since we use the $\text{CLIQUE}(<)$ arc generator on the $\text{LETTERS}$ collection the maximum number of arcs of the final graph is equal to $|\text{LETTERS}| \cdot (|\text{LETTERS}| - 1)/2$. Therefore we can rewrite the graph property $\text{NARC} = |\text{LETTERS}| \cdot (|\text{LETTERS}| - 1)/2$ to $\text{NARC} \geq |\text{LETTERS}| \cdot (|\text{LETTERS}| - 1)/2$ and simplify $\text{NARC}$ to $\text{NARC}$. 
Figure 5.531 depicts the automaton associated with the \textit{sequence\_folding} constraint. Consider the \(i^{th}\) and the \(j^{th}\) \((i < j)\) items of the collection \textit{LETTERS}. Let \(\text{INDEX}_i\) and \(\text{NEXT}_i\) respectively denote the \textit{index} and the \textit{next} attributes of the \(i^{th}\) item of the collection \textit{LETTERS}. Similarly, let \(\text{INDEX}_j\) and \(\text{NEXT}_j\) respectively denote the \textit{index} and the \textit{next} attributes of the \(j^{th}\) item of the collection \textit{LETTERS}. To each quadruple \((\text{INDEX}_i, \text{NEXT}_i, \text{INDEX}_j, \text{NEXT}_j)\) corresponds a signature variable \(S_{i,j}\), which takes its value in \(\{0, 1, 2\}\), as well as the following signature constraint:

\[
\begin{align*}
(\text{INDEX}_i \leq \text{NEXT}_i) & \land (\text{INDEX}_j \leq \text{NEXT}_j) \land (\text{NEXT}_i \leq \text{NEXT}_j) \Leftrightarrow S_{i,j} = 0 \land \\
(\text{INDEX}_i \leq \text{NEXT}_i) & \land (\text{INDEX}_j \leq \text{NEXT}_j) \land (\text{NEXT}_i > \text{INDEX}_j) \land (\text{NEXT}_j \leq \text{NEXT}_i) \Leftrightarrow S_{i,j} = 1.
\end{align*}
\]

Figure 5.531: Automaton of the \textit{sequence\_folding} constraint
5.308  set_value_precede

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[240]</td>
</tr>
</tbody>
</table>

**Constraint**

\[
\text{set_value_precede}(S, T, \text{VARIABLES})
\]

**Arguments**

\[
\begin{align*}
S & : \text{int} \\
T & : \text{int} \\
\text{VARIABLES} & : \text{collection}(\text{var}−\text{svar})
\end{align*}
\]

**Restrictions**

\[
S \neq T \\
\text{required}(\text{VARIABLES}, \text{var})
\]

**Purpose**

If there exists a set variable \(v_1\) of \(\text{VARIABLES}\) such that \(S\) does not belong to \(v_1\) and \(T\) does, then there also exists a set variable \(v_2\) preceding \(v_1\) such that \(S\) belongs to \(v_2\) and \(T\) does not.

**Example**

\[
\begin{align*}
2, 1, \\
\{\text{var}−\{0, 2\}, \text{var}−\{0, 1\}, \text{var}−\emptyset, \text{var}−\{1\}\} \\
0, 1, \\
\{\text{var}−\{0, 2\}, \text{var}−\{0, 1\}, \text{var}−\emptyset, \text{var}−\{1\}\} \\
0, 2, \\
\{\text{var}−\{0, 1\}, \text{var}−\emptyset, \text{var}−\{1\}\} \\
0, 4, \\
\{\text{var}−\{0, 2\}, \text{var}−\{0, 1\}, \text{var}−\emptyset, \text{var}−\{1\}\}
\end{align*}
\]

The following examples are taken from [239, page 58]:

- The \textit{set_value_precede}(2, 1, \{\{0, 2\}, \{0, 1\}, \emptyset, \{1\}\}) constraint holds since the first occurrence of value 2 precedes the first occurrence of value 1 (i.e., the set \{0, 2\} occurs before the set \{0, 1\}).
- The \textit{set_value_precede}(0, 1, \{\{0, 2\}, \{0, 1\}, \emptyset, \{1\}\}) constraint holds since the first occurrence of value 0 precedes the first occurrence of value 1 (i.e., the set \{0, 2\} occurs before the set \{0, 1\}).
- The \textit{set_value_precede}(0, 2, \{\{0, 2\}, \{0, 1\}, \emptyset, \{1\}\}) constraint holds since “there is no set in \{\{0, 2\}, \{0, 1\}, \emptyset, \{1\}\} that contains 2 but not 0”.
- The \textit{set_value_precede}(0, 4, \{\{0, 2\}, \{0, 1\}, \emptyset, \{1\}\}) constraint holds since no set in \{\{0, 2\}, \{0, 1\}, \emptyset, \{1\}\} contains value 4.
Typical

\[ S < T \]
\[ |VARIABLES| > 1 \]

Arg. properties

Suffix-contractible wrt. VARIABLES.

Algorithm

A filtering algorithm for maintaining value precedence on a sequence of set variables is presented in [240]. Its complexity is linear to the number of variables of the collection VARIABLES.

Systems

precede in Gecode.

See also

specialisation: int_value_precede(sequence of set variables replaced by sequence of domain variables).

Keywords

constraint arguments: constraint involving set variables.
constraint type: order constraint.
symmetry: symmetry, indistinguishable values, value precedence.
### 5.309 shift

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>N. Beldiceanu</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>( \text{shift}(\text{MIN_BREAK}, \text{MAX_RANGE}, \text{TASKS}) )</td>
<td></td>
</tr>
</tbody>
</table>
| Arguments   | \( \text{MIN\_BREAK} : \text{int} \)  
\( \text{MAX\_RANGE} : \text{int} \)  
\( \text{TASKS} : \text{collection}(\text{origin} - \text{dvar}, \text{end} - \text{dvar}) \) |       |
| Restrictions| \( \text{MIN\_BREAK} > 0 \)  
\( \text{MAX\_RANGE} > 0 \)  
\( \text{required}(\text{TASKS}, [\text{origin}, \text{end}]) \)  
\( \text{TASKS.\:origin} < \text{TASKS.end} \) |       |

The difference between the end of the last task of a *shift* and the origin of the first task of a *shift* should not exceed the quantity \( \text{MAX\_RANGE} \). Two tasks \( t_1 \) and \( t_2 \) belong to the *same shift* if at least one of the following conditions is true:

- Task \( t_2 \) starts after the end of task \( t_1 \) at a distance that is less than or equal to the quantity \( \text{MIN\_BREAK} \).
- Task \( t_1 \) starts after the end of task \( t_2 \) at a distance that is less than or equal to the quantity \( \text{MIN\_BREAK} \).
- Task \( t_1 \) overlaps task \( t_2 \).

#### Purpose

Figure 5.532 represents the different tasks of the example. Each task is drawn as a rectangle with its corresponding \( \text{id} \) attribute in the middle. We indicate the distance between two consecutive tasks of a same shift and note that it is less than or equal to \( \text{MIN\_BREAK} = 6 \). Since each shift has a range that is less than or equal to \( \text{MAX\_RANGE} = 8 \), the shift constraint holds (the *range* of a shift is the difference between the end of the last task of the shift and the origin of the first task of the shift).

#### Typical

\( \text{MIN\_BREAK} > 1 \)  
\( \text{MAX\_RANGE} > 1 \)  
\( \text{MIN\_BREAK} < \text{MAX\_RANGE} \)  
\( |\text{TASKS}| > 2 \)

#### Symmetries

- Items of \( \text{TASKS} \) are permutable.
- One and the same constant can be added to the \( \text{origin} \) attribute of all items of \( \text{TASKS} \).
The shift constraint can be used in machine scheduling problems where one has to shut down a machine for maintenance purpose after a given maximum utilisation of that machine. In this case the $\text{MAX\_RANGE}$ parameter indicates the maximum possible utilisation of the machine before maintenance, while the $\text{MIN\_BREAK}$ parameter gives the minimum time needed for maintenance.

The shift constraint can also be used for timetabling problems where the rest period of a person can move in time. In this case $\text{MAX\_RANGE}$ indicates the maximum possible working time for a person, while $\text{MIN\_BREAK}$ specifies the minimum length of the break that follows a working time period.

**Usage**

**See also**

- **common keyword**: `sliding_time_window` (*temporal constraint*).
- **used in graph description**: `range_crt`.

**Keywords**

- **constraint type**: scheduling constraint, timetabling constraint, temporal constraint.

Figure 5.532: The two shifts of the example
<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>TASKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>SELF $\mapsto$ collection(tasks)</td>
</tr>
<tr>
<td>Arc arity</td>
<td>1</td>
</tr>
</tbody>
</table>
| Arc constraint(s) | • tasks.end $\geq$ tasks.origin  
|                   | • tasks.end $-$ tasks.origin $\leq$ MAX_RANGE |
| Graph property(ies) | NARC $=$ |TASKS| |

<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>TASKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>CLIQUE $\mapsto$ collection(tasks1, tasks2)</td>
</tr>
<tr>
<td>Arc arity</td>
<td>2</td>
</tr>
</tbody>
</table>
| Arc constraint(s) | $\bigvee \left( \begin{array}{l} 
\land \left( \begin{array}{l} 
\text{tasks2.origin} \geq \text{tasks1.end}, \\
\text{tasks2.origin} - \text{tasks1.end} \leq \text{MINBREAK} 
\end{array} \right), \\
\land \left( \begin{array}{l} 
\text{tasks1.origin} \geq \text{tasks2.end}, \\
\text{tasks1.origin} - \text{tasks2.end} \leq \text{MINBREAK} 
\end{array} \right), \\
\text{tasks2.origin} < \text{tasks1.end} \land \text{tasks1.origin} < \text{tasks2.end} 
\end{array} \right) \right) 
| Sets | CC $\mapsto$ 
| | \[ \text{variables} - \text{col} \left( \begin{array}{l} 
\text{VARIABLES} - \text{collection}([\text{var} - \text{dvar}], \\
\text{item}[\text{var} - \text{TASKS}.\text{origin}], \\
\text{item}[\text{var} - \text{TASKS}.\text{end}]) 
\end{array} \right) \] 
| Constraint(s) on sets | range_ctr(variables $\leq$ MAX_RANGE) |

**Graph model**

The first graph constraint enforces the following two constraints between the attributes of each task:

- The end of a task should not be situated before its start,
- The duration of a task should not be greater than the MAX_RANGE parameter.

The second graph constraint decomposes the final graph in connected components where each component corresponds to a given shift. Finally, the **Constraint(s) on sets** slot restricts the stretch of each shift.

Parts (A) and (B) of Figure 5.533 respectively show the initial and final graph associated with the second graph constraint of the **Example** slot. Since we use the set generator CC, we show the two connected components of the final graph. They respectively correspond to the two shifts that are displayed in Figure 5.532.

**Signature**

Consider the first graph constraint. Since we use the **SELF** arc generator on the TASKS collection, the maximum number of arcs of the final graph is equal to |TASKS|. Therefore, we can rewrite the graph property NARC $=$ |TASKS| to NARC $\geq$ |TASKS| and simplify NARC to NARC.
Figure 5.533: Initial and final graph of the shift constraint
### 5.310 sign_of

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Arithmetic.</td>
</tr>
<tr>
<td>Constraint</td>
<td>sign_of(S, X)</td>
</tr>
<tr>
<td>Usual name</td>
<td>sign</td>
</tr>
</tbody>
</table>
| Arguments               | $S : \text{dvar}$
|                         | $X : \text{dvar}$ |
| Restrictions            | $S \geq -1$
|                         | $S \leq 1$ |

According to the value of the first variable $S$, restrict the sign of the second variable $X$:
- When $S = -1$, $X$ should be negative (i.e., $X < 0$).
- When $S = 0$, $X$ is also equal to 0.
- When $S = +1$, $X$ should be positive (i.e., $X > 0$).

**Example**

- The first sign_of constraint holds since $S = -1$ and $X = -8$ is negative.
- The second sign_of constraint holds since $S = 0$ and $X = 0$ is neither negative, neither positive.
- The second sign_of constraint holds since $S = +1$ and $X = 8$ is positive.

**Typical**

- $S \neq 0$
- $X \neq 0$

**Arg. properties**

- Functional dependency: $S$ determined by $X$.

**See also**

- implies: $\text{geq}$.

**Keywords**

- constraint arguments: binary constraint, pure functional dependency.
- constraint type: predefined constraint, arithmetic constraint.
- filtering: arc-consistency.
### 5.311 `size_max_seq_alldifferent`

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>N. Beldiceanu</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td><code>size_max_seq_alldifferent(SIZE, VARIABLES)</code></td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td><code>size_maximal_sequence_alldiff</code>, <code>size_maximal_sequence_alldistinct</code>, <code>size_maximal_sequence_alldifferent</code>.</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>SIZE : <code>dvar</code></td>
<td></td>
</tr>
<tr>
<td>VARIABLES : <code>collection(var−dvar)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>SIZE ≥ 0</td>
<td></td>
</tr>
<tr>
<td>SIZE ≤</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARIABLES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>required(VARIABLES, var)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>SIZE is the size of the maximal sequence (among all possible sequences of consecutive variables of the collection VARIABLES) for which the alldifferent constraint holds.</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
See also

**common keyword:** alldifferent, open_alldifferent.
size_max_starting_seq_alldifferent(all_different,disequality).
implies: atleast_nvalue.

**Keywords**

**characteristic of a constraint:** all different, disequality, hypergraph.
**combinatorial object:** sequence.
**constraint arguments:** pure functional dependency.
**constraint type:** sliding sequence constraint, conditional constraint.
**modelling:** functional dependency.
<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>$PATH_{N} \rightarrow \text{collection}$</td>
</tr>
<tr>
<td>Arc arity</td>
<td>*</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>$\text{alldifferent(collection)}$</td>
</tr>
<tr>
<td>Graph property(ies)</td>
<td>$\text{NARC} = \text{SIZE}$</td>
</tr>
</tbody>
</table>

**Graph model**

Note that this is an example of global constraint where the arc constraints do not have the same arity. However, they correspond to the same type of constraint.
### 5.312 size_max_starting_seq_alldifferent

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
</table>

#### Origin
Inspired by `size_max_seq_alldifferent`.

#### Constraint
`size_max_starting_seq_alldifferent(SIZE, VARIABLES)`

#### Synonyms
- `size_maximal_starting_sequence_alldiff`
- `size_maximal_starting_sequence_alldistinct`
- `size_maximal_starting_sequence_alldifferent`

#### Arguments

- `SIZE` : `dvar`
- `VARIABLES` : `collection(var−dvar)`

#### Restrictions

- `SIZE ≥ 0`
- `SIZE ≤ |VARIABLES|`
- `required(VARIABLES, var)`

#### Purpose
`SIZE` is the size of the maximal sequence (among all sequences of consecutive variables of the collection `VARIABLES` starting at position one) for which the `alldifferent` constraint holds.

#### Example

```
( var − 9,
  var − 2,
  var − 4,
  var − 5,
  var − 2,
  var − 7,
  var − 4 )
```

The `size_max_starting_seq_alldifferent` constraint holds since the constraint `alldifferent((var − 9, var − 2, var − 4, var − 5))` holds and since `alldifferent((var − 9, var − 2, var − 4, var − 5, var − 2))` does not hold.

#### Typical

- `SIZE > 2`
- `SIZE < |VARIABLES|`
- `range(VARIABLES, var) > 1`

#### Symmetry
One and the same constant can be added to the `var` attribute of all items of `VARIABLES`.

#### Arg. properties
Functional dependency: `SIZE` determined by `VARIABLES`.

#### Remark
A conditional constraint [266] with the specific structure that one can relax the constraints on the last variables of the collection `VARIABLES`. 
See also

common keyword: alldifferent, open_alldifferent.
size_max_seq_alldifferent (all different,disequality).
implies: atleast_nvalue.

Keywords

characteristic of a constraint: all different, disequality, hypergraph.
combinatorial object: sequence.
constraint arguments: pure functional dependency.
constraint type: sliding sequence constraint, open constraint, conditional constraint.
modelling: functional dependency.
### Arc input(s)
- VARIABLES

### Arc generator
- $PATH_1 \rightarrow \text{collection}$

### Arc arity
- *

### Arc constraint(s)
- `alldifferent(collection)`

### Graph property(ies)
- `NARC = SIZE`

#### Graph model
Note that this is an example where the arc constraints do not have the same arity. However they correspond to the same constraint.

Parts (A) and (B) of Figure 5.534 respectively show the initial and final graph associated with the Example slot.

![Graph Model](image)

(A)

![Graph Model](image)

(B)

Figure 5.534: Initial and final graph of the `size_max_starting_seq_alldifferent` constraint
### 5.313 sliding_card_skip0

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>N. Beldiceanu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>sliding_card_skip0(ATLEAST, ATMOST, VARIABLES, VALUES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>ATLEAST : int, ATMOST : int, VARIABLES : collection(var−dvar), VALUES : collection(val−int)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>ATLEAST ≥ 0, ATLEAST ≤</td>
<td>VARIABLES</td>
<td>, ATMOST ≥ 0, ATMOST ≤</td>
</tr>
</tbody>
</table>

Let \( n \) be the total number of variables of the collection VARIABLES. A **maximum non-zero set of consecutive variables** \( X_i..X_j (1 \leq i \leq j \leq n) \) is defined in the following way:

- All variables \( X_i, \ldots, X_j \) take a non-zero value,
- \( i = 1 \) or \( X_{i-1} \) is equal to 0,
- \( j = n \) or \( X_{j+1} \) is equal to 0.

Enforces that each maximum non-zero set of consecutive variables of the collection VARIABLES contains at least ATLEAST and at most ATMOST values from the collection of values VALUES.

**Example**

\[
\begin{pmatrix}
\text{var}_0, \\
\text{var}_7, \\
\text{var}_2, \\
\text{var}_9, \\
\text{var}_0, \\
\text{var}_9, \\
\text{var}_4, \\
\text{var}_9
\end{pmatrix}, \\
2, 3; \begin{pmatrix}
\text{var}_0, \\
\text{var}_9, \\
\text{var}_0, \\
\text{var}_9, \\
\text{var}_4, \\
\text{var}_9
\end{pmatrix}
\]

The sliding_card_skip0 constraint holds since the two maximum non-zero set of consecutive values 7 2 9 and 9 4 9 of its third argument \( \{0, 7, 2, 9, 0, 0, 9, 4, 9\} \) take both 2 \( (2 \in [\text{ATLEAST, ATMOST}] = [2, 3]) \) values within the set of values \( \{7, 9\} \).
Typical

\[ |\text{VARIABLES}| > 1 \]
\[ |\text{VALUES}| > 0 \]
\[ |\text{VARIABLES}| > |\text{VALUES}| \]
\[ \text{atleast}(1, \text{VARIABLES}, 0) \]
\[ \text{ATLEAST} > 0 \lor \text{ATMOST} < |\text{VARIABLES}| \]

Symmetries

- **ATLEAST** can be decreased to any value \( \geq 0 \).
- **ATMOST** can be increased to any value \( \leq |\text{VARIABLES}| \).
- Items of \text{VARIABLES} can be reversed.
- An occurrence of a value different from 0 of \text{VARIABLES}.var that belongs to \text{VALUES}.val (resp. does not belong to \text{VALUES}.val) can be replaced by any other value different from 0 in \text{VALUES}.val (resp. not in \text{VALUES}.val).

Usage

This constraint is useful in timetabling problems where the variables are interpreted as the type of job that a person does on consecutive days. Value 0 represents a rest day and one imposes a cardinality constraint on periods that are located between rest periods.

Remark

One cannot initially state a **global_cardinality** constraint since the rest days are not yet allocated. One can also not use an **among_seq** constraint since it does not hold for the sequences of consecutive variables that contains at least one rest day.

See also

- related: **among** (counting constraint on the full sequence), **global_cardinality** (counting constraint for different values on the full sequence).
- specialisation: **among_low_up** (maximal sequences replaced by the full sequence).

Keywords

- characteristic of a constraint: automaton, automaton with counters.
- combinatorial object: sequence.
- constraint network structure: alpha-acyclic constraint network(2).
- constraint type: timetabling constraint, sliding sequence constraint.
- miscellaneous: obscure.
Arc input(s) | VARIABLES
---|---
Arc generator | $PATH \rightarrow \text{collection}(\text{variables1, variables2})$
 | $LOOP \rightarrow \text{collection}(\text{variables1, variables2})$
Arc arity | 2
Arc constraint(s) | • $\text{variables1}.\text{var} \neq 0$
 | • $\text{variables2}.\text{var} \neq 0$
Sets | $\text{CC} \mapsto [\text{variables}]$
Constraint(s) on sets | $\text{among\_low\_up}(\text{ATLEAST, ATMOST, variables, VALUES})$

Graph model
Note that the arc constraint will produce the different sequences of consecutive variables that do not contain any 0. The CC set generator produces all the connected components of the final graph.

Parts (A) and (B) of Figure 5.535 respectively show the initial and final graph associated with the Example slot. Since we use the set generator CC we show the two connected components of the final graph. Since these two connected components both contains between 2 and 3 variables that take their values in $\{7, 9\}$ the sliding_card_skip0 constraint holds.

![Graph model](image)

**Figure 5.535:** Initial and final graph of the sliding_card_skip0 constraint
Automaton

Figure 5.536 depicts the automaton associated with the sliding_card_skip0 constraint. To each variable $\text{VAR}_i$ of the collection $\text{VARIABLES}$ corresponds a signature variable $S_i$. The following signature constraint links $\text{VAR}_i$ and $S_i$:

$(\text{VAR}_i = 0) \iff S_i = 0$ \quad \land \quad 

$(\text{VAR}_i \neq 0 \land \text{VAR}_i \notin \text{VALUES}) \iff S_i = 1 \quad \land \quad 

$(\text{VAR}_i \neq 0 \land \text{VAR}_i \in \text{VALUES}) \iff S_i = 2$.

Figure 5.536: Automaton of the sliding_card_skip0 constraint

Figure 5.537: Hypergraph of the reformulation corresponding to the automaton of the sliding_card_skip0 constraint
5.314 sliding_distribution

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[331]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>sliding_distribution(SEQ, VARIABLES, VALUES)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>SEQ : int, VARIABLES : collection(var−dvar), VALUES : collection(val−int, omin−int, omax−int)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>SEQ &gt; 0, SEQ ≤</td>
<td>VARIABLES</td>
</tr>
<tr>
<td>Purpose</td>
<td>For each sequence of SEQ consecutive variables of the VARIABLES collection, each value VALUES[i].val (1 ≤ i ≤</td>
<td>VALUES</td>
</tr>
</tbody>
</table>
| Example     | \[
\begin{pmatrix}
  \text{var} - 0, \\
  \text{var} - 5, \\
  \text{var} - 0, \\
  4, \text{var} - 6, \\
  \text{var} - 5, \\
  \text{var} - 0, \\
  \text{var} - 0
\end{pmatrix}
\] |       |

The sliding_distribution constraint holds since:

- On the first sequence of 4 consecutive values 0 5 0 6 values 0, 1, 4, 5 and 6 are respectively used 2, 0, 0, 1 and 1 times.
- On the second sequence of 4 consecutive values 5 0 6 5 values 0, 1, 4, 5 and 6 are respectively used 1, 0, 2 and 1 times.
- On the third sequence of 4 consecutive values 0 6 5 0 values 0, 1, 4, 5 and 6 are respectively used 2, 0, 0, 1 and 1 times.
• On the fourth sequence of 4 consecutive values 6 5 0 0 values 0, 1, 4, 5 and 6 are respectively used 2, 0, 0, 1 and 1 times.

Typical

SEQ > 1
SEQ < |VARIABLES|
|VARIABLES| > |VALUES|

Symmetries

• Items of VARIABLES can be reversed.
• An occurrence of a value of VARIABLES.var that does not belong to VALUES.val can be replaced by any other value that also does not belong to VALUES.val.
• Items of VALUES are permutible.
• VALUES.omin can be decreased to any value ≥ 0.
• VALUES.omax can be increased to any value ≤ SEQ.
• All occurrences of two distinct values in VARIABLES.var or VALUES.val can be swapped; all occurrences of a value in VARIABLES.var or VALUES.val can be renamed to any unused value.

Arg. properties

• Contractible wrt. VARIABLES when SEQ = 1.
• Prefix-contractible wrt. VARIABLES.
• Suffix-contractible wrt. VARIABLES.
• Contractible wrt. VALUES.

See also

common keyword: pattern, sliding_sum, stretch_circuit, stretch_path(sliding sequence constraint).

part of system of constraints: global_cardinality_low_up.
specialisation: among_seq(individual values replaced by single set of values).
used in graph description: global_cardinality_low_up.

Keywords

characteristic of a constraint: hypergraph.
combinatorial object: sequence.
constraint type: decomposition, sliding sequence constraint, system of constraints.
Arc input(s)  VARIABLES
Arc generator  PATH\rightarrow\text{collection}
Arc arity  SEQ
Arc constraint(s)  global\_cardinality\_low\_up(collection, VALUES)
Graph property(ies)  NARC = |\text{VARIABLES}| - SEQ + 1

Graph model  Note that the \text{sliding\_distribution} constraint is a constraint where the arc constraints do not have an arity of 2.

Parts (A) and (B) of Figure 5.538 respectively show the initial and final graph associated with the \textbf{Example} slot. Since all arc constraints hold (i.e., because of the graph property NARC = |\text{VARIABLES}| - SEQ + 1) the final graph corresponds to the initial graph.

Figure 5.538: Initial and final graph of the \text{sliding\_distribution} constraint
## 5.315 sliding_sum

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>CHIP</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>sliding_sum(LOW, UP, SEQ, VARIABLES)</td>
<td></td>
</tr>
<tr>
<td><strong>Synonym</strong></td>
<td>sequence.</td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOW : int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UP : int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEQ : int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARIABLES : collection(var−dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UP ≥ LOW</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEQ &gt; 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEQ ≤</td>
<td>VARIABLES</td>
<td></td>
</tr>
<tr>
<td>required(VARIABLES.var)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constrains all sequences of SEQ consecutive variables of the collection VARIABLES so that the sum of the variables belongs to interval [LOW, UP].</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Example

\[
\begin{pmatrix}
\text{var} - 1, \\
\text{var} - 4, \\
\text{var} - 2, \\
3, 7, 4, \\
\text{var} - 0, \\
\text{var} - 0, \\
\text{var} - 3, \\
\text{var} - 4
\end{pmatrix}
\]

The example considers all sliding sequences of SEQ = 4 consecutive values of \(\{1, 4, 2, 0, 0, 3, 4\}\) collection and constraints the sum to be in \([LOW, UP] = [3, 7]\). The sliding_sum constraint holds since the sum associated with the corresponding subsequences 1 4 2 0, 4 2 0 0, 2 0 0 3, and 0 0 3 4 are respectively 7, 6, 5 and 7.

| **Typical** | | |
| LOW ≥ 0 | | |
| UP > 0 | | |
| SEQ > 1 | | |
| SEQ < |VARIABLES| | |
| VARIABLES.var ≥ 0 | | |
| UP < |sum|VARIABLES.var| | |

### Symmetry

Items of VARIABLES can be reversed.

### Arg. properties

- **Contractible** wrt. VARIABLES when SEQ = 1.
- **Prefix-contractible** wrt. VARIABLES.
- **Suffix-contractible** wrt. VARIABLES.
Algorithm

Beldiceanu and Carlsson [29] have proposed a first incomplete filtering algorithm for the sliding_sum constraint. In 2008, Maher et al. showed in [254] that the sliding_sum constraint has a solution “if and only there are no negative cycles in the flow graph associated with the dual linear program” that encodes the conjunction of inequalities. They derive a bound-consistency filtering algorithm from this fact.

Systems

sliding_sum in MiniZinc.

See also

common keyword: sliding_distribution (sliding sequence constraint).
part of system of constraints: sum_ctr.
soft variant: relaxed_sliding_sum.
used in graph description: sum_ctr.

Keywords

characteristic of a constraint: hypergraph, sum.
combinatorial object: sequence.
constraint type: decomposition, sliding sequence constraint, system of constraints.
filtering: linear programming, flow, bound-consistency.
Arc input(s) | VARIABLES
---|---
Arc generator | PATH↦collection
Arc arity | SEQ
Arc constraint(s)
- sum_ctr(collection, ≥, LOW)
- sum_ctr(collection, ≤, UP)
Graph property(ies) | NARC = |VARIABLES| − SEQ + 1

Graph model
We use sum_ctr as an arc constraint. sum_ctr takes a collection of domain variables as its first argument.

Parts (A) and (B) of Figure 5.539 respectively show the initial and final graph associated with the Example slot. Since all arc constraints hold (i.e., because of the graph property NARC = |VARIABLES| − SEQ + 1) the final graph corresponds to the initial graph.

![Graph Model](image)

(A)

![Graph Model](image)

(B)

Figure 5.539: Initial and final graph of the sliding_sum constraint

Signature
Since we use the PATH arc generator with an arity of SEQ on the items of the VARIABLES collection, the expression |VARIABLES| − SEQ + 1 corresponds to the maximum number of arcs of the final graph. Therefore we can rewrite the graph property NARC = |VARIABLES| − SEQ + 1 to NARC ≥ |VARIABLES| − SEQ + 1 and simplify NARC to NARC.
5.3.16 sliding_time_window

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>N. Beldiceanu</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>sliding_time_window(WINDOW_SIZE, LIMIT, TASKS)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>WINDOW_SIZE : int</td>
<td>LIMIT : int</td>
</tr>
<tr>
<td>Restrictions</td>
<td>WINDOW_SIZE ≥ 0</td>
<td>LIMIT ≥ 0</td>
</tr>
</tbody>
</table>

**Purpose**

For any time window of size WINDOW_SIZE, the intersection of all the tasks of the collection TASKS with this time window is less than or equal to a given limit LIMIT.

**Example**

```
[9, 6, (origin - 10, duration - 3),
   (origin - 5, duration - 1),
   (origin - 6, duration - 2),
   (origin - 14, duration - 2),
   (origin - 2, duration - 2)]
```

The lower part of Figure 5.540 indicates the different tasks on the time axis. Each task is drawn as a rectangle with its corresponding identifier in the middle. Finally the upper part of Figure 5.540 shows the different time windows and the respective contribution of the tasks in these time windows. Note that we only need to focus on those time windows starting at the start of one of the tasks. A line with two arrows depicts each time window. The two arrows indicate the start and the end of the time window. At the left of each time window we give its occupation. Since this occupation is always less than or equal to the limit 6, the sliding_time_window constraint holds.

**Typical**

| WINDOW_SIZE ≥ 1 | LIMIT > 0 | LIMIT < sum(TASKS.duration) | | TASKS| > 1 | TASKS.duration > 0 |

**Symmetries**

- WINDOW_SIZE can be decreased.
- LIMIT can be increased.
- Items of TASKS are permutable.
- One and the same constant can be added to the origin attribute of all items of TASKS.
- TASKS.duration can be decreased to any value ≥ 0.
Arg. properties

Contractible wrt. TASKS.

Usage

The sliding time window constraint is useful for timetabling problems in order to put an upper limit on the total work over sliding time windows.

Reformulation

The sliding time window constraint can be expressed in terms of a set of |TASKS|^2 reified constraints and of |TASKS| linear inequalities constraints:

1. For each pair of tasks TASKS[i], TASKS[j] \((i, j \in [1, |TASKS|])\) of the TASKS collection we create a variable \(\text{Inter}_{ij}\) which is set to the intersection of TASKS[j] with the time window \(W_i\) of size WINDOW\_SIZE that starts at instant TASKS[i].origin:
   - If \(i = j\) (i.e., TASKS[i] and TASKS[j] coincide):
     \[\text{Inter}_{ij} = \min(\text{TASKS}[i].\text{duration}, \text{WINDOW\_SIZE}).\]
   - If \(i \neq j\) and TASKS[j].origin + TASKS[j].duration < TASKS[i].origin (i.e., TASKS[j] for sure ends before the time window \(W_i\)):
     \[\text{Inter}_{ij} = 0.\]
   - If \(i \neq j\) and TASKS[j].origin > TASKS[i].origin + WINDOW\_SIZE - 1 (i.e., TASKS[j] for sure starts after the time window \(W_i\)):
     \[\text{Inter}_{ij} = 0.\]
   - Otherwise (i.e., TASKS[j] can potentially overlap the time window \(W_i\)):
     \[\text{Inter}_{ij} = \max(0, \min(\text{TASKS}[i].\text{origin} + \text{WINDOW\_SIZE}.\text{TASKS}[j].\text{origin} + \text{TASKS}[j].\text{duration}) - \max(\text{TASKS}[i].\text{origin}, \text{TASKS}[j].\text{origin})).\]

2. For each task TASKS[i] \((i \in [1, |TASKS|])\) we create a linear inequality constraint \(\text{Inter}_{i1} + \text{Inter}_{i2} + \ldots + \text{Inter}_{i|\text{TASKS}|} \leq \text{LIMIT}\).

See also

common keyword: shift (temporal constraint).
related: sliding.time_window_sum (sum of intersections of tasks with sliding time window replaced by sum of the points of intersecting tasks with sliding time window).
used in graph description: sliding.time_window_from_start.

Keywords

constraint type: sliding sequence constraint, temporal constraint.

![Figure 5.540: Time windows of the sliding time window constraint](image-url)
Arc input(s) TASKS
Arc generator \( CLIQUE \rightarrow \text{collection}(\text{tasks1}, \text{tasks2}) \)
Arc arity 2
Arc constraint(s)

- \( \text{tasks1}\.\text{origin} \leq \text{tasks2}\.\text{origin} \)
- \( \text{tasks2}\.\text{origin} - \text{tasks1}\.\text{origin} < \text{WINDOW\_SIZE} \)

Sets \( \text{SUCC} \rightarrow [\text{source}, \text{tasks}] \)

Constraint(s) on sets \( \text{sliding\_time\_window\_from\_start}(\text{WINDOW\_SIZE}, \text{LIMIT}, \text{tasks}, \text{source}\_\text{origin}) \)

Graph model

We generate an arc from a task \( t_1 \) to a task \( t_2 \) if task \( t_2 \) does not start before task \( t_1 \) and if task \( t_2 \) intersects the time window that starts at the origin of task \( t_1 \). Each set generated by \( \text{SUCC} \) corresponds to all tasks that intersect in time the time window that starts at the origin of a given task.

Parts (A) and (B) of Figure 5.541 respectively show the initial and final graph associated with the Example slot. In the final graph, the successors of a given task \( t \) correspond to the set of tasks that do not start before task \( t \) and intersect the time window that starts at the origin of task \( t \).

![Figure 5.541: Initial and final graph of the sliding\_time\_window constraint](image-url)
5.317  **sliding_time_window_from_start**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Used for defining <code>sliding_time_window</code>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sliding_time_window_from_start(WINDOW_SIZE, LIMIT, TASKS, START)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>WINDOW_SIZE : int</code></td>
<td><code>LIMIT : int</code></td>
<td><code>TASKS : collection(origin-dvar, duration-dvar)</code></td>
</tr>
<tr>
<td><code>START : dvar</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>WINDOW_SIZE &gt; 0</code></td>
<td><code>LIMIT ≥ 0</code></td>
<td><code>required(TASKS,[origin,duration])</code></td>
</tr>
<tr>
<td><code>TASKS.duration ≥ 0</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The sum of the intersections of all the tasks of the TASKS collection with interval <code>[START, START + WINDOW_SIZE - 1]</code> is less than or equal to <code>LIMIT</code>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| \[
\begin{pmatrix}
9, 6, \langle \text{origin } - 10 \text{ duration } - 3, \\
\text{origin } - 5 \text{ duration } - 1, \\
\text{origin } - 6 \text{ duration } - 2 \rangle, 5
\end{pmatrix}
\] |       |       |
<p>| The intersections of tasks (\langle \text{id } - 1 \text{ origin } - 10 \text{ duration } - 3, \rangle, \langle \text{id } - 2 \text{ origin } - 5 \text{ duration } - 1, \rangle, \langle \text{id } - 3 \text{ origin } - 6 \text{ duration } - 2 \rangle) with interval ([\text{START}, \text{START } + \text{WINDOW_SIZE } - 1] = [5, 5 + 9 - 1] = [5, 13]) are respectively equal to 3, 1, and 2 (i.e., the three tasks of the TASKS collection are in fact included within interval [5, 13]). Consequently, the <code>sliding_time_window_from_start</code> constraint holds since the sum (3 + 1 + 2) of these intersections does not exceed the value of its second argument <code>LIMIT = 6</code>. |       |       |
| Typical    |       |       |
| <code>WINDOW_SIZE &gt; 1</code> | <code>LIMIT &gt; 0</code> | <code>LIMIT &lt; WINDOW_SIZE</code> |
| <code>|TASKS| &gt; 1</code> | <code>TASKS.duration &gt; 0</code> |       |
| Symmetries |       |       |
| - <code>WINDOW_SIZE</code> can be <strong>decreased</strong>. |       |       |
| - <code>LIMIT</code> can be <strong>increased</strong>. |       |       |
| - Items of <code>TASKS</code> are <strong>permutable</strong>. |       |       |
| - <code>TASKS.duration</code> can be <strong>decreased</strong> to any value (≥ 0). |       |       |
| - One and the same constant can be <strong>added</strong> to <code>START</code> as well as to the <code>origin</code> attribute of all items of <code>TASKS</code>. |       |       |</p>
<table>
<thead>
<tr>
<th>Argument properties</th>
<th>Contractible wrt. TASKS.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reformulation</td>
<td>Similar to the reformulation of sliding_time_window.</td>
</tr>
<tr>
<td>Used in</td>
<td>sliding_time_window.</td>
</tr>
<tr>
<td>Keywords</td>
<td>characteristic of a constraint: derived collection.</td>
</tr>
<tr>
<td></td>
<td>constraint type: sliding sequence constraint, temporal constraint.</td>
</tr>
</tbody>
</table>
Derived Collection
\[
\text{col}(S \rightarrow \text{collection}(\text{var} \rightarrow \text{dvar}), [\text{item} (\text{var} \rightarrow \text{START})])
\]

Arc input(s)
\[S \text{ TASKS}\]

Arc generator
\[\text{PRODUCT} \rightarrow \text{collection}(s, \text{tasks})\]

Arc arity
\[2\]

Arc constraint(s)
\[\text{TRUE}\]

Graph property(ies)
\[
\text{SUM\_WEIGHT\_ARC} \left( \max \left( 0, \min \left( \text{s.var + WINDOW\_SIZE}, \text{tasks.origin + tasks.duration} \right) \right) - \max (\text{s.var, tasks.origin}) \right) \leq \text{LIMIT}
\]

Graph model
Since we use the \text{TRUE} arc constraint the final and the initial graph are identical. The unique source of the final graph corresponds to the interval [\text{START}, \text{START} + \text{WINDOW\_SIZE} - 1]. Each sink of the final graph represents a given task of the \text{TASKS} collection. We associate to each arc the value given by the intersection of the task associated with one of the extremities of the arc with the time window [\text{START}, \text{START} + \text{WINDOW\_SIZE} - 1]. Finally, the graph property \text{SUM\_WEIGHT\_ARC} sums up all the valuations of the arcs and check that it does not exceed a given limit.

Parts (A) and (B) of Figure 5.542 respectively show the initial and final graph associated with the \text{Example} slot. To each arc of the final graph we associate the intersection of the corresponding sink task with interval [\text{START}, \text{START} + \text{WINDOW\_SIZE} - 1]. The constraint \text{sliding\_time\_window\_from\_start} holds since the sum of the previous intersections does not exceed \text{LIMIT}.

![Graph model](image)

Figure 5.542: Initial and final graph of the \text{sliding\_time\_window\_from\_start} constraint
5.318  sliding_time_window_sum

### Origin
Derived from `sliding_time_window`.

### Constraint
```
sliding_time_window_sum(WINDOW_SIZE, LIMIT, TASKS)
```

### Arguments
- `WINDOW_SIZE` : int
- `LIMIT` : int
- `TASKS` : collection((origin−dvar, end−dvar, npoint−dvar))

### Restrictions
```
WINDOW_SIZE > 0
LIMIT ≥ 0
required(TASKS,[origin, end, npoint])
TASKS.origin ≤ TASKS.end
TASKS.npoint ≥ 0
```

### Purpose
For any time window of size `WINDOW_SIZE`, the sum of the points of the tasks of the collection `TASKS` that overlap that time window do not exceed a given limit `LIMIT`.

### Example
```
\[
\begin{pmatrix}
\text{origin} - 10 & \text{end} - 13 & \text{npoint} - 2, \\
\text{origin} - 5 & \text{end} - 6 & \text{npoint} - 3, \\
9, 16, & \text{origin} - 6 & \text{end} - 8 & \text{npoint} - 4, \\
\text{origin} - 14 & \text{end} - 16 & \text{npoint} - 5, \\
\text{origin} - 2 & \text{end} - 4 & \text{npoint} - 6
\end{pmatrix}
\]
```
The lower part of Figure 5.543 indicates the different tasks on the time axis. Each task is drawn as a rectangle with its corresponding identifier in the middle. Finally the upper part of Figure 5.543 shows the different time windows and the respective contribution of the tasks in these time windows. A line with two arrows depicts each time window. The two arrows indicate the start and the end of the time window. At the right of each time window we give its occupation. Since this occupation is always less than or equal to the limit 16, the `sliding_time_window_sum` constraint holds.

### Typical
```
WINDOW_SIZE > 1
LIMIT > 0
LIMIT < sum(TASKS.npoint)
|TASKS| > 1
TASKS.origin < TASKS.end
TASKS.npoint > 0
```

### Symmetries
- `WINDOW_SIZE` can be decreased.
- `LIMIT` can be increased.
- Items of `TASKS` are permutable.
- `TASKS.npoint` can be decreased to any value ≥ 0.
- One and the same constant can be added to the `origin` and `end` attributes of all items of `TASKS`.

### Graph
- DESCRIPTION
- LINKS
- GRAPHS
Arg. properties  

Contractible wrt. TASKS.

Usage  

This constraint may be used for timetabling problems in order to put an upper limit on the cumulated number of points in a shift.

Reformulation  

The sliding time window sum constraint can be expressed in term of a set of $|\text{TASKS}|^2$ reified constraints and of $|\text{TASKS}|$ linear inequalities constraints:

1. For each pair of tasks $\text{TASKS}[i], \text{TASKS}[j]$ ($i, j \in [1, |\text{TASKS}|]$) of the TASKS collection we create a variable $\text{Point}_{ij}$ which is set to $\text{TASKS}[j].\text{npoint}$ if $\text{TASKS}[j]$ intersects the time window $W_i$ of size $\text{WINDOW}\_\text{SIZE}$ that starts at instant $\text{TASKS}[i].\text{origin}$, or 0 otherwise:
   - If $i = j$ (i.e., $\text{TASKS}[i]$ and $\text{TASKS}[j]$ coincide):
     - $\text{Point}_{ij} = \text{TASKS}[i].\text{npoint}$.
   - If $i \neq j$ and $\text{TASKS}[j].\text{end} < \text{TASKS}[i].\text{origin}$ (i.e., $\text{TASKS}[j]$ for sure ends before the time window $W_i$):
     - $\text{Point}_{ij} = 0$.
   - If $i \neq j$ and $\text{TASKS}[j].\text{origin} > \text{TASKS}[i].\text{origin} + \text{WINDOW}\_\text{SIZE} - 1$ (i.e., $\text{TASKS}[j]$ for sure starts after the time window $W_i$):
     - $\text{Point}_{ij} = 0$.
   - Otherwise (i.e., $\text{TASKS}[j]$ can potentially overlap the time window $W_i$):
     - $\text{Point}_{ij} = \min(1, \max(0, \min(\text{TASKS}[i].\text{origin} + \text{WINDOW}\_\text{SIZE}, \text{TASKS}[j].\text{end}) - \max(\text{TASKS}[i].\text{origin}, \text{TASKS}[j].\text{origin})) - \text{TASKS}[j].\text{npoint})$.

2. For each task $\text{TASKS}[i]$ ($i \in [1, |\text{TASKS}|]$) we create a linear inequality constraint $\text{Point}_{i1} + \text{Point}_{i2} + \ldots + \text{Point}_{i|\text{TASKS}|} \leq \text{LIMIT}$.

See also  

related: sliding time window (sum of the points of intersecting tasks with sliding time window replaced by sum of intersections of tasks with sliding time window).

used in graph description: sum.ctr.

Keywords  

characteristic of a constraint: time window, sum.

constraint type: sliding sequence constraint, temporal constraint.
Figure 5.543: Time windows of the `sliding_time_window_sum` constraint
Arc input(s)  TASKS
Arc generator \( \text{SELF} \rightarrow \text{collection}(\text{tasks}) \)
Arc arity 1
Arc constraint(s)  \( \text{tasks}.\text{origin} \leq \text{tasks}.\text{end} \)
Graph property(ies)  \( \text{NARC} = |\text{TASKS}| \)

Arc input(s)  TASKS
Arc generator \( \text{CLIQUE} \rightarrow \text{collection}(\text{tasks1}, \text{tasks2}) \)
Arc arity 2
Arc constraint(s)  
  - \( \text{tasks1}.\text{end} \leq \text{tasks2}.\text{end} \)
  - \( \text{tasks2}.\text{origin} - \text{tasks1}.\text{end} < \text{WINDOW\_SIZE} - 1 \)
Sets  \( \text{SUCC} \rightarrow \left[ \begin{array}{c}
\text{source}, \\
\text{variables} - \text{col}(\begin{array}{c}
\text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \\
\text{item}(\text{var} - \text{TASKS}.\text{npoint})
\end{array})
\end{array} \right] \)
Constraint(s) on sets  \( \text{sum\_ctr}(\text{variables}, \leq, \text{LIMIT}) \)

Graph model  

We generate an arc from a task \( t_1 \) to a task \( t_2 \) if task \( t_2 \) does not end before the end of task \( t_1 \) and if task \( t_2 \) intersects the time window that starts at the last instant of task \( t_1 \). Each set generated by \( \text{SUCC} \) corresponds to all tasks that intersect in time the time window that starts at instant \( \text{end} - 1 \), where \( \text{end} \) is the end of a given task.

Parts (A) and (B) of Figure 5.544 respectively show the initial and final graph associated with the Example slot. In the final graph, the successors of a given task \( t \) correspond to the set of tasks that both do not end before the end of task \( t \), and intersect the time window that starts at the end of task \( t \).

Signature  

Consider the first graph constraint. Since we use the \( \text{SELF} \) arc generator on the \( \text{TASKS} \) collection the maximum number of arcs of the final graph is equal to \( |\text{TASKS}| \). Therefore we can rewrite \( \text{NARC} = |\text{TASKS}| \) to \( \text{NARC} \geq |\text{TASKS}| \) and simplify \( \text{NARC} \) to \( \text{NARC} \).
Figure 5.544: Initial and final graph of the sliding_time_window_sum constraint
5.319 smooth

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived from change.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Constraint**

\[ \text{smooth}(\text{NCHANGE}, \text{TOLERANCE}, \text{VARIABLES}) \]

**Arguments**

- \text{NCHANGE} : dvar
- \text{TOLERANCE} : int
- \text{VARIABLES} : collection(var - dvar)

**Restrictions**

- \text{NCHANGE} \geq 0
- \text{NCHANGE} < |\text{VARIABLES}|
- \text{TOLERANCE} \geq 0
- \text{required}(\text{VARIABLES}, \text{var})

**Purpose**

\text{NCHANGE} is the number of times that |X - Y| > \text{TOLERANCE} holds; \( X \) and \( Y \) correspond to consecutive variables of the collection \text{VARIABLES}.

**Example**

\( (1, 2, (1, 3, 4, 5, 2)) \)

In the example we have one change between values 5 and 2 since the difference in absolute value is greater than the tolerance (i.e., |5 - 2| > 2). Consequently the \text{NCHANGE} argument is fixed to 1 and the smooth constraint holds.

**Typical**

- \text{NCHANGE} > 0
- \text{TOLERANCE} > 0
- |\text{VARIABLES}| > 2
- \text{range}(\text{VARIABLES}.\text{var}) > 1

**Symmetries**

- Items of \text{VARIABLES} can be reversed.
- One and the same constant can be added to the \text{var} attribute of all items of \text{VARIABLES}.

**Arg. properties**

- Functional dependency: \text{NCHANGE} determined by \text{TOLERANCE} and \text{VARIABLES}.
- Prefix-contractible wrt. \text{VARIABLES} when \text{NCHANGE} = 0.
- Suffix-contractible wrt. \text{VARIABLES} when \text{NCHANGE} = 0.
- Prefix-contractible wrt. \text{VARIABLES} when \text{NCHANGE} = |\text{VARIABLES}| - 1.
- Suffix-contractible wrt. \text{VARIABLES} when \text{NCHANGE} = |\text{VARIABLES}| - 1.

**Usage**

This constraint is useful for the following problems:

- Assume that \text{VARIABLES} corresponds to the number of people that work on consecutive weeks. One may not normally increase or decrease too drastically the number of people from one week to the next week. With the smooth constraint you can state a limit on the number of drastic changes.
Assume you have to produce a set of orders, each order having a specific attribute.
You want to generate the orders in such a way that there is not a too big difference
between the values of the attributes of two consecutive orders. If you can’t achieve
this on two given specific orders, this would imply a set-up or a cost. Again, with the
smooth constraint, you can control this kind of drastic changes.

Algorithm
A first incomplete algorithm is described in [29]. The sketch of a filtering algorithm for the
conjunction of the smooth and the stretch constraints based on dynamic programming
achieving arc-consistency is mentioned by Lars Hellsten in [191, page 60].

Reformulation
The smooth constraint can be reformulated with the seq_bin constraint [290] that we now
introduce. Given \( N \) a domain variable, \( X \) a sequence of domain variables, and \( C \) and \( B \) two
binary constraints, seq_bin(\( N, X, C, B \)) holds if (1) \( N \) is equal to the number of \( C \)-stretches
in the sequence \( X \), and (2) \( B \) holds on any pair of consecutive variables in \( X \). A \( C \)-stretch
is a generalisation of the notion of stretch introduced by G. Pesant [285], where the equal-
ity constraint is made explicit by replacing it by a binary constraint \( C \), i.e., a \( C \)-stretch
is a maximal length subsequence of \( X \) for which the binary constraint \( C \) is satisfied on
consecutive variables. smooth(\( \text{NCHANGE, VARIABLES, TOLERANCE} \)) can be reformulated
as \( N = N1 - 1 \land \text{seq_bin}(N1, X, |x_i - x_{i+1}| \leq \text{TOLERANCE}, \text{true}) \), where true is the
universal constraint.

See also
common keyword: change (number of changes in a sequence with respect to a binary constraint).
related: distance.

Keywords
characteristic of a constraint: automaton, automaton with counters, non-deterministic automaton, non-deterministic automaton.
constraint arguments: pure functional dependency.
constraint network structure: sliding cyclic(1) constraint network(2), Berge-acyclic constraint network.
constraint type: timetabling constraint.
filtering: dynamic programming.
modelling: number of changes, functional dependency.
modelling exercises: n-Amazon.
puzzles: n-Amazon.
Arc input(s) | VARIABLES
---|---
Arc generator | $PATH \rightarrow \text{collection}(\text{variables1}, \text{variables2})$
Arc arity | 2
Arc constraint(s) | $\text{abs}(\text{variables1.var} - \text{variables2.var}) > \text{TOLERANCE}$
Graph property(ies) | NARC = NCHANGE

Graph model

Parts (A) and (B) of Figure 5.545 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the unique arc of the final graph is stressed in bold.

![Graph model](image)

Figure 5.545: Initial and final graph of the smooth constraint
Automaton

Figure 5.546 depicts a first automaton that only accepts all the solutions of the smooth constraint. This automaton uses a counter in order to record the number of satisfied constraints of the form \((|\text{VAR}_i - \text{VAR}_{i+1}| > \text{TOLERANCE})\) already encountered. To each pair of consecutive variables \((\text{VAR}_i, \text{VAR}_{i+1})\) of the collection VARIABLES corresponds a 0-1 signature variable \(S_i\). The following signature constraint links \(\text{VAR}_i, \text{VAR}_{i+1}\) and \(S_i\):

\[
(|\text{VAR}_i - \text{VAR}_{i+1}| > \text{TOLERANCE}) \Leftrightarrow S_i = 1.
\]

\(\{C=0\}\)

\(|\text{VAR}_1 - \text{VAR}_{1+1}| > \text{TOLERANCE}, S_1 = 1 \quad \text{NCHANGE}=C \quad |\text{VAR}_1 - \text{VAR}_{1+1}| <= \text{TOLERANCE}\)

\(\{C=C+1\}\)

Figure 5.546: Automaton (with a counter) of the smooth constraint

Since the reformulation associated with the previous automaton is not Berge-acyclic, we now describe a second counter free automaton that also only accepts all the solutions of the smooth constraint. Without loss of generality, assume that the collection of variables VARIABLES contains at least two variables (i.e., \(|\text{VARIABLES}| \geq 2\)). Let \(n, \text{min}, \text{max},\) and \(D\) respectively denote the number of variables of the collection VARIABLES, the smallest value that can be assigned to the variables of VARIABLES, the largest value that can be assigned to the variables of VARIABLES, and the union of the domains of the variables of VARIABLES. Clearly, the maximum number of changes (i.e., the number of times the constraint \((|\text{VAR}_i - \text{VAR}_{i+1}| > \text{TOLERANCE})\) (1 \(\leq i < n\)) holds) cannot exceed the quantity \(m = \min(n-1, \text{NCHANGE})\). The \((m+1) \cdot |D| + 2\) states of the automaton that only accepts all the solutions of the smooth constraint are defined in the following way:

- We have an initial state labelled by \(s_I\).
- We have \(m \cdot |D|\) intermediate states labelled by \(s_{ij}\) (\(i \in D, j \in [0, m]\)). The first subscript \(i\) of state \(s_{ij}\) corresponds to the value currently encountered. The second subscript \(j\) denotes the number of already encountered satisfied constraints of the form \((|\text{VAR}_k - \text{VAR}_{k+1}| > \text{TOLERANCE})\) from the initial state \(s_I\) to the state \(s_{ij}\).
- We have a final state labelled by \(s_F\).

Four classes of transitions are respectively defined in the following way:
1. There is a transition, labelled by $i$, from the initial state $s_I$ to the state $s_{i0}$, ($i \in \mathcal{D}$).

2. There is a transition, labelled by $j$, from every state $s_{ij}$, ($i \in \mathcal{D}, j \in [0,m]$), to the final state $s_F$.

3. $\forall i \in \mathcal{D}, \forall j \in [0,m], \forall k \in \mathcal{D} \cap [\max(min, i - \text{TOLERANCE}), \min(max, i + \text{TOLERANCE})]$ there is a transition labelled by $k$ from $s_{ij}$ to $s_{kj}$ (i.e., the counter $j$ does not change for values $k$ that are too closed from value $i$).

4. $\forall i \in \mathcal{D}, \forall j \in [0, m - 1], \forall k \in \mathcal{D} \setminus [\max(min, i - \text{TOLERANCE}), \min(max, i + \text{TOLERANCE})]$ there is a transition labelled by $k$ from $s_{ij}$ to $s_{kj+1}$ (i.e., the counter $j$ is incremented by $+1$ for values $k$ that are too far from $i$).

We have $|\mathcal{D}|$ transitions of type 1, $|\mathcal{D}| \cdot (m + 1)$ transitions of type 2, and at least $|\mathcal{D}|^2 \cdot m$ transitions of types 3 and 4. Since the maximum value of $m$ is equal to $n - 1$, in the worst case we have at least $|\mathcal{D}|^2 \cdot (n - 1)$ transitions. This leads to a worst case time complexity of $O(|\mathcal{D}|^2 \cdot n^2)$ if we use Pesant’s algorithm for filtering the regular constraint [286].

Figure 5.548 depicts the corresponding counter free non deterministic automaton associated with the smooth constraint under the hypothesis that (1) all variables of VARIABLES are assigned a value in $\{0, 1, 2, 3\}$, (2) $|\text{VARIABLES}|$ is equal to 4, and (3) TOLERANCE is equal to 1.
The sequence of variables
\( \text{VAR}_1, \text{VAR}_2, \text{VAR}_3, \text{VAR}_4, \text{NCHANGE} \)

is passed to the automaton.

Figure 5.548: Counter free non deterministic automaton of the smooth(\text{NCHANGE}, 1, (\text{VAR}_1, \text{VAR}_2, \text{VAR}_3, \text{VAR}_4)) constraint assuming \( \text{VAR}_i \in [0, 3] \)
\((1 \leq i \leq 3)\), with initial state \( s_I \) and final state \( s_F \).
5.320  soft_all_equal_max_var

**DESCRIPTION**

**Constraint**

soft_all_equal_max_var(N, VARIABLES)

**Arguments**

N : dvar
VARIABLES : collection(var\--dvar)

**Restrictions**

N ≥ 0
N ≤ |VARIABLES|
required(VARIABLES, var)

**Purpose**

Let $M$ be the number of occurrences of the most often assigned value to the variables of the VARIABLES collection. $N$ is less than or equal to the total number of variables of the VARIABLES collection minus $M$ (i.e., $N$ is less than or equal to the minimum number of variables that need to be reassigned in order to obtain a solution where all variables are assigned a same value).

**Example**

(1, ⟨5, 1, 5, 5⟩)

Within the collection ⟨5, 1, 5, 5⟩, 3 is the number of occurrences of the most assigned value. Consequently, the soft_all_equal_max_var constraint holds since the argument $N = 1$ is less than or equal to the total number of variables 4 minus 3.

**Typical**

$N > 0$
$N < |VARIABLES|$
$|VARIABLES| > 1$

**Symmetries**

- $N$ can be decreased to any value $\geq 0$.
- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

**Algorithm**

[137].

**See also**

common keyword:  soft_all_equal_min_ctr,  soft_all_equal_min_var,
soft_alldifferentCtr, soft_alldifferent_var (soft constraint).
	hard version: all_equal.

related: atmost_nvalue.

**Keywords**

constraint type:  soft constraint,  value constraint,  relaxation,
variable-based violation measure.

filtering: arc-consistency, bound-consistency.
Graph model

We generate an initial graph with binary equalities constraints between each vertex and its successors. The graph property states that \( N \) is less than or equal to the difference between the total number of vertices of the initial graph and the number of vertices of the largest strongly connected component of the final graph.

Parts (A) and (B) of Figure 5.549 respectively show the initial and final graph associated with the Example slot. Since we use the MAX_NS CC graph property we show one of the largest strongly connected component of the final graph.

Figure 5.549: Initial and final graph of the soft_all_equal_max_var constraint
5.321 soft_all_equal_min_ctr

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[190]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>soft_all_equal_min_ctr(N,VARIABLES)</td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>soft_alldiff_max_ctr, soft_alldifferent_max_ctr, soft_alldistinct_max_ctr</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>N : int</td>
<td>VARIABLES : collection(var−dvar)</td>
</tr>
<tr>
<td>Restrictions</td>
<td>N ≥ 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N ≤</td>
<td>VARIABLES</td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES, var)</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>Consider the equality constraints involving two distinct variables of the collection VARIABLES. Among the previous set of constraints, N is less than or equal to the number of equality constraints that hold.</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>(6, ⟨5, 1, 5, 5⟩)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Within the collection ⟨5, 1, 5, 5⟩ six equality constraints hold. Consequently, the soft_all_equal_ctr constraint holds since the argument N = 6 is less than or equal to the number of equality constraints that hold.</td>
<td></td>
</tr>
<tr>
<td>Typical</td>
<td>N &gt; 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N &lt;</td>
<td>VARIABLES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VARIABLES</td>
</tr>
<tr>
<td>Symmetries</td>
<td>• N can be decreased to any value ≥ 0.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Items of VARIABLES are permutable.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.</td>
<td></td>
</tr>
<tr>
<td>Remark</td>
<td>It was shown in [190] that, finding out whether the soft_all_equal_ctr constraint has a solution or not is NP-hard. This was achieved by reduction from 3-dimensional-matching. Hebrard et al. also identify a tractable class when no value occurs in more than two variables of the collection VARIABLES that is equivalent to the vertex matching problem. One year later, [137] shows how to achieve bound-consistency in polynomial time.</td>
<td></td>
</tr>
<tr>
<td>See also</td>
<td>common keyword: soft_all_equal_max_var, soft_all_equal_min_var, soft_alldifferent_ctr, soft_alldifferent_var (soft constraint).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hard version: all_equal.</td>
<td></td>
</tr>
<tr>
<td>related:</td>
<td>atmost_nvalue.</td>
<td></td>
</tr>
</tbody>
</table>
Keywords

complexity: 3-dimensional-matching.
constraint type: soft constraint, value constraint, relaxation,
decomposition-based violation measure.
filtering: bound-consistency.
Graph model

We generate an initial graph with binary *equalities* constraints between each vertex and its successors. We use the arc generator *CLIQUE(≠)* in order to avoid considering *equality* constraints between the same variable. The graph property states that $N$ is less than or equal to the number of *equalities* that hold in the final graph.

Parts (A) and (B) of Figure 5.550 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. Six equality constraints remain in the final graph.

![Initial and final graph](image-url)
### 5.322 soft_all_equal_min_var

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[137]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>soft_all_equal_min_var(N, VARIABLES)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>N : dvar</td>
<td>VARIABLES : collection(var–dvar)</td>
</tr>
<tr>
<td>Restrictions</td>
<td>N ≥ 0</td>
<td>required(VARIABLES, var)</td>
</tr>
</tbody>
</table>

**Purpose**

Let $M$ be the number of occurrences of the most often assigned value to the variables of the VARIABLES collection. $N$ is greater than or equal to the total number of variables of the VARIABLES collection minus $M$ (i.e., $N$ is greater than or equal to the minimum number of variables that need to be reassigned in order to obtain a solution where all variables are assigned a same value).

**Example**

\[(1, (5, 1, 5, 5))\]

Within the collection \((5, 1, 5, 5)\), 3 is the number of occurrences of the most assigned value. Consequently, the soft_all_equal_min_var constraint holds since the argument $N = 1$ is greater than or equal to the total number of variables 4 minus 3.

**Typical**

\[N > 0 \quad |\text{VARIABLES}| > 1\]

**Symmetries**

- $N$ can be increased.
- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

**Algorithm**

Let $m$ denote the total number of potential values that can be assigned to the variables of the VARIABLES collection. In [137], E. Hebrard et al. provides an $O(m)$ filtering algorithm achieving arc-consistency on the soft_all_equal_min_var constraint. The same paper also provides an algorithm with a lower complexity for achieving range consistency. Both algorithms are based on the following ideas:

- In a first phase, they both compute an envelope of the union $\mathcal{D}$ of the domains of the variables of the VARIABLES collection, i.e., an array $A$ that indicates for each potential value $v$ of $\mathcal{D}$, the maximum number of variables that could possibly be assigned value $v$. Let $max_{occ}$ denote the maximum value over the entries of array $A$, and let $\mathcal{V}_{max_{occ}}$ denote the set of values which all occur in $max_{occ}$ variables of the VARIABLES collection. The quantity $|\text{VARIABLES}| - max_{occ}$ is a lower bound of $N$. 
In a second phase, depending on the relative ordering between \( \text{max \_occ} \) and the minimum value of \(| \text{VARIABLES} | - \overline{N} \), i.e., \(| \text{VARIABLES} | - \overline{N}, \) we have the three following cases:

1. When \( \text{max \_occ} < | \text{VARIABLES} | - \overline{N} \), the constraint \( \text{soft \_all \_equal \_min \_var} \) simply fails since not enough variables of the \( \text{VARIABLES} \) collection can be assigned the same value.

2. When \( \text{max \_occ} = | \text{VARIABLES} | - \overline{N} \), the constraint \( \text{soft \_all \_equal \_min \_var} \) can be satisfied. In this context, a value \( v \) can be removed from the domain of a variable \( V \) of the \( \text{VARIABLES} \) collection if and only if:
   (a) value \( v \) does not belong to \( V_{\text{max \_occ}} \),
   (b) the domain of variable \( V \) contains all values of \( V_{\text{max \_occ}} \).

On the one hand, the first condition can be understood as the fact that value \( v \) is not a value that allows to have at least \( | \text{VARIABLES} | - \overline{N} \) variables assigned the same value. On the other hand, the second condition can be interpreted as the fact that variable \( V \) is absolutely required in order to have at least \( | \text{VARIABLES} | - \overline{N} \) variables assigned the same value.

3. When \( \text{max \_occ} > | \text{VARIABLES} | - \overline{N} \), the constraint \( \text{soft \_all \_equal \_min \_var} \) can be satisfied, but no value can be pruned.

Note that, in the context of range consistency, the first phase of the filtering algorithm can be interpreted as a sweep algorithm were:

- On the one hand, the sweep status corresponds to the maximum number of occurrence of variables that can be assigned a given value.
- On the other hand, the event point series correspond to the minimum values of the variables of the \( \text{VARIABLES} \) collection as well as to the maximum values (+1) of the same variables.

Figure 5.551 illustrates the previous filtering algorithm on an example where \( \overline{N} \) is equal to 1, and where we have four variables \( V_1, V_2, V_3 \) and \( V_4 \) respectively taking their values within intervals \([1, 3], [3, 7], [0, 8] \) and \([5, 6] \) (see Part (A) of Figure 5.551, where the values of each variable are assigned the same colour that we retrieve in the other parts of Figure 5.551).

Part (B) of Figure 5.551 illustrates the first phase of the filtering algorithm, namely the computation of the envelope of the domains of variables \( V_1, V_2, V_3 \) and \( V_4 \). The start events \( s_1, s_2, s_3, s_4 \) (i.e., the events respectively associated with the minimum value of variables \( V_1, V_2, V_3, V_4 \)) where the envelope is increased by 1 are represented by the character ↑. Similarly, the end events (i.e., the events \( e_1, e_2, e_3, e_4 \) respectively associated with the maximum value (+1) of \( V_1, V_2, V_3, V_4 \) are represented by the character ↓). Since the highest peak of the envelope is equal to 3 we have that \( \text{max \_occ} \) is equal to 3. The values that allow to reach this highest peak are equal to \( V_{\text{max \_occ}} = \{3, 5, 6\} \) (i.e., shown in red in Part (B) of Figure 5.551).

Finally, Part (C) of Figure 5.551 illustrates the second phase of the filtering algorithm. Since \( \text{max \_occ} = 3 \) is equal to \( | \text{VARIABLES} | - \overline{N} = 4 - 1 \) we remove from the variables whose domains contain \( V_{\text{max \_occ}} = \{3, 5, 6\} \) (i.e., variables \( V_2 \) and \( V_4 \)) all values not in \( V_{\text{max \_occ}} = \{3, 5, 6\} \) (i.e., values 4, 7 for variable \( V_2 \) and values 0, 1, 2, 4, 7, 8 for variable \( V_3 \)).

See also common keyword: \( \text{soft \_all \_equal \_max \_var}, \text{soft \_all \_equal \_min \_ctr}, \text{soft \_alldifferent \_ctr}, \text{soft \_alldifferent \_var}(\text{soft \_constraint}) \).
hard version: all_equal.
related: atmost_nvalue.

Keywords

constraint type: soft constraint, value constraint, relaxation,
variable-based violation measure.
filtering: arc-consistency, sweep.
Figure 5.551: Illustration of the two phases filtering algorithm
Arc input(s) | VARIABLES  
---|---
Arc generator | CLIQUE ↦ \(\text{collection}(\text{variables}_1, \text{variables}_2)\)
Arc arity | 2
Arc constraint(s) | \(\text{variables}_1.\text{var} = \text{variables}_2.\text{var}\)
Graph property(ies) | \(\text{MAX_NSCC} \geq |\text{VARIABLES}| - N\)

**Graph model**

We generate an initial graph with binary *equalities* constraints between each vertex and its successors. The graph property states that \(N\) is greater than or equal to the difference between the total number of vertices of the initial graph and the number of vertices of the largest strongly connected component of the final graph.

Parts (A) and (B) of Figure 5.552 respectively show the initial and final graph associated with the Example slot. Since we use the MAX_NSCC graph property we show one of the largest strongly connected component of the final graph.

![Figure 5.552: Initial and final graph of the soft_all_equal_min_var constraint](image)

Figure 5.552: Initial and final graph of the soft_all_equal_min_var constraint
5.323  soft_alldifferent_ctr

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[294]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>soft_alldifferent_ctr(C, VARIABLES)</td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>soft_alldiff_ctr, soft_alldistinct_ctr, soft_alldiff_min_ctr, soft_alldifferent_min_ctr, soft_alldistinct_min_ctr, soft_all_equal_max_ctr.</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>C : dvar</td>
<td>VARIABLES : collection(var−dvar)</td>
</tr>
<tr>
<td>Restrictions</td>
<td>C ≥ 0 required(VARIABLES, var)</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>Consider the disequality constraints involving two distinct variables VARIABLES[i].var and VARIABLES[j].var (i &lt; j) of the collection VARIABLES. Among the previous set of constraints, C is greater than or equal to the number of disequality constraints that do not hold.</td>
<td></td>
</tr>
</tbody>
</table>
| Example     | \[
\begin{pmatrix}
 4, \\
  \text{var} - 5, \\
  \text{var} - 1, \\
  \text{var} - 9, \\
  \text{var} - 1, \\
  \text{var} - 5, \\
  \text{var} - 5
\end{pmatrix}
\] |
| Typical     | C > 0 [VARIABLES] > 1 |        |
| Symmetries  | • C can be increased. |
|             | • Items of VARIABLES are permutable. |
|             | • All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value. |
| Usage       | A soft alldifferent constraint. |
| Remark      | The soft_alldifferent_ctr constraint is called soft_alldiff_min_ctr or soft_all_equal_max_ctr in [137]. |
Algorithm

Since it focuses on the soft aspect of the \texttt{alldifferent} constraint, the original article \cite{394} that introduces this constraint describes how to evaluate the minimum value of $C$ and how to prune according to the maximum value of $C$. The corresponding filtering algorithm does not achieve \textit{arc-consistency}. W.-J. van Hoeve \cite{397} presents a new filtering algorithm that achieves \textit{arc-consistency}. This algorithm is based on a reformulation into a \textit{minimum-cost flow} problem.

See also

- \texttt{soft\_all\_equal\_max\_var}, \texttt{soft\_all\_equal\_min\_ctr}, \texttt{soft\_all\_equal\_min\_var}, \texttt{soft\_alldifferent\_var (soft\_constraint)}
- \texttt{hard\_version: alldifferent}.
- \texttt{related: atmost\_nvalue}.

Keywords

- \texttt{characteristic of a constraint: all different, disequality}.
- \texttt{constraint type: soft constraint, value constraint, relaxation, decomposition-based violation measure}.
- \texttt{filtering: minimum cost flow}.
- \texttt{modelling: degree of diversity of a set of solutions}.
- \texttt{modelling exercises: degree of diversity of a set of solutions}.
Arc input(s) | VARIABLES
--- | ---
Arc generator | CLIQUE(<) \( \mapsto \) collection(variables1, variables2)
Arc arity | 2
Arc constraint(s) | variables1.var = variables2.var
Graph property(ies) | NARC \( \leq \) C

Graph model

We generate an initial graph with binary equalities constraints between each vertex and its successors. We use the arc generator CLIQUE(<) in order to avoid counting twice the same equality constraint. The graph property states that C is greater than or equal to the number of equalities that hold in the final graph.

Parts (A) and (B) of Figure 5.553 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold. Since four equality constraints remain in the final graph the cost variable C is greater than or equal to 4.

Figure 5.553: Initial and final graph of the soft_alldifferent_ctr constraint
5.324  soft_alldifferent_var

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
</table>

**Origin**

[294]

**Constraint**

soft_alldifferent_var(C, VARIABLES)

**Synonyms**

soft_alld iff_var, soft_all distinct_var, soft_all diff_min_var, soft_all different_min_var, soft_all distinct_min_var.

**Arguments**

C : dvar
VARIABLES : collection(var−dvar)

**Restrictions**

\[ C \geq 0 \]
\[ \text{required}(\text{VARIABLES}, \text{var}) \]

**Purpose**

C is greater than or equal to the minimum number of variables of the collection VARIABLES for which the value needs to be changed in order that all variables of VARIABLES take a distinct value.

**Example**

\[
\begin{pmatrix}
3, \\
\text{var} - 5, \\
\text{var} - 1, \\
\text{var} - 9, \\
\text{var} - 1, \\
\text{var} - 5, \\
\text{var} - 5
\end{pmatrix}
\]

Within the collection \( (5, 1, 9, 1, 5, 5) \), 3 and 2 items are respectively fixed to values 5 and 1. Therefore one must change the values of at least \((3 - 1) + (2 - 1) = 3\) items to get back to 6 distinct values. Consequently, the soft_alldifferent_var constraint holds since its first argument \( C \) is greater than or equal to 3.

**Typical**

\[
C > 0 \\
2 * C \leq |\text{VARIABLES}| \\
|\text{VARIABLES}| > 1
\]

**Symmetries**

- C can be increased.
- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

**Usage**

A soft alldifferent constraint.

**Remark**

Since it focus on the soft aspect of the alldifferent constraint, the original article [294], which introduce this constraint, describes how to evaluate the minimum value of \( C \) and how to prune according to the maximum value of \( C \).

The soft_alldifferent_var constraint is called soft_alldiff_min_var in [137].
Algorithm The filtering algorithm presented in [294] achieves arc-consistency.

Reformulation By introducing a variable $M$ that gives the number of distinct values used by variables of the collection VARIABLES, the soft_alldifferent_var($C, VARIABLES$) constraint can be expressed as a conjunction of the nvalue($M, VARIABLES$) constraint and of the linear constraint $C \geq |VARIABLES| - M$.

See also common keyword: soft_all_equal_max_var, soft_all_equal_min_ctr, soft_all_equal_min_var, soft_alldifferent_ctr, weighted_partial_alldiff (soft constraint).

hard version: alldifferent.
related: atmost_nvalue, nvalue.

Keywords characteristic of a constraint: all different, disequality.
constraint type: soft constraint, value constraint, relaxation, variable-based violation measure.
final graph structure: strongly connected component, equivalence.
Graph model

We generate a clique with binary *equalities* constraints between each pair of vertices (this include an arc between a vertex and itself) and we state that $C$ is equal to the difference between the total number of variables and the number of strongly connected components.

Parts (A) and (B) of Figure 5.554 respectively show the initial and final graph associated with the Example slot. Since we use the NSCC graph property we show the different strongly connected components of the final graph. Each strongly connected component of the final graph includes all variables that take the same value. Since we have 6 variables and 3 strongly connected components the cost variable $C$ is greater than or equal to $6 - 3$.

Figure 5.554: Initial and final graph of the soft_allDifferent_var constraint
5.325  soft_cumulative

DESCRIPTION

Origin
Derived from cumulative

Constraint
soft_cumulative(TASKS, LIMIT, INTERMEDIATE_LEVEL, SURFACE_ON_TOP)

Arguments

<table>
<thead>
<tr>
<th>TASKS</th>
<th>collection (origin−dvar, duration−dvar, end−dvar, height−dvar)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIMIT</td>
<td>int</td>
</tr>
<tr>
<td>INTERMEDIATE_LEVEL</td>
<td>int</td>
</tr>
<tr>
<td>SURFACE_ON_TOP</td>
<td>dvar</td>
</tr>
</tbody>
</table>

Restrictions
require_at_least(2, TASKS, [origin, duration, end])
required(TASKS.height)
TASKS.duration ≥ 0
TASKS.origin ≤ TASKS.end
TASKS.height ≥ 0
LIMIT ≥ 0
INTERMEDIATE_LEVEL ≥ 0
INTERMEDIATE_LEVEL ≤ LIMIT
SURFACE_ON_TOP ≥ 0

Purpose
Consider a set \( T \) of \( n \) tasks described by the TASKS collection, where \( \text{origin}_j, \text{duration}_j, \text{end}_j, \text{height}_j \) are shortcuts for \( \text{TASKS}[j].\text{origin}, \text{TASKS}[j].\text{duration}, \text{TASKS}[j].\text{end}, \text{TASKS}[j].\text{height} \). In addition let \( \alpha \) and \( \beta \) respectively denote the earliest possible start over all tasks and the latest possible end over all tasks. The soft_cumulative constraint enforces the three following conditions:

1. For each task \( \text{TASKS}[j] \) (1 ≤ \( j \) ≤ \( n \)) of \( T \) we have \( \text{origin}_j + \text{duration}_j = \text{end}_j \).
2. At each point in time, the cumulated height of the set of tasks that overlap that point, does not exceed a given limit \( \text{LIMIT} \) (i.e., \( \forall i \in [\alpha, \beta] : \sum_{j \in [1,n]} \text{origin}_j \leq i < \text{end}_j \text{ height}_j \leq \text{LIMIT} \)).
3. The surface of the profile resource utilisation, which is greater than \( \text{INTERMEDIATE_LEVEL} \), is equal to \( \text{SURFACE_ON_TOP} \) (i.e., \( \sum_{i \in [\alpha, \beta]} \max(0, (\sum_{j \in [1,n]} \text{origin}_j \leq i < \text{end}_j \text{ height}_j) - \text{INTERMEDIATE_LEVEL}) = \text{SURFACE_ON_TOP} \)).

Example

| origin − 1 | duration − 4 | end − 5 | height − 1, |
| origin − 1 | duration − 1 | end − 3 | height − 2, |
| origin − 3 | duration − 3 | end − 6 | height − 2 |

| ,3,3,3 |

Figure 5.555 shows the cumulated profile associated with the example. To each
task of the cumulative constraint corresponds a set of rectangles coloured with the same colour: the sum of the lengths of the rectangles corresponds to the duration of the task, while the height of the rectangles (i.e., all the rectangles associated with a task have the same height) corresponds to the resource consumption of the task. The soft_cumulative constraint holds since:

1. For each task we have that its end is equal to the sum of its origin and its duration.
2. At each point in time we do not have a cumulated resource consumption strictly greater than the upper limit LIMIT = 3 enforced by the second argument of the soft_cumulative constraint.
3. The surface of the cumulated profile located on top of the intermediate level INTERMEDIATE_LEVEL = 2 is equal to SURFACE_ON_TOP = 3.
Keywords  constraint type: predefined constraint, soft constraint, scheduling constraint, resource constraint, temporal constraint, relaxation.
5.326 soft_same_interval_var

### DESCRIPTION

Derived from same.interval

### LINKS

soft_same_interval_var(C, VARIABLES1, VARIABLES2, SIZE_INTERVAL)

### GRAPH

soft_same_interval.

### Arguments

- **C**: dvar
- **VARIABLES1**: collection(var–dvar)
- **VARIABLES2**: collection(var–dvar)
- **SIZE_INTERVAL**: int

### Restrictions

1. \( C \geq 0 \)
2. \( C \leq |\text{VARIABLES1}| \)
3. \( |\text{VARIABLES1}| = |\text{VARIABLES2}| \)
4. required(\text{VARIABLES1}.var)
5. required(\text{VARIABLES2}.var)
6. \( \text{SIZE_INTERVAL} > 0 \)

### Purpose

Let \( N_i \) (respectively \( M_i \)) denote the number of variables of the collection \( \text{VARIABLES1} \) (respectively \( \text{VARIABLES2} \)) that take a value in the interval \([3 \cdot k, 3 \cdot k + 2]\), where \( k \) is an integer. Consequently the values of the collections \((9, 9, 9, 9, 1)\) and \((9, 1, 1, 1, 8)\) are respectively located within intervals \([9, 11], [9, 11], [9, 11], [9, 11], [9, 11], [0, 2]\) and intervals \([9, 11], [0, 2], [0, 2], [0, 2], [0, 2], [6, 8]\). Since there is a correspondence between two pairs of intervals we must unset at least \( 6 - 2 \) items (6 is the number of items of the \( \text{VARIABLES1} \) and \( \text{VARIABLES2} \) collections). Consequently, the soft_same_interval_var constraint holds since its first argument \( C \) is set to \( 6 - 2 \).

### Example

\[
\left( \begin{array}{c}
\text{var} - 9, \\
\text{var} - 9, \\
\text{var} - 9, \\
\text{var} - 9, \\
\text{var} - 9, \\
\text{var} - 9, \\
\text{var} - 1 \\
\text{var} - 1, \\
\text{var} - 1, \\
\text{var} - 1, \\
\text{var} - 1, \\
\text{var} - 8 \\
\end{array} \right)
\]

In the example, the fourth argument \( \text{SIZE_INTERVAL} = 3 \) defines the following family of intervals \([3 \cdot k, 3 \cdot k + 2]\), where \( k \) is an integer. Consequently the values of the collections \((9, 9, 9, 9, 1)\) and \((9, 1, 1, 1, 8)\) are respectively located within intervals \([9, 11], [9, 11], [9, 11], [9, 11], [9, 11], [0, 2]\) and intervals \([9, 11], [0, 2], [0, 2], [0, 2], [0, 2], [6, 8]\). Since there is a correspondence between two pairs of intervals we must unset at least \( 6 - 2 \) items (6 is the number of items of the \( \text{VARIABLES1} \) and \( \text{VARIABLES2} \) collections). Consequently, the soft_same_interval_var constraint holds since its first argument \( C \) is set to \( 6 - 2 \).
Typical

\[
\begin{align*}
C & > 0 \\
|\text{VARIABLES1}| & > 1 \\
\text{range}(\text{VARIABLES1}.\text{var}) & > 1 \\
\text{range}(\text{VARIABLES2}.\text{var}) & > 1 \\
\text{SIZE_INTERVAL} & > 1 \\
\text{SIZE_INTERVAL} & < \text{range}(\text{VARIABLES1}.\text{var}) \\
\text{SIZE_INTERVAL} & < \text{range}(\text{VARIABLES2}.\text{var})
\end{align*}
\]

Symmetries

- Arguments are permutable w.r.t. permutation \((C) (\text{VARIABLES1}, \text{VARIABLES2}) (\text{SIZE_INTERVAL})\).
- Items of \text{VARIABLES1} are permutable.
- Items of \text{VARIABLES2} are permutable.
- An occurrence of a value of \text{VARIABLES1}.\text{var} that belongs to the \(k\)-th interval, of size \text{SIZE_INTERVAL}, can be replaced by any other value of the same interval.
- An occurrence of a value of \text{VARIABLES2}.\text{var} that belongs to the \(k\)-th interval, of size \text{SIZE_INTERVAL}, can be replaced by any other value of the same interval.

Usage

A soft \textit{same_interval} constraint.

Algorithm

See algorithm of the \textit{soft_same_var} constraint.

See also

- \textbf{hard version}: \textit{same_interval}.
- \textbf{implies}: \textit{soft_used_by_interval_var}.

Keywords

- \textbf{constraint arguments}: constraint between two collections of variables.
- \textbf{constraint type}: soft constraint, relaxation, variable-based violation measure.
- \textbf{modelling}: interval.
Graph model

Parts (A) and (B) of Figure 5.556 respectively show the initial and final graph associated with the Example slot. Since we use the **NSINK_NSOURCE** graph property, the source and sink vertices of the final graph are stressed with a double circle. The `soft_same_interval_var` constraint holds since the cost 4 corresponds to the difference between the number of variables of `VARIABLES1` and the sum over the different connected components of the minimum number of sources and sinks.

Figure 5.556: Initial and final graph of the `soft_same_interval_var` constraint
5.327  **soft_same_modulo_var**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from <em>same_modulo</em></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td><em>soft_same_modulo_var</em>(C, VARIABLES1, VARIABLES2, M)</td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td><em>soft_same_modulo</em>.</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>C : dvar</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES1 : <em>collection</em>(var−dvar)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES2 : <em>collection</em>(var−dvar)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M : int</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>C ≥ 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C ≤</td>
<td>VARIABLES1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VARIABLES1</td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES1, var)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES2, var)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M &gt; 0</td>
<td></td>
</tr>
</tbody>
</table>

**Purpose**

For each integer *R* in [0, *M* − 1], let *N1R* (respectively *N2R*) denote the number of variables of VARIABLES1 (respectively VARIABLES2) that have *R* as a rest when divided by *M*. *C* is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that for all *R* in [0, *M* − 1] we have *N1R* = *N2R*.

**Example**

\[
\begin{pmatrix}
\text{var} - 9, \\
\text{var} - 9, \\
\text{var} - 9, \\
\text{var} - 9, \\
\text{var} - 9, \\
\text{var} - 1, \\
\text{var} - 1, \\
\text{var} - 1, \\
\text{var} - 1, \\
\text{var} - 8 \\
\end{pmatrix},
\]

In the example, the values of the collections \{9, 9, 9, 9, 9\} and \{9, 1, 1, 1, 1, 8\} are respectively associated with the equivalence classes 9 mod 3 = 0, 9 mod 3 = 0, 9 mod 3 = 0, 9 mod 3 = 0, 9 mod 3 = 0, 9 mod 3 = 0, 1 mod 3 = 1, 1 mod 3 = 1, 1 mod 3 = 1, 1 mod 3 = 1, 8 mod 3 = 2. Since there is a correspondence between two pairs of equivalence classes we must unset at least 6 − 2 items (6 is the number of items of the VARIABLES1 and VARIABLES2 collections). Consequently, the *soft_same_modulo_var* constraint holds since its first argument *C* is set to 6 − 2.
Typical

<table>
<thead>
<tr>
<th>C &gt; 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>range(VARIABLES1.var) &gt; 1</td>
</tr>
<tr>
<td>range(VARIABLES2.var) &gt; 1</td>
</tr>
<tr>
<td>M &gt; 1</td>
</tr>
<tr>
<td>M &lt; maxval(VARIABLES1.var)</td>
</tr>
<tr>
<td>M &lt; maxval(VARIABLES2.var)</td>
</tr>
</tbody>
</table>

Symmetries

- Arguments are permutable w.r.t. permutation (C) (VARIABLES1, VARIABLES2) (M).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value u of VARIABLES1.var can be replaced by any other value v such that v is congruent to u modulo M.
- An occurrence of a value u of VARIABLES2.var can be replaced by any other value v such that v is congruent to u modulo M.

Usage

A soft same_modulo constraint.

Algorithm

See algorithm of the soft_same_var constraint.

See also

- hard version: same_modulo.
- implies: soft_used_by_modulo_var.

Keywords

- characteristic of a constraint: modulo.
- constraint arguments: constraint between two collections of variables.
- constraint type: soft constraint, relaxation, variable-based violation measure.
Arc input(s) \( \text{VARIABLES1, VARIABLES2} \)

Arc generator \( \text{PRODUCT} \overset{\text{collection}}{\rightarrow} \text{collection(\text{variables1, variables2})} \)

Arc arity 2

Arc constraint(s) \( \text{variables1.var mod M = variables2.var mod M} \)

Graph property(ies) \( \text{NSINK_NSOURCE = |VARIABLES1| - C} \)

Graph model

Parts (A) and (B) of Figure 5.557 respectively show the initial and final graph associated with the Example slot. Since we use the \text{NSINK_NSOURCE} graph property, the source and sink vertices of the final graph are stressed with a double circle. The \text{soft_same_modulo_var} constraint holds since the cost 4 corresponds to the difference between the number of variables of \text{VARIABLES1} and the sum over the different connected components of the minimum number of sources and sinks.

\[ \text{NSINK_NSOURCE = min(5,1) + min(1,4) = 2} \]

Figure 5.557: Initial and final graph of the \text{soft_same_modulo_var} constraint
5.328  soft_same_partition_var

**Origin**
Derived from `same_partition`

**Constraint**
`soft_same_partition_var(C, VARIABLES1, VARIABLES2, PARTITIONS)`

**Synonym**
`soft_same_partition`.

**Type**
VALUES : `collection(val=int)`

**Arguments**
- `C` : `dvar`
- `VARIABLES1` : `collection(var=dvar)`
- `VARIABLES2` : `collection(var=dvar)`
- `PARTITIONS` : `collection(p=VALUES)`

**Restrictions**
- `C ≥ 0`
- `C ≤ |VARIABLES1|`
- `|VARIABLES1| = |VARIABLES2|`
- `required(VARIABLES1, var)`
- `required(VARIABLES2, var)`
- `required(PARTITIONS, p)`
- `|PARTITIONS| ≥ 2`
- `|VALUES| ≥ 1`
- `required(VALUES, val)`
- `distinct(VALUES, val)`

**Purpose**
For each integer $i$ in $[1, |PARTITIONS|]$, let $N1_i$ (respectively $N2_i$) denote the number of variables of `VARIABLES1` (respectively `VARIABLES2`) that take their value in the $i^{th}$ partition of the collection `PARTITIONS`. $C$ is the minimum number of values to change in the `VARIABLES1` and `VARIABLES2` collections so that for all $i$ in $[1, |PARTITIONS|]$ we have $N1_i = N2_i$. 
Example

\[
\begin{pmatrix}
\text{var} - 9, \\
\text{var} - 9, \\
4, \\
\text{var} - 9, \\
\text{var} - 9, \\
\text{var} - 9, \\
\text{var} - 1 \\
\text{var} - 9, \\
\text{var} - 1, \\
\text{var} - 1, \\
\text{var} - 1, \\
\text{var} - 8 \\
p - (1, 2), \\
p - (9), \\
p - (7, 8)
\end{pmatrix}
\]

In the example, the values of the collections \( \langle 9, 9, 9, 9, 1 \rangle \) and \( \langle 9, 1, 1, 1, 8 \rangle \) are respectively associated with the partitions \( p - \langle 9 \rangle, p - \langle 9 \rangle, p - \langle 9 \rangle, p - \langle 9 \rangle, p - \langle 9 \rangle \), \( p - (1, 2) \) and \( p - \langle 9 \rangle, p - (1, 2), p - (1, 2), p - (1, 2), p - (7, 8) \). Since there is a correspondence between two pairs of partitions we must unset at least \( 6 - 2 \) items (6 is the number of items of the VARIABLES1 and VARIABLES2 collections). Consequently, the \texttt{soft\_same\_partition\_var} constraint holds since its first argument \( C \) is set to \( 6 - 2 \).

Typical

\[
\begin{align*}
C & > 0 \\
|\text{VARIABLES1}| & > 1 \\
\text{range}(\text{VARIABLES1}.\text{var}) & > 1 \\
\text{range}(\text{VARIABLES2}.\text{var}) & > 1 \\
|\text{VARIABLES1}| & > |\text{PARTITIONS}| \\
|\text{VARIABLES2}| & > |\text{PARTITIONS}|
\end{align*}
\]

Symmetries

- Arguments are \texttt{permutable} w.r.t. permutation \((C) \ (\text{VARIABLES1}, \text{VARIABLES2}) (\text{PARTITIONS})\).
- Items of VARIABLES1 are \texttt{permutable}.
- Items of VARIABLES2 are \texttt{permutable}.
- Items of PARTITIONS are \texttt{permutable}.
- Items of PARTITIONS.p are \texttt{permutable}.
- An occurrence of a value of VARIABLES1.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.
- An occurrence of a value of VARIABLES2.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.

Usage

A soft \texttt{same\_partition} constraint.

Algorithm

See algorithm of the \texttt{soft\_same\_var} constraint.

See also

\texttt{hard\_version: same\_partition.}

\texttt{implies: soft\_used\_by\_partition\_var.}
Keywords

- characteristic of a constraint: partition.
- constraint arguments: constraint between two collections of variables.
- constraint type: soft constraint, relaxation, variable-based violation measure.
Arc input(s) VARIABLES1 VARIABLES2
Arc generator \textit{PRODUCT}\rightarrow\textit{collection}(\textit{variables1}, \textit{variables2})
Arc arity 2
Arc constraint(s) \textit{in\_same\_partition}(\textit{variables1\_var}, \textit{variables2\_var}, \textit{PARTITIONS})
Graph property(ies) \textit{NSINK\_NSOURCE}=|\textit{VARIABLES1}|−C

Graph model
Parts (A) and (B) of Figure 5.558 respectively show the initial and final graph associated with the \textbf{Example} slot. Since we use the \texttt{NSINK\_NSOURCE} graph property, the source and sink vertices of the final graph are stressed with a double circle. The \texttt{soft\_same\_partition\_var} constraint holds since the cost 4 corresponds to the difference between the number of variables of \texttt{VARIABLES1} and the sum over the different connected components of the minimum number of sources and sinks.

\begin{center}
\includegraphics[width=\textwidth]{figure}
\end{center}

(A)

(B) \texttt{NSINK\_NSOURCE}=\text{min}(5,1)+\text{min}(1,4)=2

Figure 5.558: Initial and final graph of the \texttt{soft\_same\_partition\_var} constraint
5.329  soft_same_var

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin [398]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint soft_same_var(C, VARIABLES1, VARIABLES2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synonym soft_same.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments C : dvar VARIABLES1 : collection(var–dvar) VARIABLES2 : collection(var–dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions C ≥ 0 C ≤</td>
<td>VARIABLES1</td>
<td></td>
</tr>
<tr>
<td>Purpose C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that the variables of the VARIABLES2 collection correspond to the variables of the VARIABLES1 collection according to a permutation.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As illustrated by Figure 5.559, there is a correspondence between two pairs of values of the collections (9, 9, 9, 9, 9, 1) and (9, 1, 1, 1, 1, 8). Consequently, we must unset at least 6 – 2 items (6 is the number of items of the VARIABLES1 and VARIABLES2 collections). The soft_same_var constraint holds since its first argument C is set to 6 – 2.

Typical C > 0 |VARIABLES1| > 1 range(VARIABLES1.var) > 1 range(VARIABLES2.var) > 1
## Symmetries
- Arguments are **permutable** w.r.t. permutation (C) \((\text{VARIABLES1}, \text{VARIABLES2})\).
- Items of \text{VARIABLES1} are **permutable**.
- Items of \text{VARIABLES2} are **permutable**.
- All occurrences of two distinct values in \text{VARIABLES1.var} or \text{VARIABLES2.var} can be **swapped**; all occurrences of a value in \text{VARIABLES1.var} or \text{VARIABLES2.var} can be **renamed** to any unused value.

## Usage
A soft **same** constraint.

## Algorithm
[398, page 80].

## See also
**hard version**: **same**.

**implies**: **soft_used_by_var**.

## Keywords
**constraint arguments**: constraint between two collections of variables.
**constraint type**: soft constraint, relaxation, variable-based violation measure.
**filtering**: minimum cost flow.

![Figure 5.559: Correspondence between collection \(\langle 9, 9, 9, 9, 9, 1 \rangle\) and collection \(\langle 9, 1, 1, 1, 1, 8 \rangle\)](image-url)
Graph model

Parts (A) and (B) of Figure 5.560 respectively show the initial and final graph associated with the Example slot. Since we use the \texttt{NSINK\_NSOURCE} graph property, the source and sink vertices of the final graph are stressed with a double circle. The soft \texttt{same\_var} constraint holds since the cost 4 corresponds to the difference between the number of variables of \texttt{VARIABLES1} and the sum over the different connected components of the minimum number of sources and sinks.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{figure5_560.png}
\caption{Initial and final graph of the soft\_same\_var constraint}
\end{figure}
5.330 soft_used_by_interval_var

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from used_by_interval.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>soft_used_by_interval_var(C, VARIABLES1, VARIABLES2, SIZE_INTERVAL)</td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td>soft_used_by_interval.</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>C : dvar</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES1 : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VARIABLES2 : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SIZE_INTERVAL : int</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>C ≥ 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C ≤</td>
<td>VARIABLES2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VARIABLES1</td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES1.var)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>required(VARIABLES2.var)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SIZE_INTERVAL &gt; 0</td>
<td></td>
</tr>
</tbody>
</table>

**Purpose**

Let $N_i$ (respectively $M_i$) denote the number of variables of the collection VARIABLES1 (respectively VARIABLES2) that take a value in the interval $[\text{SIZE_INTERVAL} \cdot i, \text{SIZE_INTERVAL} \cdot i + \text{SIZE_INTERVAL} - 1]$. C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that for all integer $i$ we have $M_i > 0 \Rightarrow N_i \geq M_i$.

**Example**

\[
\begin{pmatrix}
2, (9,1,1,8,8),
(9,9,9,1), 3
\end{pmatrix}
\]

In the example, the fourth argument $\text{SIZE_INTERVAL} = 3$ defines the following family of intervals $[3 \cdot k, 3 \cdot k + 2]$, where $k$ is an integer. Consequently the values of the collections $\langle 9,1,1,8,8 \rangle$ and $\langle 9,9,9,1 \rangle$ are respectively located within intervals $[9,11]$, $[0,2]$, $[0,2]$, $[6,8]$, $[6,8]$ and intervals $[9,11]$, $[9,11]$, $[9,11]$, $[0,2]$. Since there is a correspondence between two pairs of intervals we must unset at least $4 - 2$ items ($4$ is the number of items of the VARIABLES2 collection). Consequently, the soft_used_by_interval_var constraint holds since its first argument $C$ is set to $4 - 2$.

**Typical**

\[
\begin{align*}
C & > 0 \\
|\text{VARIABLES1}| & > 1 \\
|\text{VARIABLES2}| & > 1 \\
\text{range(}\text{VARIABLES1}.var\text{)} & > 1 \\
\text{range(}\text{VARIABLES2}.var\text{)} & > 1 \\
\text{SIZE_INTERVAL} & > 1 \\
\text{SIZE_INTERVAL} & < \text{range(}\text{VARIABLES1}.var\text{)} \\
\text{SIZE_INTERVAL} & < \text{range(}\text{VARIABLES2}.var\text{)}
\end{align*}
\]
Symmetries

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value of VARIABLES1.var that belongs to the k-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.
- An occurrence of a value of VARIABLES2.var that belongs to the k-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.

Usage

A soft used_by_interval constraint.

See also

hard version: used_by_interval.

implied by: soft_same_interval.var.

Keywords

constraint arguments: constraint between two collections of variables.

constraint type: soft constraint, relaxation, variable-based violation measure.

modelling: interval.
Arc input(s) \text{VARIABLES1, VARIABLES2} \\
Arc generator \text{PRODUCT} \rightarrow \text{collection} (\text{variables1, variables2}) \\
Arc arity 2 \\
Arc constraint(s) variables1.\text{var}/\text{SIZE_INTERVAL} = \text{variables2.\text{var}/SIZE_INTERVAL} \\
Graph property(ies) \text{NSINK_NSOURCE} = |\text{VARIABLES2}| - C \\

Graph model

Parts (A) and (B) of Figure 5.561 respectively show the initial and final graph associated with the Example slot. Since we use the \text{NSINK_NSOURCE} graph property, the source and sink vertices of the final graph are stressed with a double circle. The soft_used_by_interval_var constraint holds since the cost 2 corresponds to the difference between the number of variables of \text{VARIABLES2} and the sum over the different connected components of the minimum number of sources and sinks.

\text{NSINK_NSOURCE} = \text{min(1,3)+min(2,1)=2} 

Figure 5.561: Initial and final graph of the soft_used_by_interval_var constraint
5.331 soft_used_by_modulo_var

**DESCRIPTION**

<table>
<thead>
<tr>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Origin**

Derived from used_by_modulo

**Constraint**

soft_used_by_modulo_var(C, VARIABLES1, VARIABLES2, M)

**Synonym**

soft_used_by_modulo.

**Arguments**

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>dvar</td>
</tr>
<tr>
<td>VARIABLES1</td>
<td>collection(var−dvar)</td>
</tr>
<tr>
<td>VARIABLES2</td>
<td>collection(var−dvar)</td>
</tr>
<tr>
<td>M</td>
<td>int</td>
</tr>
</tbody>
</table>

**Restrictions**

C ≥ 0
C ≤ |VARIABLES2|
|VARIABLES1| ≥ |VARIABLES2|
required(VARIABLES1.var)
required(VARIABLES2.var)
M > 0

**Purpose**

For each integer R in [0, M − 1], let N1_R (respectively N2_R) denote the number of variables of VARIABLES1 (respectively VARIABLES2) that have R as a rest when divided by M. C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that for all R in [0, M − 1] we have N2_R > 0 ⇒ N1_R ≥ N2_R.

**Example**

\[
\begin{pmatrix}
2, \langle 9, 1, 1, 8, 8 \rangle, \\
\langle 9, 9, 9, 1 \rangle, 3
\end{pmatrix}
\]

In the example, the values of the collections \langle 9, 1, 1, 8, 8 \rangle and \langle 9, 9, 9, 1 \rangle are respectively associated with the equivalence classes 9 mod 3 = 0, 1 mod 3 = 1, 1 mod 3 = 1, 8 mod 3 = 2, 8 mod 3 = 2 and 9 mod 3 = 0, 9 mod 3 = 0, 9 mod 3 = 0, 1 mod 3 = 1. Since there is a correspondence between two pairs of equivalence classes we must unset at least 4 − 2 items (4 is the number of items of the VARIABLES2 collection). Consequently, the soft_used_by_modulo_var constraint holds since its first argument C is set to 4 − 2.

**Typical**

C > 0
|VARIABLES1| > 1
|VARIABLES2| > 1
range(VARIABLES1.var) > 1
range(VARIABLES2.var) > 1
M > 1
M < maxval(VARIABLES1.var)
M < maxval(VARIABLES2.var)
Symmetries

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value $u$ of VARIABLES1.var can be replaced by any other value $v$ such that $v$ is congruent to $u$ modulo $M$.
- An occurrence of a value $u$ of VARIABLES2.var can be replaced by any other value $v$ such that $v$ is congruent to $u$ modulo $M$.

Usage

A soft used by modulo constraint.

See also

hard version: used_by_modulo.

implied by: soft_same_modulo_var.

Keywords

characteristic of a constraint: modulo.

constraint arguments: constraint between two collections of variables.

constraint type: soft constraint, relaxation, variable-based violation measure.
Arc input(s) | VARIABLES1 VARIABLES2
---|---
Arc generator | PRODUCT\rightarrow\text{collection}(\text{variables1},\text{variables2})
Arc arity | 2
Arc constraint(s) | \text{variables1}.\text{var} \mod M = \text{variables2}.\text{var} \mod M
Graph property(ies) | \text{NSINK}_\text{NSOURCE} = |\text{VARIABLES2}| - C

Graph model

Parts (A) and (B) of Figure 5.562 respectively show the initial and final graph associated with the Example slot. Since we use the \text{NSINK}_\text{NSOURCE} graph property, the source and sink vertices of the final graph are stressed with a double circle. The soft_used_by_modulo_var constraint holds since the cost 2 corresponds to the difference between the number of variables of VARIABLES2 and the sum over the different connected components of the minimum number of sources and sinks.

Figure 5.562: Initial and final graph of the soft_used_by_modulo_var constraint
5.332  soft_used_by_partition_var

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from used_by_partition.</td>
<td></td>
</tr>
</tbody>
</table>

**Constraint**

soft_used_by_partition_var(C, VARIABLES1, VARIABLES2, PARTITIONS)

**Synonym**

soft_used_by_partition.

**Type**

VALUES : collection(val=int)

**Arguments**

C : dvar
VARIABLES1 : collection(var=dvar)
VARIABLES2 : collection(var=dvar)
PARTITIONS : collection(p=VALUES)

**Restrictions**

\[ C \geq 0 \]
\[ C \leq |VARIABLES2| \]
\[ |VARIABLES1| \geq |VARIABLES2| \]
\[ required(VARIABLES1, var) \]
\[ required(VARIABLES2, var) \]
\[ required(PARTITIONS, p) \]
\[ |PARTITIONS| \geq 2 \]
\[ |VALUES| \geq 1 \]
\[ required(VALUES, val) \]
\[ distinct(VALUES, val) \]

For each integer \( i \) in \([1, |PARTITIONS|]\), let \( N1_i \) (respectively \( N2_i \)) denote the number of variables of VARIABLES1 (respectively VARIABLES2) that take their value in the \( i^{th} \) partition of the collection PARTITIONS. \( C \) is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that for all \( i \) in \([1, |PARTITIONS|]\) we have \( N2_i > 0 \Rightarrow N1_i \geq N2_i \).

**Purpose**

For each integer \( i \) in \([1, |PARTITIONS|]\), let \( N1_i \) (respectively \( N2_i \)) denote the number of variables of VARIABLES1 (respectively VARIABLES2) that take their value in the \( i^{th} \) partition of the collection PARTITIONS. \( C \) is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that for all \( i \) in \([1, |PARTITIONS|]\) we have \( N2_i > 0 \Rightarrow N1_i \geq N2_i \).

**Example**

\[
\begin{aligned}
&\{2, (9, 1, 1, 8, 8), \\
&(9, 9, 9, 1) , \\
&p \rightarrow (1, 2) , \\
&p \rightarrow (9) , \\
&p \rightarrow (7, 8) \}
\end{aligned}
\]

In the example, the values of the collections \( (9, 1, 1, 8, 8) \) and \( (9, 9, 9, 1) \) are respectively associated with the partitions \( p \rightarrow (9) \), \( p \rightarrow (1, 2) \), \( p \rightarrow (1, 2) \), \( p \rightarrow (7, 8) \), \( p \rightarrow (7, 8) \) and \( p \rightarrow (9) \), \( p \rightarrow (9) \), \( p \rightarrow (1, 2) \). Since there is a correspondence between two pairs of partitions we must unset at least \( 4 - 2 \) items (\( 4 \) is the number of items of the VARIABLES2 collection). Consequently, the soft_used_by_partition_var constraint holds since its first argument \( C \) is set to \( 4 - 2 \).
Typical

\[
\begin{align*}
C & > 0 \\
|\text{VARIABLES1}| & > 1 \\
|\text{VARIABLES2}| & > 1 \\
\text{range}(\text{VARIABLES1}.\text{var}) & > 1 \\
\text{range}(\text{VARIABLES2}.\text{var}) & > 1 \\
|\text{VARIABLES1}| & > |\text{PARTITIONS}| \\
|\text{VARIABLES2}| & > |\text{PARTITIONS}|
\end{align*}
\]

Symmetries

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- Items of PARTITIONS are permutable.
- Items of PARTITIONS.p are permutable.
- An occurrence of a value of VARIABLES1.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.
- An occurrence of a value of VARIABLES2.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.

Usage

A soft used by partition constraint.

See also

- hard version: used by partition.
- implied by: soft_same_partition_var.

Keywords

- characteristic of a constraint: partition.
- constraint arguments: constraint between two collections of variables.
- constraint type: soft constraint, relaxation, variable-based violation measure.
Arc input(s) \[ \text{VARIABLES1, VARIABLES2} \]
Arc generator \[ \text{PRODUCT} \mapsto \text{collection} \text{(variables1, variables2)} \]
Arc arity 2
Arc constraint(s) \[ \text{in same partition} \text{(variables1.var, variables2.var, PARTITIONS)} \]
Graph property(ies) \[ \text{NSINK_NSOURCE} = |\text{VARIABLES2}| - C \]

Graph model

Parts (A) and (B) of Figure 5.563 respectively show the initial and final graph associated with the Example slot. Since we use the \text{NSINK_NSOURCE} graph property, the source and sink vertices of the final graph are stressed with a double circle. The \text{soft_used_by_partition_var} constraint holds since the cost 2 corresponds to the difference between the number of variables of \text{VARIABLES2} and the sum over the different connected components of the minimum number of sources and sinks.

Figure 5.563: Initial and final graph of the \text{soft_used_by_partition_var} constraint
5.333  soft_used_by_var

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from <code>used_by</code></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td><code>soft_used_by_var(C, VARIABLES1, VARIABLES2)</code></td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td><code>soft_used_by</code></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td><code>C : dvar</code>&lt;br&gt;<code>VARIABLES1 : collection(var−dvar)</code>&lt;br&gt;<code>VARIABLES2 : collection(var−dvar)</code></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td><code>C ≥ 0</code>&lt;br&gt;`C ≤</td>
<td>VARIABLES2</td>
</tr>
<tr>
<td>Purpose</td>
<td><code>C</code> is the minimum number of values to change in the <code>VARIABLES1</code> and <code>VARIABLES2</code> collections so that all the values of the variables of collection <code>VARIABLES2</code> are used by the variables of collection <code>VARIABLES1</code>.</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td><code>(2, ⟨9, 1, 1, 8, 8⟩, ⟨9, 9, 9, 1⟩)</code></td>
<td></td>
</tr>
</tbody>
</table>

As illustrated by Figure 5.564, there is a correspondence between two pairs of values of the collections `(9, 1, 1, 8, 8)` and `(9, 9, 9, 1)`. Consequently, we must unset at least `4 − 2` items (4 is the number of items of the `VARIABLES2` collection). The `soft_used_by_var` constraint holds since its first argument `C` is set to `4 − 2`.

![Diagram](attachment:Diagram.png)

Figure 5.564: Correspondence between collection `(9, 1, 1, 8, 8)` and collection `(9, 9, 9, 1)`

<table>
<thead>
<tr>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>C &gt; 0</code>&lt;br&gt;`</td>
</tr>
</tbody>
</table>
Symmetries

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- All occurrences of two distinct values in VARIABLES1.var or VARIABLES2.var can be swapped; all occurrences of a value in VARIABLES1.var or VARIABLES2.var can be renamed to any unused value.

Usage

A soft used_by constraint.

See also

- hard version: used_by.
- implied by: soft_same_var.

Keywords

- constraint arguments: constraint between two collections of variables.
- constraint type: soft constraint, relaxation, variable-based violation measure.
Arc input(s)  VARIABLES1 VARIABLES2
Arc generator  \textit{PRODUCT} \rightarrow \textit{collection}(\textit{variables1,variables2})
Arc arity  2
Arc constraint(s)  \textit{variables1}\text{.}\text{var} = \textit{variables2}\text{.}\text{var}
Graph property(ies)  \textsf{NSINK\_NSOURCE}=|\textit{VARIABLES2}| - C

Graph model

Parts (A) and (B) of Figure 5.565 respectively show the initial and final graph associated with the Example slot. Since we use the \textsf{NSINK\_NSOURCE} graph property, the source and sink vertices of the final graph are stressed with a double circle. The soft\_used\_by\_var constraint holds since the cost 2 corresponds to the difference between the number of variables of \textit{VARIABLES2} and the sum over the different connected components of the minimum number of sources and sinks.

Figure 5.565: Initial and final graph of the soft\_used\_by\_var constraint
## 5.334 some_equal

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from <code>alldifferent</code></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td><code>some_equal(VARIABLES)</code></td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td><code>some_eq</code>, <code>not_alldifferent</code>, <code>not_alldiff</code>, <code>not_alldistinct</code>, <code>not_distinct</code>.</td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td><code>VARIABLES : collection(var−dvar)</code></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td><code>required(VARIABLES, var)</code></td>
<td>`</td>
</tr>
<tr>
<td>Purpose</td>
<td>Enforce at least two variables of the collection <code>VARIABLES</code> to be assigned the same value.</td>
<td></td>
</tr>
</tbody>
</table>

### Example

```plaintext
((1, 4, 6))
```

The `some_equal` constraint holds since the first and the third variables are both assigned the same value 1.

### Typical

```
|VARIABLES| > 2
```

### Symmetries

- Items of `VARIABLES` are *permutable*.
- All occurrences of two distinct values of `VARIABLES.var` can be *swapped*; all occurrences of a value of `VARIABLES.var` can be *renamed* to any unused value.

### Arg. properties

Extensible wrt. `VARIABLES`.

### See also

Negation: `alldifferent`.

### Keywords

- Characteristic of a constraint: sort based reformulation.
- Constraint type: value constraint.
Arc input(s) | VARIABLES
---|---
Arc generator | $\text{CLIQUE}(\leq) \rightarrow \text{collection}(\text{variables1}, \text{variables2})$
Arc arity | 2
Arc constraint(s) | variables1.var $=$ variables2.var
Graph property(ies) | NARC $>$ 0

Graph model

We generate a clique with an equality constraint between each pair of distinct vertices and state that the number of arcs of the final graph should be strictly greater than 0.

Parts (A) and (B) of Figure 5.566 respectively show the initial and final graph associated with the Example slot. The some equal constraint holds since the final graph has at one arc, i.e. two variables are assigned the same value.

![Graph Diagram](image)

Figure 5.566: Initial and final graph of the some_equal constraint
### 5.335 sort

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[277]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>sort(VARIABLES1, VARIABLES2)</td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>sortedness, sorted, sorting.</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VARIABLES1 : collection(var−dvar) VARIABLES2 : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>[VARIABLES1] = [VARIABLES2] required(VARIABLES1.var) required(VARIABLES2.var)</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>The variables of the collection VARIABLES2 correspond to the variables of VARIABLES1 according to a permutation. The variables of VARIABLES2 are also sorted in increasing order.</td>
<td></td>
</tr>
</tbody>
</table>

#### Example

```
( var − 1, var − 9, ( var − 1, var − 5, var − 2, var − 1, var − 1, var − 1, var − 1, var − 2, var − 5, var − 9 ) )
```

The sort constraint holds since:

- Values 1, 2, 5 and 9 have the same number of occurrences within both collections (1, 9, 1, 5, 2, 1) and (1, 1, 1, 2, 5, 9). Figure 5.567 illustrates this correspondence.
- The items of collection (1, 1, 1, 2, 5, 9) are sorted in increasing order.

#### Typical

\[ |\text{VARIABLES1}| > 1 \]

\[ \text{range(}\text{VARIABLES1.var}) > 1 \]

#### Symmetries

- Items of VARIABLES1 are permutable.
- One and the same constant can be added to the var attributes of all items of VARIABLES1 and VARIABLES2.
Functional dependency: VARIABLES2 determined by VARIABLES1.

Usage

The main usage of the sort constraint, that was not foreseen when the sort constraint was invented, is its use in many reformulations. Many constraints involving one or several collections of variables become much simpler to express when the variables of these collections are sorted. In addition these reformulations typically have a size that is linear in the number of variables of the original constraint. This justifies why the sort constraint is considered to be a core constraint. As illustrative examples of these types of reformulations we successively consider the alldifferent and the same constraints:

- The alldifferent\((\langle v_1, v_2, \ldots, v_n \rangle)\) constraint can be reformulated as the conjunction sort\((\langle v_1, v_2, \ldots, v_n \rangle, \langle w_1, w_2, \ldots, w_n \rangle)\) ∧ strictly.increasing\((\langle w_1, w_2, \ldots, w_n \rangle)\).
- The same\((\langle u_1, u_2, \ldots, u_n \rangle, \langle v_1, v_2, \ldots, v_n \rangle)\) constraint can be reformulated as the conjunction sort\((\langle u_1, u_2, \ldots, u_n \rangle, \langle w_1, w_2, \ldots, w_n \rangle)\) ∧ sort\((\langle v_1, v_2, \ldots, v_n \rangle, \langle w_1, w_2, \ldots, w_n \rangle)\).

Remark

A variant of this constraint was introduced in [423]. In this variant an additional list of domain variables represents the permutation that allows to go from VARIABLES1 to VARIABLES2.

Algorithm

[74, 262].

Systems

sorting in Choco, sorted in Gecode, sort in MiniZinc, sorting in SICStus.

See also

generalisation: sort_permutation (PERMUTATION parameter added).

implies: lex.greatereq, same.

uses in its reformulation: alldifferent, same.

Keywords

characteristic of a constraint: core, sort.

combinatorial object: permutation.

constraint arguments: constraint between two collections of variables,

pure functional dependency.

filtering: bound-consistency.

modelling: functional dependency.

Figure 5.567: Correspondence between collection \(\langle 1, 9, 1, 5, 2, 1 \rangle\) and collection \(\langle 1, 1, 1, 2, 5, 9 \rangle\)
Parts (A) and (B) of Figure 5.68 respectively show the initial and final graph associated with the first graph constraint of the Example slot. Since it uses the \textit{NSOURCE} and \textit{NSINK} graph properties, the source and sink vertices of this final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. The \textit{sort} constraint holds since:

- Each connected component of the final graph of the first graph constraint has the same number of sources and of sinks.
- The number of sources of the final graph of the first graph constraint is equal to $\|\text{VARIABLES1}\|$. 
- The number of sinks of the final graph of the first graph constraint is equal to $\|\text{VARIABLES2}\|$. 
- Finally the second graph constraint holds also since its corresponding final graph contains exactly $|\text{VARIABLES1} - 1|$ arcs: all the inequalities constraints between consecutive variables of $\text{VARIABLES2}$ holds.

Consider the first graph constraint. Since the initial graph contains only sources and sinks, and since isolated vertices are eliminated from the final graph, we make the following observations:

- Sources of the initial graph cannot become sinks of the final graph.
- Sinks of the initial graph cannot become sources of the final graph.

From the previous observations and since we use the \textit{PRODUCT} arc generator on the collections $\text{VARIABLES1}$ and $\text{VARIABLES2}$, we have that the maximum number of sources and sinks of the final graph is respectively equal to $\|\text{VARIABLES1}\|$ and $\|\text{VARIABLES2}\|$. Therefore we can rewrite $\text{NSOURCE} = \|\text{VARIABLES1}\|$ to $\text{NSOURCE} \geq \|\text{VARIABLES1}\|$ and simplify $\text{NSOURCE}$ to $\text{NSOURCE}$. In a similar way, we can rewrite $\text{NSINK} = \|\text{VARIABLES2}\|$ to $\text{NSINK} \geq \|\text{VARIABLES2}\|$ and simplify $\text{NSINK}$ to $\text{NSINK}$.
Figure 5.568: Initial and final graph of the sort constraint
Consider now the second graph constraint. Since we use the \textit{PATH} arc generator with an arity of 2 on the \textsc{variables2} collection, the maximum number of arcs of the final graph is equal to $|\textsc{variables2}| - 1$. Therefore we can rewrite the graph property $\textsc{narc} = |\textsc{variables2}| - 1$ to $\textsc{narc} \geq |\textsc{variables2}| - 1$ and simplify $\textsc{narc}$ to $\textsc{narc}$. 
**5.336 sort_permutation**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[423]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>sort_permutation(FROM, PERMUTATION, TO)</td>
<td></td>
</tr>
<tr>
<td>Usual name</td>
<td>sort</td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>extended_sortedness, sortedness, sorted, sorting.</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>FROM : collection(var – dvar)</td>
<td>PERMUTATION : collection(var – dvar)</td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The variables of collection FROM correspond to the variables of collection TO according to the permutation PERMUTATION (i.e., FROM[i].var = TO[PERMUTATION[i]].var). The variables of collection TO are also sorted in increasing order.

Example

\[
\begin{pmatrix}
\text{var – 1,} \\
\text{var – 9,} \\
\text{var – 1,} \\
\text{var – 5,} \\
\text{var – 2,} \\
\text{var – 1,} \\
\text{var – 6,} \\
\text{var – 3,} \\
\text{var – 5,} \\
\text{var – 4,} \\
\text{var – 2,} \\
\text{var – 1,} \\
\text{var – 1,} \\
\text{var – 2,} \\
\text{var – 5,} \\
\text{var – 9}
\end{pmatrix}
\]

The sort_permutation constraint holds since:
- The first item \texttt{FROM[1].var = 1} of collection \texttt{FROM} corresponds to the \texttt{PERMUTATION[1].var = 1}st item of collection \texttt{TO}.
- The second item \texttt{FROM[2].var = 9} of collection \texttt{FROM} corresponds to the \texttt{PERMUTATION[2].var = 6}th item of collection \texttt{TO}.
- The third item \texttt{FROM[3].var = 1} of collection \texttt{FROM} corresponds to the \texttt{PERMUTATION[3].var = 3}rd item of collection \texttt{TO}.
- The fourth item \texttt{FROM[4].var = 5} of collection \texttt{FROM} corresponds to the \texttt{PERMUTATION[4].var = 5}th item of collection \texttt{TO}.
- The fifth item \texttt{FROM[5].var = 2} of collection \texttt{FROM} corresponds to the \texttt{PERMUTATION[5].var = 4}th item of collection \texttt{TO}.
- The sixth item \texttt{FROM[6].var = 1} of collection \texttt{FROM} corresponds to the \texttt{PERMUTATION[6].var = 2}nd item of collection \texttt{TO}.

- The items of collection \texttt{TO} = \{1, 1, 1, 2, 5, 9\} are sorted in increasing order.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure5_569.png}
\caption{Illustration of the correspondence between the items of the \texttt{FROM} and the \texttt{TO} collections according to the permutation defined by the items of the \texttt{PERMUTATION} collection}
\end{figure}

\begin{itemize}
\item Typical
<table>
<thead>
<tr>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
</tr>
<tr>
<td>$\texttt{range(FROM.var)} &gt; 1$</td>
</tr>
<tr>
<td>\texttt{lex_different(FROM, TO)}</td>
</tr>
</tbody>
</table>

\item Symmetry
<table>
<thead>
<tr>
<th>Symmetry</th>
</tr>
</thead>
<tbody>
<tr>
<td>One and the same constant can be added to the \texttt{var} attributes of all items of \texttt{FROM} and \texttt{TO}.</td>
</tr>
</tbody>
</table>

\item Arg. properties
<table>
<thead>
<tr>
<th>Arg. properties</th>
</tr>
</thead>
</table>
| $\bullet$ Functional dependency: \texttt{TO} determined by \texttt{FROM}.
| $\bullet$ Functional dependency: \texttt{PERMUTATION} determined by \texttt{FROM} and \texttt{TO}. |

\item Remark
<table>
<thead>
<tr>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>This constraint is referenced under the name \texttt{sorting} in \texttt{SICStus Prolog}.</td>
</tr>
</tbody>
</table>

\item Algorithm
<table>
<thead>
<tr>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>{423}.</td>
</tr>
</tbody>
</table>

\item Reformulation
<table>
<thead>
<tr>
<th>Reformulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let $n$ denote the number of variables in the collection \texttt{FROM}. The sort_permutation constraint can be reformulated as a conjunction of the form: $\texttt{element(PERMUTATION[1], FROM, TO[1])}$, $\texttt{element(PERMUTATION[2], FROM, TO[2])}$.</td>
</tr>
</tbody>
</table>
... element(PERMUTATION[n], FROM, TO[n]),
alldifferent(PERMUTATION),
increasing(TO).

To enhance the previous model, the following necessary condition was proposed by P. Schaus. \( \forall i \in [1, n] : \sum_{j=i}^{n} (\text{FROM}[j] < \text{TO}[i]) \leq i - 1 \) (i.e., at most \( i - 1 \) variables of the collection FROM are assigned a value strictly less than \( \text{TO}[i] \)). Similarly, we have that \( \forall i \in [1, n] : \sum_{j=i}^{n} (\text{FROM}[j] > \text{TO}[i]) \geq n - i \) (i.e., at most \( n - i \) variables of the collection FROM are assigned a value strictly greater than \( \text{TO}[i] \)).

Systems 
sorted in Gecode, sorting in SICStus.

See also 
implies: correspondence.
specialisation: sort (PERMUTATION parameter removed).
used in reformulation: alldifferent, element, increasing.

Keywords 
characteristic of a constraint: sort, derived collection.
combinatorial object: permutation.
constraint arguments: constraint between three collections of variables.
modelling: functional dependency.
### Derived Collection

$col\left(\text{FROM}_{\text{PERMUTATION}}−\text{collection}(\text{var}−\text{dvar}, \text{ind}−\text{dvar}), \right)$

\[
\left[\text{item}((\text{var}−\text{FROM}\.\text{var}, \text{ind}−\text{PERMUTATION}\.\text{var})\right]
\]

### Arc input(s)

FROM\_PERMUTATION TO

### Arc generator

$PRODUCT\mapsto\text{collection}(\text{from}\.\text{permutation}, \text{to})$

### Arc arity

2

### Arc constraint(s)

- from\_permutation\.\text{var} = to\.\text{var}
- from\_permutation\.\text{ind} = to\.\text{key}

### Graph property(ies)

$\text{NARC} = |\text{PERMUTATION}|$

### Arc input(s)

TO

### Arc generator

$PATH\mapsto\text{collection}(\text{to1}, \text{to2})$

### Arc arity

2

### Arc constraint(s)

$\text{to1}\.\text{var} \leq \text{to2}\.\text{var}$

### Graph property(ies)

$\text{NARC} = |\text{TO}| − 1$

### Graph model

Parts (A) and (B) of Figure 5.570 respectively show the initial and final graph associated with the first graph constraint of the Example slot. In both graphs the source vertices correspond to the items of the derived collection FROM\_PERMUTATION, while the sink vertices correspond to the items of the TO collection. Since the first graph constraint uses the NARC graph property, the arcs of its final graph are stressed in bold. The sort permutation constraint holds since:

- The first graph constraint holds since its final graph contains exactly PERMUTATION arcs.
- Finally the second graph constraint holds also since its corresponding final graph contains exactly $|\text{PERMUTATION} − 1|$ arcs: all the inequalities constraints between consecutive variables of TO holds.

### Signature

Consider the first graph constraint where we use the $PRODUCT$ arc generator. Since all the key attributes of the TO collection are distinct, and because of the second condition from\_permutation\.\text{ind} = to\.\text{key} of the arc constraint, each vertex of the final graph has at most one successor. Therefore the maximum number of arcs of the final graph is equal to $|\text{PERMUTATION}|$. So we can rewrite the graph property $\text{NARC} = |\text{PERMUTATION}|$ to $\text{NARC} \geq |\text{PERMUTATION}|$ and simplify $\text{NARC}$ to $\text{NARC}$.

Consider now the second graph constraint. Since we use the $PATH$ arc generator with an arity of 2 on the TO collection, the maximum number of arcs of the corresponding final graph is equal to $|\text{TO}| − 1$. Therefore we can rewrite $\text{NARC} = |\text{TO}| − 1$ to $\text{NARC} \geq |\text{TO}| − 1$ and simplify $\text{NARC}$ to $\text{NARC}$. 
Figure 5.570: Initial and final graph of the sort_permutation constraint
5.337 stable_compatibility

**Description**

- **Origin**: P. Flener, [41]
- **Constraint**: `stable_compatibility(NODES)`
- **Argument**: `NODES : collection(index=int, father=dvar, prec=sint, inc=sint)`
- **Restrictions**: `required(NODES,[index,father,prec,inc])`
  - `NODES.index ≥ 1`
  - `NODES.index ≤ |NODES|`
  - `distinct(NODES,index)`
  - `NODES.father ≥ 1`
  - `NODES.father ≤ |NODES|`
  - `NODES.prec ≥ 1`
  - `NODES.prec ≤ |NODES|`
  - `NODES.inc ≥ 1`
  - `NODES.inc ≤ |NODES|`
  - `NODES.inc > NODES.index`

**Purpose**

Enforce the construction of a *stably compatible* supertree that is compatible with several given trees. The notion of stable compatibility and its context are detailed in the **Usage** slot.

**Example**

```
  \[
  \begin{array}{c@{\quad}c@{\quad}c@{\quad}c@{\quad}c@{\quad}c}
  \text{index} & \text{father} & \text{prec} & \text{inc} \\
  1 & 4 & \{11,12\} & \emptyset , \\
  2 & 3 & \{8,9\} & \emptyset , \\
  3 & 4 & \{2,10\} & \emptyset , \\
  4 & 5 & \{1,3\} & \emptyset , \\
  5 & 7 & \{4,13\} & \emptyset , \\
  6 & 2 & \{8,14\} & \emptyset , \\
  7 & 6 & \{6,13\} & \emptyset , \\
  8 & 6 & \emptyset & \{9,10,11,12,13,14\} , \\
  9 & 2 & \emptyset & \{10,11,12,13\} , \\
  10 & 3 & \emptyset & \{11,12,13\} , \\
  11 & 1 & \emptyset & \{12,13\} , \\
  12 & 1 & \emptyset & \{13\} , \\
  13 & 5 & \emptyset & \{14\} , \\
  14 & 6 & \emptyset & \emptyset \\
  \end{array}
\]
```

Figure 5.571 shows the two trees we want to merge. Note that the leaves `a` and `f` occur in both trees. Figure 5.572 gives one way to merge the two previous trees. This solution corresponds to the ground instance provided by the example. Note that there exist 7 other ways to merge these two trees. They are respectively depicted by Figures 5.572 to 5.579.
Figure 5.571: The two trees to merge

Figure 5.572: First solution corresponding to the ground instance of the example

Figure 5.573: Second solution

Figure 5.574: Third solution

Figure 5.575: Fourth solution
Figure 5.576: Fifth solution

Figure 5.577: Sixth solution

Figure 5.578: Seventh solution

Figure 5.579: Eighth solution
For example, the trees $T_1$ and $T_2$ of Figure 5.580 have $T$ and $T'$ as supertrees under both weak and strong compatibility. As shown, all the internal nodes of $T'$ can be labelled by
corresponding internal nodes of the two given trees, but this is not the case for the father of
b and g in T. Hence T and four other such supertrees are debatable because they speculate
about the existence of extinct species that were not in any of the given trees. Consider
also the three small trees in Figure 5.58: T3 and T4 have T5 as a supertree under weak
compatibility, as it suffices to contract the arc (3, 2) to get T5 from T4. However, T3 and
T4 have no supertree under strong compatibility, as the most recent common ancestor of b
and c, denoted by mrca(b, c), is the same as mrca(a, b) in T5, namely 1, but not the same
in T4, as mrca(b, c) = 3 is an evolutionary descendant of mrca(a, b) = 2. Also, T4 and
T5 have neither weakly nor strongly compatible supertrees.

Under strong compatibility, a first supertree algorithm was given in [3], with an application
for database management systems; it takes $O(l^2)$ time, where l is the number of leaves
in the given trees. Derived algorithms have emerged from phylogeny, for instance One-
Tree [274]. The first constraint program was proposed in [176], using standard, non-global
constraints. Under weak compatibility, a phylogenetic supertree algorithm can be found
in [374] for instance. Under stable compatibility, the algorithm from computational lin-
guistics of [75] has supertree construction as special case.

See also

root concept: tree.

Keywords

application area: bioinformatics, phylogeny.
constraint type: graph constraint.
final graph structure: tree.
Arc input(s)  NODES
Arc generator  $CLIQUE\mapsto collection(nodes1, nodes2)$
Arc arity  2
Arc constraint(s)  nodes1.father = nodes2.index
Graph property(ies)  
  • MAX_NSCC $\leq$ 1
  • NCC = 1
  • MAX_ID $\leq$ 2
  • $PATH_{FROM\_TO}(index, index, prec) = 1$
  • $PATH_{FROM\_TO}(index, index, inc) = 0$
  • $PATH_{FROM\_TO}(index, inc, index) = 0$

Graph model  
To each distinct leaf (i.e., each species) of the trees to merge corresponds a vertex of the initial graph. To each internal vertex of the trees to merge corresponds also a vertex of the initial graph. Each vertex of the initial graph has the following attributes:

  • An index corresponding to a unique identifier.
  • A father corresponding to the father of the vertex in the final tree. Since the leaves of the trees to merge must remain leaves we remove the index value of all the leaves from all the father variables.
  • A set of precedence constraints corresponding to all the arcs of the trees to merge.
  • A set of incomparability constraints corresponding to the incomparable vertices of each tree to merge.

The arc constraint describes the fact that we link a vertex to its father variable. Finally we use the following six graph properties on our final graph:

  • The first graph property MAX_NSCC $\leq$ 1 enforces the fact that the size of the largest strongly connected component does not exceed one. This avoid having circuits containing more than one vertex. In fact the root of the merged tree is a strongly connected component with one single vertex.
  • The second graph property NCC = 1 imposes having only one single tree.
  • The third graph property $PATH_{FROM\_TO}(index, index, prec) = 1$ enforces for each vertex $i$ a set of precedence constraints; for each vertex $j$ of the precedence set there is a path from $i$ to $j$ in the final graph.
  • The fourth graph property MAX_ID $\leq$ 2 enforces that the number of predecessors (i.e., arcs from a vertex to itself are not counted) of each vertex does not exceed 2 (i.e., the final graph is a binary tree).
  • The fifth and sixth graph properties $PATH_{FROM\_TO}(index, index, inc) = 0$ and $PATH_{FROM\_TO}(index, inc, index) = 0$ enforces for each vertex $i$ a set of incomparability constraints; for each vertex $j$ of the incomparability set there is neither a path from $i$ to $j$, nor a path from $j$ to $i$.

Figures 5.582 and 5.583 respectively show the precedence and the incomparability graphs associated with the Example slot. As it contains too many arcs the initial graph is not shown. Figures 5.572 shows the first solution satisfying all the precedence and incomparability constraints.
Figure 5.582: Precedence graph associated with the two trees to merge described by Figure 5.571

Figure 5.583: Incomparability graph associated with the two trees to merge described by Figure 5.571; the two cliques respectively correspond to the leaves of the two trees to merge.
### 5.338 stage_element

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
</table>

**Origin**: Choco, derived from `element`

**Constraint**: `stage_element(ITEM, TABLE)`

**Usual name**: `stage_el`

**Synonym**: `stage_elem`

**Arguments**

- **ITEM**: `collection(index−dvar, value−dvar)`
- **TABLE**: `collection(low−int, up−int, value−int)`

**Restrictions**

- `required(ITEM, [index, value])`
- `ITEM = 1`
- `TABLE > 0`
- `required(TABLE, [low, up, value])`
- `TABLE.low ≤ TABLE.up`
- `increasing_seq(TABLE, [low])`

**Purpose**

Let `low_i`, `up_i`, and `value_i` respectively denote the values of the `low`, `up` and `value` attributes of the `i`th item of the `TABLE` collection. First we have that: `low_i ≤ up_i` and `up_i + 1 = low_{i+1}`.

Second, the `stage_element` constraint enforces the following equivalence:

`low_i ≤ ITEM.index ∧ ITEM.index ≤ up_i ⇔ ITEM.value = value_i`.

**Example**

```
⟨index−5 value−6),
   low−3 up−7 value−6,
   low−8 up−8 value−9,
   low−9 up−14 value−2,
   low−15 up−19 value−9
⟩
```

Figure 5.584 depicts the function associated with the items of the `TABLE` collection. The `stage_element` constraint holds since:

- The value of `ITEM[1].index` is located between the values of the `low` and `up` attributes of the first item of the `TABLE` collection (i.e., `5 ∈ [3, 7]`).
- The value of `ITEM[1].value` corresponds to the `value` attribute of the first item of the `TABLE` collection (i.e., 6).

**Typical**

- `|TABLE| > 1`
- `range(TABLE.value) > 1`
- `TABLE.low < TABLE.up`
Symmetry

All occurrences of two distinct values in ITEM.value or TABLE.value can be swapped; all occurrences of a value in ITEM.value or TABLE.value can be renamed to any unused value.

Arg. properties

- Functional dependency: ITEM.value determined by ITEM.index and TABLE.
- Suffix-extensible wrt. TABLE.

See also

common keyword: elem, element (data constraint).

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.
constraint arguments: binary constraint, pure functional dependency.
constraint network structure: centered cyclic(2) constraint network(1).
constraint type: data constraint.
filtering: arc-consistency.
modelling: table, functional dependency.
Figure 5.584: Function associated with the TABLE collection of the example
Arc input(s) | TABLE
---|---
Arc generator | \( PATH \mapsto \text{collection}(table1, table2) \)
Arc arity | 2
Arc constraint(s)
- \( \text{table1}.\text{low} \leq \text{table1}.\text{up} \)
- \( \text{table1}.\text{up} + 1 = \text{table2}.\text{low} \)
- \( \text{table2}.\text{low} \leq \text{table2}.\text{up} \)

Graph property(ies) | \( \text{NARC} = |\text{TABLE}| - 1 \)

Arc input(s) | ITEM TABLE
---|---
Arc generator | \( PRODUCT \mapsto \text{collection}(\text{item}, \text{table}) \)
Arc arity | 2
Arc constraint(s)
- \( \text{item}.\text{index} \geq \text{table}.\text{low} \)
- \( \text{item}.\text{index} \leq \text{table}.\text{up} \)
- \( \text{item}.\text{value} = \text{table}.\text{value} \)

Graph property(ies) | \( \text{NARC} = 1 \)

Graph model

The first graph constraint models the restrictions on the \( \text{low} \) and \( \text{up} \) attributes of the \( \text{TABLE} \) collection, while the second graph constraint is similar to the one used for defining the \text{element} constraint.

Parts (A) and (B) of Figure 5.585 respectively show the initial and final graph associated with the second graph constraint of the \text{Example} slot. Since we use the \( \text{NARC} \) graph property, the unique arc of the final graph is stressed in bold.

![Graph model](image_url)

Figure 5.585: Initial and final graph of the \text{stage_element} constraint
Automaton

Figure 5.586 depicts the automaton associated with the stage element constraint. Let INDEX and VALUE respectively be the index and the value attributes of the unique item of the ITEM collection. Let LOW, UP, and VALUE, respectively be the low, the up and the value attributes of the $i^{th}$ item of the TABLE collection. To each quintuple (INDEX, VALUE, LOW, UP, VALUE) corresponds a 0-1 signature variable $S_i$ as well as the following signature constraint: $(\text{LOW} \leq \text{INDEX}) \land (\text{INDEX} \leq \text{UP}) \land (\text{VALUE} = \text{VALUE}_i) \Leftrightarrow S_i$.

$Q_0 = s \quad Q_1 \quad Q_n = t$

$S_1 \quad S_2 \quad S_n$

$\text{TABLE_LOW} < \text{ITEM_INDEX} \lor \text{ITEM_INDEX} > \text{TABLE_UP} \lor \text{ITEM_VALUE} \neq \text{TABLE_VALUE}_i$

$\text{TABLE_LOW} \leq \text{ITEM_INDEX} \land \text{ITEM_INDEX} \leq \text{TABLE_UP} \land \text{ITEM_VALUE} = \text{TABLE_VALUE}_i$

Figure 5.586: Automaton of the stage element constraint

Figure 5.587: Hypergraph of the reformulation corresponding to the automaton of the stage element constraint
5.339  stretch_circuit

**DESCRIPTION**

- **Constraint**: stretch_circuit(VARIABLES, VALUES)
- **Usual name**: stretch

**ARGUMENTS**

- VARIABLES: collection(var, dvar)
- VALUES: collection(val, int, lmin, int, lmax, int)

**RESTRICTIONS**

- |VARIABLES| > 0
- required(VARIABLES, var)
- |VALUES| > 0
- required(VALUES, [val, lmin, lmax])
- distinct(VALUES, val)
- VALUES.lmin ≤ VALUES.lmax
- VALUES.lmin ≤ |VARIABLES|
- sum(VALUES.lmin) ≤ |VARIABLES|

In order to define the meaning of the stretch_path constraint, we first introduce the notions of stretch and span. Let n be the number of variables of the collection VARIABLES and let i, j (0 ≤ i < n, 0 ≤ j < n) be two positions within the collection of variables VARIABLES such that the following conditions apply:

- If i ≤ j then all variables $X_i, \ldots, X_j$ take a same value from the set of values of the val attribute.
- If i > j then all variables $X_i, \ldots, X_{n-1}, X_0, \ldots, X_j$ take a same value from the set of values of the val attribute.
- $X_{(i-1) \mod n}$ is different from $X_i$.
- $X_{(j+1) \mod n}$ is different from $X_j$.

We call such a set of variables a *stretch*. The span of the stretch is equal to $1 + (j - i) \mod n$, while the value of the stretch is $X_i$. We now define the condition enforced by the stretch_circuit constraint.

Each item (val - v, lmin - s, lmax - t) of the VALUES collection enforces the minimum value $s$ as well as the maximum value $t$ for the span of a stretch of value $v$.

Note that:

1. Having an item (val - v, lmin - s, lmax - t) with $s$ strictly greater than 0 does not mean that value $v$ should be assigned to one of the variables of collection VARIABLES. It rather means that, when value $v$ is used, all stretches of value $v$ must have a span that belong to interval $[s, t]$.
2. A variable of the collection VARIABLES may be assigned a value that is not defined in the VALUES collection.
Example

\[
\begin{pmatrix}
\text{var} - 6, \\
\text{var} - 6, \\
\text{var} - 3, \\
\text{var} - 1, \\
\text{var} - 1, \\
\text{var} - 1, \\
\text{var} - 6, \\
\text{var} - 6 \\
\text{val} - 1 \ \text{lmin} - 2 \ \text{lmax} - 4, \\
\text{val} - 2 \ \text{lmin} - 2 \ \text{lmax} - 3, \\
\text{val} - 3 \ \text{lmin} - 1 \ \text{lmax} - 6, \\
\text{val} - 6 \ \text{lmin} - 2 \ \text{lmax} - 4
\end{pmatrix}
\]

The \textit{stretch\_circuit} constraint holds since the sequence 6 6 3 1 1 6 6 contains three stretches 6 6 6, 3, and 1 1 1 respectively verifying the following conditions:

- The span of the first stretch 6 6 6 is located within interval [2, 4] (i.e., the limit associated with value 6).
- The span of the second stretch 3 is located within interval [1, 6] (i.e., the limit associated with value 3).
- The span of the third stretch 1 1 1 is located within interval [2, 4] (i.e., the limit associated with value 1).

**Typical**

\[
\begin{align*}
|\text{VARIABLES}| & > 1 \\
\text{range}(\text{VARIABLES}.\text{var}) & > 1 \\
|\text{VARIABLES}| & > |\text{VALUES}| \\
|\text{VALUES}| & > 1 \\
\text{VALUES}.\text{lmax} & \leq |\text{VARIABLES}|
\end{align*}
\]

**Symmetries**

- Items of \text{VARIABLES} can be shifted.
- Items of \text{VALUES} are permutable.
- All occurrences of two distinct values in \text{VARIABLES}.\text{var} or \text{VALUES}.\text{val} can be swapped; all occurrences of a value in \text{VARIABLES}.\text{var} or \text{VALUES}.\text{val} can be renamed to any unused value.

**Usage**

The article \cite{285}, which originally introduced the \textit{stretch} constraint, quotes rostering problems as typical examples of use of this constraint.

**Remark**

We split the origin \textit{stretch} constraint into the \textit{stretch\_circuit} and the \textit{stretch\_path} constraints that respectively use the \textit{PATH LOOP} and \textit{CIRCUIT LOOP} arc generators. We also reorganise the parameters: the \text{VALUES} collection describes the attributes of each value that can be assigned to the variables of the \textit{stretch\_circuit} constraint. Finally we skipped the pattern constraint that tells what values can follow a given value.

**Algorithm**

A first filtering algorithm was described in the original article of G. Pesant \cite{285}. An algorithm that also generates explanations is given in \cite{340}. The first filtering algorithm achieving arc-consistency is depicted in \cite{191, 192}. This algorithm is based on \textit{dynamic programming} and handles the fact that some values can be followed by only a given subset of values.
Reformulation

The **stretch_circuit** constraint can be reformulated in term of a **stretch_path** constraint. Let \( LMAX \) denote the maximum value taken by the \( lmax \) attribute within the items of the collection VALUES, let \( n \) be the number of variables of the collection VARIABLES, and let \( \delta = \min(LMAX, n) \). The first and second arguments of the **stretch_path** constraint are created in the following way:

- We pass to the **stretch_path** the variables of the collection VARIABLES to which we add the \( \delta \) first variables of the collection VARIABLES.
- We pass to the **stretch_path** the values of the collection VALUES with the following modification: to each value \( v \) for which the corresponding \( lmax \) attribute is greater than or equal to \( n \) we reset its value to \( n + \delta \).

Even if **stretch_path** can achieve arc-consistency this reformulation may not achieve arc-consistency since it duplicates variables.

Using this reformulation, the example

\[
\text{stretch_circuit}((6, 6, 3, 1, 1, 1, 6, 6),
\begin{align*}
& (\text{val} - 1 \text{lmin} - 2 \text{lmax} - 4, \text{val} - 2 \text{lmin} - 2 \text{lmax} - 3, \\
& \text{val} - 3 \text{lmin} - 1 \text{lmax} - 6, \text{val} - 6 \text{lmin} - 2 \text{lmax} - 4))
\end{align*}
\]

of the Example slot is reformulated as:

\[
\text{stretch_path}((6, 6, 3, 1, 1, 1, 6, 6, 6, 6, 3, 1, 1, 1),
\begin{align*}
& (\text{val} - 1 \text{lmin} - 2 \text{lmax} - 4, \text{val} - 2 \text{lmin} - 2 \text{lmax} - 3, \\
& \text{val} - 3 \text{lmin} - 1 \text{lmax} - 6, \text{val} - 6 \text{lmin} - 2 \text{lmax} - 4))
\end{align*}
\]

In the reformulation \( \delta \) was equal to 6, and the VALUES collection was left unchanged since no \( lmax \) attribute was equal to the number of variables of the VARIABLES collection (i.e., 8).

See also

- **common keyword**: group (timetabling constraint),
- pattern (sliding sequence constraint, timetabling constraint),
- sliding_distribution (sliding sequence constraint),
- **stretch_path** (sliding sequence constraint, timetabling constraint).

used in reformulation: **stretch_path**.

Keywords

- **characteristic of a constraint**: cyclic.
- **constraint type**: timetabling constraint, sliding sequence constraint.
- **filtering**: dynamic programming, arc-consistency, duplicated variables.
For all items of VALUES:

**Arc input(s)**

**Arc generator**

- \( \text{CIRCUIT} \rightarrow \text{collection}(\text{variables}_1, \text{variables}_2) \)
- \( \text{LOOP} \rightarrow \text{collection}(\text{variables}_1, \text{variables}_2) \)

**Arc arity**

2

**Arc constraint(s)**

- \( \text{variables}_1.\text{var} = \text{VALUES}.\text{val} \)
- \( \text{variables}_2.\text{var} = \text{VALUES}.\text{val} \)

**Graph property(ies)**

- \( \text{not.in}(\text{MIN}_\text{NCC}, 1, \text{VALUES}.\text{lmin} - 1) \)
- \( \text{MAX}_\text{NCC} \leq \text{VALUES}.\text{lmax} \)

**Graph model**

Part (A) of Figure 5.588 shows the initial graphs associated with values 1, 2, 3 and 6 of the Example slot. Part (B) of Figure 5.588 shows the corresponding final graphs associated with values 1, 3 and 6. Since value 2 is not assigned to any variable of the \( \text{VARIABLES} \) collection the final graph associated with value 2 is empty. The \text{stretch_circuit} constraint holds since:

- For value 1 we have one connected component for which the number of vertices is greater than or equal to 2 and less than or equal to 4,
- For value 2 we do not have any connected component,
- For value 3 we have one connected component for which the number of vertices is greater than or equal to 1 and less than or equal to 6,
- For value 6 we have one connected component for which the number of vertices is greater than or equal to 2 and less than or equal to 4.

![Graphs](image-url)
5.340  stretch_path

In order to define the meaning of the stretch_path constraint, we first introduce the notions of stretch and span. Let \(n\) be the number of variables of the collection VARIABLES. Let \(X_i, \ldots, X_j\) (\(1 \leq i \leq j \leq n\)) be consecutive variables of the collection of variables VARIABLES such that the following conditions apply:

- All variables \(X_i, \ldots, X_j\) take a same value from the set of values of the \texttt{val} attribute,
- \(i = 1\) or \(X_{i-1}\) is different from \(X_i\),
- \(j = n\) or \(X_{j+1}\) is different from \(X_j\).

We call such a set of variables a stretch. The span of the stretch is equal to \(j - i + 1\), while the value of the stretch is \(X_i\). We now define the condition enforced by the stretch_path constraint.

Each item \((\text{val} - v, \text{lmin} - s, \text{lmax} - t)\) of the VALUES collection enforces the minimum value \(s\) as well as the maximum value \(t\) for the span of a stretch of value \(v\) over consecutive variables of the VARIABLES collection.

Note that:

1. Having an item \((\text{val} - v, \text{lmin} - s, \text{lmax} - t)\) with \(s\) strictly greater than 0 does not mean that value \(v\) should be assigned to one of the variables of collection VARIABLES. It rather means that, when value \(v\) is used, all stretches of value \(v\) must have a span that belong to interval \([s, t]\).
2. A variable of the collection VARIABLES may be assigned a value that is not defined in the VALUES collection.
The \textbf{stretch\_path} constraint holds since the sequence 6 6 3 1 1 1 6 6 contains four stretches 6 6, 3, 1 1 1, and 6 6 respectively verifying the following conditions:

\begin{itemize}
  \item The span of the first stretch 6 6 is located within interval [2, 2] (i.e., the limit associated with value 6).
  \item The span of the second stretch 3 is located within interval [1, 6] (i.e., the limit associated with value 3).
  \item The span of the third stretch 1 1 1 is located within interval [2, 4] (i.e., the limit associated with value 1).
  \item The span of the fourth stretch 6 6 is located within interval [2, 2] (i.e., the limit associated with value 6).
\end{itemize}

\textbf{Typical}

\begin{align*}
  |\text{VARIABLES}| & > 1 \\
  \text{range}(\text{VARIABLES}.\text{var}) & > 1 \\
  |\text{VARIABLES}| & > |\text{VALUES}| \\
  |\text{VALUES}| & > 1 \\
  \sum(\text{VALUES}.\text{lmin}) & \leq |\text{VARIABLES}| \\
  \text{VALUES}.\text{lmax} & \leq |\text{VARIABLES}| \\
\end{align*}

\textbf{Symmetries}

\begin{itemize}
  \item Items of \text{VARIABLES} can be \textbf{reversed}.
  \item Items of \text{VALUES} are \textbf{permutable}.
  \item All occurrences of two distinct values in \text{VARIABLES}.\text{var} or \text{VALUES}.\text{val} can be \textbf{swapped}; all occurrences of a value in \text{VARIABLES}.\text{var} or \text{VALUES}.\text{val} can be \textbf{renamed} to any unused value.
\end{itemize}

\textbf{Usage}

The article [285], which originally introduced the \textbf{stretch} constraint, quotes rostering problems as typical examples of use of this constraint.

\textbf{Remark}

We split the original \textbf{stretch} constraint into the \textbf{stretch\_path} and the \textbf{stretch\_circuit} constraints that respectively use the \textit{PATH LOOP} and the \textit{CIRCUIT LOOP} arc generators. We also reorganise the parameters: the \text{VALUES} collection describes the attributes of each value that can be assigned to the variables of the \textbf{stretch\_path} constraint. Finally we skipped the pattern constraint that tells what values can follow a given value. A extension of this constraint (i.e., stretch plus pattern), called \textbf{forced\_shift\_stretch}, where one can specify for each value \(v\) with a
0-1 variable, whether it should occur at least once or not at all, was proposed in [192]. By reduction to Hamiltonian path it was shown that enforcing arc-consistency for forced_shift_stretch is NP-hard [192].

**Algorithm**

A first filtering algorithm was described in the original article of G. Pesant [285]. A second filtering algorithm, based on dynamic programming, achieving arc-consistency is depicted in [191, 192]. It also handles the fact that some values can be followed by only a given subset of values. An other alternative achieving arc-consistency is to use the automaton described in the Automaton slot.

**Systems**

stretchPath in Choco, stretch in JaCoP.

**See also**

common keyword: change_continuity, group(timetabling constraint), group_skip_isolated_item(timetabling constraint, sequence), pattern(sliding sequence constraint, timetabling constraint), sliding_distribution(sliding sequence constraint), stretch_circuit(sliding sequence constraint, timetabling constraint).

generalisation: stretch_path_partition(variable replaced by variable ∈ partition).

uses in its reformulation: stretch_circuit.

**Keywords**

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

combinatorial object: sequence.

constraint network structure: Berge-acyclic constraint network.

constraint type: timetabling constraint, sliding sequence constraint.

filtering: dynamic programming, arc-consistency.

final graph structure: consecutive loops are connected.
For all items of VALUES:

<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>( \text{PATH} \rightarrow \text{collection}(\text{variables}.1, \text{variables}.2) )</td>
</tr>
<tr>
<td>Arc generator</td>
<td>( \text{LOOP} \rightarrow \text{collection}(\text{variables}.1, \text{variables}.2) )</td>
</tr>
<tr>
<td>Arc arity</td>
<td>2</td>
</tr>
</tbody>
</table>
| Arc constraint(s) | • \( \text{variables}.1.\text{var} = \text{VALUES}.\text{val} \)  
• \( \text{variables}.2.\text{var} = \text{VALUES}.\text{val} \) |
| Graph property(ies) | • not.in(\text{MIN}_\text{NCC}, 1, \text{VALUES}.\text{lmin} - 1)  
• \( \text{MAX}_\text{NCC} \leq \text{VALUES}.\text{lmax} \) |

Graph model

Part (A) of Figure 5.589 shows the initial graphs associated with values 1, 2, 3 and 6 of the Example slot. Part (B) of Figure 5.589 shows the corresponding final graphs associated with values 1, 3 and 6. Since value 2 is not assigned to any variable of the VARIABLES collection the final graph associated with value 2 is empty. The stretch_path constraint holds since:

- For value 1 we have one connected component for which the number of vertices 3 is greater than or equal to 2 and less than or equal to 4,
- For value 2 we do not have any connected component,
- For value 3 we have one connected component for which the number of vertices 1 is greater than or equal to 1 and less than or equal to 6,
- For value 6 we have two connected components that both contain two vertices: this is greater than or equal to 2 and less than or equal to 2.

![Figure 5.589: Initial and final graph of the stretch_path constraint](image)
During the presentation of this constraint at CP’2001 the following point was mentioned: it could be useful to allow domain variables for the minimum and the maximum values of a stretch. This could be achieved in the following way: the \( l_{\text{min}} \) (respectively \( l_{\text{max}} \)) attribute would now be a domain variable that gives the size of the shortest (respectively longest) stretch. Finally within the Graph property(ies) slot we would replace \( \geq \) (and \( \leq \)) by \( = \).
Automaton

Let \( n \) and \( m \) respectively denote the quantities \(|\text{VARIABLES}|\) and \(|\text{VALUES}|\). Furthermore, let \( \text{val}_i, \text{lmin}_i, \text{lmax}_i, (i \in [1, m]) \), respectively be shortcuts for the expressions \( \text{VALUES}[i].\text{val}, \text{VALUES}[i]_\text{lmin} \) and \( \text{VALUES}[i]_\text{lmax} \). Without loss of generality, we assume that all the \( \text{lmin} \) attributes of the items of the \( \text{VALUES} \) collection are at least equal to 1. The following automaton \( A \) involving \( 1 + \text{lmax}_1 + \text{lmax}_2 + \ldots + \text{lmax}_m \) states only accepts solutions of the stretch path constraint. Automaton \( A \) has the following states:

- an initial state \( s \) that is also a terminal state,
- \( \forall i \in [1, m], \forall j \in [1, \text{lmin}_i - 1], \) a non-terminal state \( s_{i,j} \),
- \( \forall i \in [1, m], \forall j \in [\text{lmin}_i, \text{lmax}_i], \) a terminal state \( s_{i,j} \).

Transitions of \( A \) are defined in the following way:

- \( \forall i \in [1, m], \) a transition from \( s \) to \( s_{i,1} \) labelled by condition \( X_l = \text{val}_i \),
- a transition from \( s \) to \( s \) labelled by condition \( X_l \neq \text{val}_1 \land X_l \neq \text{val}_2 \land \ldots \land X_l \neq \text{val}_m \),
- \( \forall i \in [1, m], \forall j \in [\text{lmin}_i, \text{lmax}_i], \) a transition from \( s_{i,j} \) to \( s \) labelled by condition \( X_l \neq \text{val}_1 \land X_l \neq \text{val}_2 \land \ldots \land X_l \neq \text{val}_m \),
- \( \forall i \in [1, m], \forall j \in [1, \text{lmax}_i - 1], \) a transition from \( s_{i,j} \) to \( s_{i,j+1} \) labelled by condition \( X_l = \text{val}_i \),
- \( \forall i \in [1, m], \forall j \in [\text{lmin}_i, \text{lmax}_i], \forall k \neq i \in [1, m], \) a transition from \( s_{i,j} \) to \( s_{k,1} \) labelled by condition \( X_l = \text{val}_k \).

Figure 5.590 depicts the automaton associated with the stretch path constraint of the Example slot. Transitions labels 0, 1, 2, 3 and 4 respectively correspond to the conditions \( X_l \neq 1 \land X_l \neq 2 \land X_l \neq 3 \land X_l \neq 6 \), \( X_l = 1 \), \( X_l = 2 \), \( X_l = 3 \), \( X_l = 6 \) (since values 1, 2, 3 and 6 respectively correspond to the values of the first, second, third and fourth item of the \( \text{VALUES} \) collection). The stretch path constraint holds since the corresponding sequence of visited states, \( s s_{41} s_{42} s_{31} s_{11} s_{12} s_{13} s_{41} s_{42} \), ends up in a terminal state (i.e., terminal states are depicted by thick circles in the figure).
Figure 5.590: Automaton of the stretch path constraint of the Example slot
5.341 stretch_path_partition

**DESCRIPTION**

Origin: Derived from `stretch_path`.

Constraint: `stretch_path_partition(VARIABLES, PARTLIMITS)`

**SYNONYM**

`stretch`.

**TYPE**

VALUES: `collection(val-int)`

**ARGUMENTS**

VARIABLES: `collection(var-dvar)`

PARTLIMITS: `collection(p - VALUES, lmin-int, lmax-int)`

**RESTRICTIONS**

- `|VALUES| ≥ 1`
- `required(VALUES, val)`
- `distinct(VALUES, val)`
- `|VARIABLES| > 0`
- `required(VARIABLES, var)`
- `|PARTLIMITS| > 0`
- `required(PARTLIMITS, [p, lmin, lmax])`
- `PARTLIMITS.lmin ≥ 0`
- `PARTLIMITS.lmin ≤ PARTLIMITS.lmax`
- `PARTLIMITS.lmin ≤ |VARIABLES|`
In order to define the meaning of the stretch_path_partition constraint, we first introduce the notions of stretch and span. Let $n$ be the number of variables of the collection VARIABLES. Let $X_i, \ldots, X_j$ ($1 \leq i \leq j \leq n$) be consecutive variables of the collection of variables VARIABLES such that the following conditions apply:

- All variables $X_i, \ldots, X_j$ take their values in the same partition of the PARTLIMITS collection (i.e., $\exists l \in [1, |\text{PARTLIMITS}|]$ such that $\forall k \in [i, j] : X_k \in \text{PARTLIMITS}[l].p$),
- $i = 1$ or $X_{i-1}$ is different from $X_i$,
- $j = n$ or $X_{j+1}$ is different from $X_j$.

We call such a set of variables a stretch. The span of the stretch is equal to $j - i + 1$, while the value of the stretch is $l$. We now define the condition enforced by the stretch_path_partition constraint.

Each item $\text{PARTLIMITS}[l] = (p - values, \text{imin} - s, \text{imax} - t)$ of the PARTLIMITS collection enforces the minimum value $s$ as well as the maximum value $t$ for the span of a stretch of value $l$ over consecutive variables of the VARIABLES collection.

Note that:

1. Having an item $\text{PARTLIMITS}[l] = (p - values, \text{imin} - s, \text{imax} - t)$ with $s$ strictly greater than 0 does not mean that values of values should be assigned to one of the variables of collection VARIABLES. It rather means that, when a value of values is used, all stretches of value $l$ must have a span that belong to interval $[s, t]$.

2. A variable of the collection VARIABLES may be assigned a value that is not defined in the attribute $p$ of the PARTLIMITS collection.

**Example**

$$\begin{align*}
\langle \text{var} - 1, \\
\text{var} - 2, \\
\text{var} - 0, \\
\langle \text{var} - 0, \\
\text{var} - 2, \\
\text{var} - 2, \\
\text{var} - 0, \\
\langle p = (1, 2), \text{lmin} - 2, \text{imax} - 4, \\
p = (3), \text{lmin} - 0, \text{imax} - 2 \rangle \rangle
\end{align*}$$

The stretch_path_partition constraint holds since the sequence $1 2 0 0 2 2 2 0$ contains two stretches $1 2$, and $2 2 2$ respectively verifying the following conditions:

- The span of the first stretch $1 2$ is located within interval $[2, 4]$ (i.e., the limit associated with item PARTLIMITS[1]).
- The span of the second stretch $2 2 2$ is located within interval $[2, 4]$ (i.e., the limit associated with item PARTLIMITS[1]).
**Typical**

\[ |\text{VARIABLES}| > 1 \]
\[ \text{range}(\text{VARIABLES}.\text{var}) > 1 \]
\[ |\text{VARIABLES}| > |\text{PARTLIMITS}| \]
\[ |\text{PARTLIMITS}| > 1 \]
\[ \text{sum}(\text{PARTLIMITS}.\text{lmin}) \leq |\text{VARIABLES}| \]
\[ \text{PARTLIMITS}.\text{lmax} \leq |\text{VARIABLES}| \]

**Symmetries**

- Items of \text{VARIABLES} can be reversed.
- Items of \text{PARTLIMITS} are permutable.
- Items of \text{PARTLIMITS}.\text{p} are permutable.
- All occurrences of two distinct tuples of values in \text{VARIABLES}.\text{var} or \text{PARTLIMITS}.\text{p}.\text{val} can be swapped; all occurrences of a tuple of values in \text{VARIABLES}.\text{var} or \text{PARTLIMITS}.\text{p}.\text{val} can be renamed to any unused tuple of values.

**See also**

- **common keyword:** pattern (sliding sequence constraint).
- **specialisation:** stretch path (variable ∈ partition replaced by variable).

**Keywords**

- **characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint, partition.
- **combinatorial object:** sequence.
- **constraint network structure:** Berge-acyclic constraint network.
- **constraint type:** timetabling constraint, sliding sequence constraint.
- **filtering:** arc-consistency.
- **final graph structure:** consecutive loops are connected.
5.342  strict_lex2

**DESCRIPTION**

- **Origin**: [155]
- **Constraint**: `strict_lex2(MATRIX)`
- **Type**: `VECTOR : collection(var - dvar)`
- **Argument**: `MATRIX : collection(vec - VECTOR)`
- **Restrictions**: 
  - `|VECTOR| ≥ 1`
  - `required(VECTOR, var)`
  - `required(MATRIX, vec)`
  - `same_size(MATRIX, vec)`
- **Purpose**: Given a matrix of domain variables, enforces that both adjacent rows, and adjacent columns are lexicographically ordered (adjacent rows and adjacent columns cannot be equal).

**Example**

```
((vec - [2, 2, 3],
  vec - [2, 3, 1]))
```

The `strict_lex2` constraint holds since:

- The first row `[2, 2, 3]` is lexicographically strictly less than the second row `[2, 3, 1]`.
- The first column `[2, 2]` is lexicographically strictly less than the second column `[2, 3]`.
- The second column `[2, 3]` is lexicographically strictly less than the third column `[3, 1]`.

- **Typical**: `|VECTOR| > 1`
  - `|MATRIX| > 1`

- **Symmetry**: One and the same constant can be added to the `var` attribute of all items of `MATRIX.vec`.
- **Usage**: A symmetry-breaking constraint.
- **Reformulation**: The `strict_lex2` constraint can be expressed as a conjunction of two `lex_chain_less` constraints: A first `lex_chain_less` constraint on the `MATRIX` argument and a second `lex_chain_less` constraint on the transpose of the `MATRIX` argument.
- **Systems**: `strict_lex2` in MiniZinc.
See also

- common keyword: allperm, lex_lesseq (lexicographic order).
- implies: lex2, lex_chain_less.
- part of system of constraints: lex_chain_less.

Keywords

- constraint type: predefined constraint, system of constraints, order constraint.
- modelling: matrix, matrix model.
- symmetry: symmetry, matrix symmetry, lexicographic order.
### 5.343 strictly_decreasing

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from strictly_increasing.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>strictly_decreasing(VARIABLES)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>VARIABLES : collection(var−dvar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restriction</td>
<td>required(VARIABLES, var)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>The variables of the collection VARIABLES are strictly decreasing.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>((8, 4, 3, 1))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The strictly_decreasing constraint holds since $8 > 4 > 3 > 1$.

| Typical     | $|\text{VARIABLES}| > 2$ |       |           |
|-------------|---------------------|-------|-----------|
| Symmetry    | One and the same constant can be added to the var attribute of all items of VARIABLES. |       |           |
| Arg. properties | Contractible wrt. VARIABLES. |       |           |
| Systems     | increasingNValue in Choco, rel in Gecode. |       |           |
| See also    | common keyword: increasing (order constraint). |       |           |
|             | comparison swapped: strictly_increasing. |       |           |
|             | implies: alldifferent, decreasing. |       |           |
| Keywords    | characteristic of a constraint: automaton, automaton without counters, reified automaton constraint. |       |           |
|             | constraint network structure: sliding cyclic(1) constraint network(1). |       |           |
|             | constraint type: decomposition, order constraint. |       |           |
|             | filtering: arc-consistency. |       |           |
Arc input(s) | VARIABLES
---|---
Arc generator | $PATH\rightarrow\text{collection}(\text{variables}_1,\text{variables}_2)$
Arc arity | 2
Arc constraint(s) | $\text{variables}_1.\text{var} > \text{variables}_2.\text{var}$
Graph property(ies) | $\text{NARC} = |\text{VARIABLES}| - 1$

Graph model

Parts (A) and (B) of Figure 5.591 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

Figure 5.591: Initial and final graph of the strictly_decreasing constraint
Automaton

Figure 5.592 depicts the automaton associated with the strictly_decreasing constraint. To each pair of consecutive variables \((\text{VAR}_i, \text{VAR}_{i+1})\) of the collection VARIABLES corresponds a 0-1 signature variable \(S_i\). The following signature constraint links \(\text{VAR}_i\), \(\text{VAR}_{i+1}\) and \(S_i\): \(\text{VAR}_i \leq \text{VAR}_{i+1} \leftrightarrow S_i\).

Figure 5.592: Automaton of the strictly_decreasing constraint

Figure 5.593: Hypergraph of the reformulation corresponding to the automaton of the strictly_decreasing constraint
5.344 strictly_increasing

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin: KOALOG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint: strictly_increasing(VARIABLES)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Argument: VARIABLES : collection(var—dvar)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restriction: required(VARIABLES, var)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose: The variables of the collection VARIABLES are strictly increasing.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example**

((1, 3, 6, 8))

The strictly_increasing constraint holds since 1 < 3 < 6 < 8.

**Typical**

| VARIABLES | > 2 |

**Symmetry**

One and the same constant can be added to the var attribute of all items of VARIABLES.

**Arg. properties**

Contractible wrt. VARIABLES.

**Systems**

increasingNValue in Choco, rel in Gecode.

**Used in**

golomb, int_value_precede_chain.

**See also**

common keyword: decreasing (order constraint).

comparison swapped: strictly_decreasing.

implied by: golomb.

implies: alldifferent, increasing.

uses in its reformulation: alldifferent.

**Keywords**

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint network structure: sliding cyclic(1) constraint network(1).

constraint type: decomposition, order constraint.

filtering: arc-consistency.
Arc input(s)  VARIABLES
Arc generator  \( PATH \mapsto \text{collection}(\text{variables1, variables2}) \)
Arc arity  2
Arc constraint(s)  \( \text{variables1}.\text{var} < \text{variables2}.\text{var} \)
Graph property(ies)  \( \text{NARC} = |\text{VARIABLES}| - 1 \)

Graph model  Parts (A) and (B) of Figure 5.594 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

![Graph model diagram](A) ![Graph model diagram](B)

Figure 5.594: Initial and final graph of the strictly increasing constraint
Automaton

Figure 5.595 depicts the automaton associated with the strictly increasing constraint. To each pair of consecutive variables \((\text{VAR}_i, \text{VAR}_{i+1})\) of the collection VARIABLES corresponds a 0-1 signature variable \(S_i\). The following signature constraint links \(\text{VAR}_i\), \(\text{VAR}_{i+1}\) and \(S_i\): \(\text{VAR}_i \geq \text{VAR}_{i+1} \iff S_i\).

![Automaton diagram]

Figure 5.595: Automaton of the strictly increasing constraint

![Hypergraph diagram]

Figure 5.596: Hypergraph of the reformulation corresponding to the automaton of the strictly increasing constraint
5.345 strongly_connected

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[4]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>strongly_connected(NODES)</td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>NODES : collection(index=int,succ=svar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(NODES,[index,succ])</td>
<td>NODES.index ≥ 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NODES.index ≤</td>
</tr>
<tr>
<td></td>
<td></td>
<td>distinct(NODES,index)</td>
</tr>
<tr>
<td>Purpose</td>
<td>Consider a digraph G described by the NODES collection. Select a subset of arcs of G so that we have one single strongly connected component involving all vertices of G.</td>
<td></td>
</tr>
</tbody>
</table>
| Example     | \[
\begin{align*}
  \text{index} - 1 & \quad \text{succ} = \{2\}, \\
  \text{index} - 2 & \quad \text{succ} = \{3\}, \\
  \text{index} - 3 & \quad \text{succ} = \{2,5\}, \\
  \text{index} - 4 & \quad \text{succ} = \{1\}, \\
  \text{index} - 5 & \quad \text{succ} = \{4\}.
\end{align*}
\] |
| Typical     | | | |
| Symmetry    | Items of NODES are permutable. |
| Algorithm   | The sketch of a filtering algorithm for the strongly_connected constraint is given in [131, page 89]. |
| See also    | common keyword: link_set_to_bools (constraint involving set variables). |
|             | implied by: connected. |
|             | related: circuit (one single strongly connected component in the final solution). |
| Keywords    | constraint arguments: constraint involving set variables. |
|             | constraint type: graph constraint. |
|             | filtering: linear programming. |
|             | final graph structure: strongly connected component. |
Arc input(s) \( \text{NODES} \)

Arc generator \( CLIQUE \mapsto \text{collection}(\text{nodes1}, \text{nodes2}) \)

Arc arity 2

Arc constraint(s) \( \text{in}_\text{set}(\text{nodes2}.\text{index}, \text{nodes1}.\text{succ}) \)

Graph property(ies) \( \text{MIN}_{\text{NSCC}} = |\text{NODES}| \)

Graph model

Part (A) of Figure 5.597 shows the initial graph from which we start. It is derived from the set associated with each vertex. Each set describes the potential values of the \text{succ} attribute of a given vertex. Part (B) of Figure 5.597 gives the final graph associated with the \text{Example} slot. The \text{strongly_connected} constraint holds since the final graph contains one single strongly connected component mentioning every vertex of the initial graph.

![Figure 5.597: Initial and final graph of the \text{strongly_connected} set constraint](image)

Signature

Since the maximum number of vertices of the final graph is equal to \( |\text{NODES}| \) we can rewrite the graph property \( \text{MIN}_{\text{NSCC}} = |\text{NODES}| \) to \( \text{MIN}_{\text{NSCC}} \geq |\text{NODES}| \) and simplify \( \text{MIN}_{\text{NSCC}} \) to \( \text{MIN}_{\text{NSCC}} \).
### 5.346 subgraph_isomorphism

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>[258]</td>
</tr>
</tbody>
</table>

**Constraint**

```
subgraph_isomorphism(NODES_PATTERN, NODES_TARGET, FUNCTION)
```

**Arguments**

- **NODES_PATTERN** : `collection(index=int, succ=int)
- **NODES_TARGET** : `collection(index=int, succ=dvar)
- **FUNCTION** : `collection(image=dvar)

**Restrictions**

- `required(NODES_PATTERN, [index, succ])`
- `NODES_PATTERN.index ≥ 1`
- `NODES_PATTERN.index ≤ |NODES_PATTERN|`
- `distinct(NODES_PATTERN, index)`
- `NODES_PATTERN.succ ≥ 1`
- `NODES_PATTERN.succ ≤ |NODES_PATTERN|`
- `required(NODES_TARGET, [index, succ])`
- `NODES_TARGET.index ≥ 1`
- `NODES_TARGET.index ≤ |NODES_TARGET|`
- `distinct(NODES_TARGET, index)`
- `NODES_TARGET.succ ≥ 1`
- `NODES_TARGET.succ ≤ |NODES_TARGET|`
- `required(FUNCTION, [image])`
- `FUNCTION.image ≥ 1`
- `FUNCTION.image ≤ |NODES_TARGET|`
- `distinct(FUNCTION, image)`
- `|FUNCTION| = |NODES_PATTERN|`

**Purpose**

Given two directed graphs `PATTERN` and `TARGET` enforce a one to one correspondence, defined by the function `FUNCTION`, between the vertices of the graph `PATTERN` and the vertices of an induced subgraph of `TARGET` so that, if there is an arc from `u` to `v` in the graph `PATTERN`, then there is also an arc from the image of `u` to the image of `v` in the induced subgraph of `TARGET`. The vertices of both graphs are respectively defined by the two collections of vertices `NODES_PATTERN` and `NODES_TARGET`. Within collection `NODES_PATTERN` the set of successors of each node is fixed, while this is not the case for the collection `NODES_TARGET`. This stems from the fact that the `TARGET` graph is not fixed (i.e., the lower and upper bounds of the target graph are specified when we post the `subgraph_isomorphism` constraint, while the induced subgraph of a solution to the `subgraph_isomorphism` constraint corresponds to a graph for which the upper and lower bounds are identical).
Example

\[
\begin{align*}
\text{index} - 1 & \quad \text{succ} = \{2, 4\}, \\
\text{index} - 2 & \quad \text{succ} = \{1, 3, 4\}, \\
\text{index} - 3 & \quad \text{succ} = \emptyset, \\
\text{index} - 4 & \quad \text{succ} = \emptyset, \\
\text{index} - 1 & \quad \text{succ} = \emptyset, \\
\text{index} - 2 & \quad \text{succ} = \{3, 4, 5\}, \\
\text{index} - 3 & \quad \text{succ} = \emptyset, \\
\text{index} - 4 & \quad \text{succ} = \{2, 5\}, \\
\text{index} - 5 & \quad \text{succ} = \emptyset, \\
\{4, 2, 3, 5\}
\end{align*}
\]

Figure 5.59 gives the pattern (see Part (A)) and target graph (see Part (B)) of the Example slot as well as the one to one correspondence (see Part (C)) between the pattern graph and the induced subgraph of the target graph. The \textit{subgraph isomorphism} constraint since:

- To the arc from vertex 1 to vertex 4 in the pattern graph corresponds the arc from vertex 4 to 5 in the induced subgraph of the target graph.
- To the arc from vertex 1 to vertex 2 in the pattern graph corresponds the arc from vertex 4 to 2 in the induced subgraph of the target graph.
- To the arc from vertex 2 to vertex 1 in the pattern graph corresponds the arc from vertex 2 to 4 in the induced subgraph of the target graph.
- To the arc from vertex 2 to vertex 4 in the pattern graph corresponds the arc from vertex 2 to 5 in the induced subgraph of the target graph.
- To the arc from vertex 2 to vertex 3 in the pattern graph corresponds the arc from vertex 2 to 3 in the induced subgraph of the target graph.

**Typical**

<table>
<thead>
<tr>
<th>NODES_PATTERN</th>
<th>&gt; 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODES_TARGET</td>
<td>&gt; 1</td>
</tr>
</tbody>
</table>

**Symmetries**

- Items of NODES_PATTERN are \textit{permutable}.
- Items of NODES_TARGET are \textit{permutable}.

**Usage**

Within the context of constraint programming the constraint was used for finding symmetries [305, 307, 306].

**Algorithm**

[387, 321, 236, 419].

**See also**

related: \textit{graph isomorphism}.

**Keywords**

\textit{constraint arguments}: constraint involving set variables.
\textit{constraint type}: predefined constraint, graph constraint.
\textit{symmetry}: symmetry.
Figure 5.598: (A) The pattern graph, (B) the initial target graph – plain arcs must belong to the induced subgraph, while dotted arcs may or may not belong to the induced subgraph – and (C) the correspondence between the vertices of the pattern graph and the vertices of the induced subgraph of the target graph
5.347 sum

<table>
<thead>
<tr>
<th>Description</th>
<th>Links</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>[418].</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>(\text{sum}(\text{INDEX, SETS, CONSTANTS, } S))</td>
<td></td>
</tr>
<tr>
<td><strong>Synonym</strong></td>
<td>(\text{sum_pred.})</td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>INDEX : dvar</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SETS : (\text{collection}(\text{ind-int, set-sint}))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CONSTANTS : (\text{collection}(\text{cst-int}))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S : dvar</td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>(</td>
<td>\text{SETS}</td>
</tr>
<tr>
<td></td>
<td>(\text{distinct}(\text{SETS, ind}))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(</td>
<td>\text{CONSTANTS}</td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>(S) is equal to the sum of the constants of (\text{CONSTANTS}) corresponding to the (\text{INDEX}^{th}) set of the (\text{SETS}) collection.</td>
<td></td>
</tr>
</tbody>
</table>
| **Example** | \[
\begin{pmatrix}
\text{ind} - 8 & \text{set} = \{2, 3\}, \\
\text{ind} - 1 & \text{set} = \{3\}, \\
\text{ind} - 3 & \text{set} = \{1, 4, 5\}, \\
\text{ind} - 6 & \text{set} = \{2, 4\}, \\
\{4, 9, 1, 3, 1\}, 10
\end{pmatrix}
\] |       |
| **Typical** | \(|\text{SETS}| > 1\) \(|\text{CONSTANTS}| > |\text{SETS}|\) \(\text{range}(\text{CONSTANTS, cst}) > 1\) |       |
| **Symmetry** | Items of \(\text{SETS}\) are permutable. |       |
| **Arg. properties** | Functional dependency: \(S\) determined by \(\text{INDEX, SETS}\) and \(\text{CONSTANTS}\). |       |
| **Usage** | In his article introducing the \(\text{sum}\) constraint, Tallys H. Yunes mentions the \emph{Sequence Dependent Cumulative Cost Problem} as the subproblem that originally motivates this constraint. |       |
Remark

The sum constraint is called `sum_pred` in MiniZinc (http://www.g12.cs.mu.oz.au/minizinc/).

Algorithm

The article [418] gives the convex hull relaxation of the sum constraint.

Systems

`sum_pred` in MiniZinc.

See also

common keyword: element (data constraint), `sum_ctr`, `sum_set` (sum).

used in graph description: `in_set`.

Keywords

characteristic of a constraint: convex hull relaxation, sum.

constraint type: data constraint.

filtering: linear programming.

modelling: functional dependency.

Figure 5.599: Illustration of the correspondence between the arguments of the sum constraint in the context of the Example slot
According to the value assigned to INDEX the arc constraint selects for the final graph:

- The INDEX\textsuperscript{th} item of the SETS collection.
- The items of the CONSTANTS collection for which the key correspond to the indices of the INDEX\textsuperscript{th} set of the SETS collection.

Finally, since we use the SUM graph property on the \texttt{cst} attribute of the CONSTANTS collection, the last argument \texttt{S} of the \texttt{sum} constraint is equal to the sum of the constants associated with the vertices of the final graph.

Parts (A) and (B) of Figure 5.600 respectively show the initial and final graph associated with the \texttt{Example} slot. Since we use the SUM graph property we show the vertices from which we compute \texttt{S} in a box.
### 5.348 sum ctr

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
</table>

**Origin**

Arithmetic constraint.

**Constraint**

\[ \text{sum}_\text{ctr}(\text{VARIABLES}, \text{CTR}, \text{VAR}) \]

**Synonyms**

constant, \text{sum}, \text{linear}, \text{scalar product}.

**Arguments**

- \text{VARIABLES} : collection(var= \text{dvar})
- \text{CTR} : atom
- \text{VAR} : \text{dvar}

**Restrictions**

\[
\begin{align*}
\text{required}(\text{VARIABLES}, \text{var}) \\
\text{CTR} \in \{=, \neq, <, \geq, >, \leq\}
\end{align*}
\]

**Purpose**

Constraint the sum of a set of domain variables. More precisely, let \(S\) denote the sum of the variables of the \text{VARIABLES} collection (when the collection is empty the corresponding sum is equal to 0). Enforce the following constraint to hold: \(S \leq \text{CTR} \leq \text{VAR}\).

**Example**

\[(1, 1, 4), (=, 6)\]

The \text{sum}_\text{ctr} constraint holds since the condition \(1 + 1 + 4 = 6\) is satisfied.

**Typical**

\[
\begin{align*}
|\text{VARIABLES}| > 1 \\
\text{range}(\text{VARIABLES}.\text{var}) > 1 \\
\text{CTR} \in \{=, <, \geq, >, \leq\}
\end{align*}
\]

**Symmetry**

Items of \text{VARIABLES} are permutable.

**Arg. properties**

- **Contractible** wrt. \text{VARIABLES} when \text{CTR} \in \{<, \leq\} and \text{minval}(\text{VARIABLES}.\text{var}) \geq 0.
- **Contractible** wrt. \text{VARIABLES} when \text{CTR} \in \{\geq, >\} and \text{maxval}(\text{VARIABLES}.\text{var}) \leq 0.
- **Extensible** wrt. \text{VARIABLES} when \text{CTR} \in \{\geq, >\} and \text{minval}(\text{VARIABLES}.\text{var}) \geq 0.
- **Extensible** wrt. \text{VARIABLES} when \text{CTR} \in \{<, \leq\} and \text{maxval}(\text{VARIABLES}.\text{var}) \leq 0.
- **Aggregate**: \text{VARIABLES}(union), \text{CTR}(\text{id}, \text{VAR}(+)).

**Remark**

When \text{CTR} corresponds to = this constraint is referenced under the names \text{constant_sum} in \text{KOALOG} (http://www.koalog.com/php/index.php) and \text{sum} in \text{JaCoP} (http://www.jacop.eu/).
equation in Choco, linear in Gecode, scalar_product in SICStus.

Used in:
- bin_packing,
- cumulative,
- cumulative_convex,
- cumulative_with_level_of_priority,
- cumulatives,
- indexed_sum,
- interval_and_sum,
- relaxed_sliding_sum,
- sliding_sum,
- sliding_time_window_sum.

See also:
- assignment dimension added: interval_and_sum (assignment dimension corresponding to intervals is added).

Common keyword:
- arith_sliding (arithmetic constraint),
- increasing_sum (sum),
- product_ctr,
- range_ctr (arithmetic constraint),
- sum,
- sum_cubes_ctr (sum),
- sum_set (arithmetic constraint),
- sum_squares_ctr (sum).

Generalisation:
- scalar_product (arithmetic constraint) where all coefficients are not necessarily equal to 1.

System of constraints: sliding_sum.

Keywords:
- characteristic of a constraint: sum.
- constraint type: arithmetic constraint.
- heuristics: regret based heuristics, regret based heuristics in matrix problems.
Since we want to keep all the vertices of the initial graph we use the **SELF** arc generator together with the **TRUE** arc constraint. This predefined arc constraint always holds.

Parts (A) and (B) of Figure 5.601 respectively show the initial and final graph associated with the **Example** slot. Since we use the **TRUE** arc constraint both graphs are identical.

![Graph model](image)

Figure 5.601: Initial and final graph of the `sum,ctr` constraint
### 5.349 sum_cubes_ctr

**DESCRIPTION**

Arithmetic constraint.

**Constraint**

sum_cubes_ctr(VARIABLES, CTR, VAR)

**Synonyms**

sum_cubes, sum_of_cubes, sum_of_cubes_ctr.

**Arguments**

- VARIABLES: `collection(var−dvar)`
- CTR: `atom`
- VAR: `dvar`

**Restrictions**

- `required(VARIABLES.var)`
- `CTR ∈ [=, ≠, <, ≥, >, ≤]`

**Purpose**

Constraint the sum of the cubes of a set of domain variables. More precisely, let $S$ denote the sum of the cubes of the variables of the VARIABLES collection (when the collection is empty the corresponding sum is equal to 0). Enforce the following constraint to hold: $S < CTR < VAR$.

**Example**

$((1, 2, 2), =, 17)$

The `sum_cubes_ctr` constraint holds since the condition $1^3 + 2^3 + 2^3 = 17$ is satisfied.

**Typical**

- `|VARIABLES| > 1`
- `range(VARIABLES.var) > 1`
- `CTR ∈ [=, <, >, ≤]`

**Symmetry**

Items of VARIABLES are permutable.

**Arg. properties**

- **Contractible wrt.** VARIABLES when $CTR ∈ [<, ≤]$ and $\minval(VARIABLES.var) ≥ 0$.
- **Contractible wrt.** VARIABLES when $CTR ∈ [≥, >]$ and $\maxval(VARIABLES.var) ≤ 0$.
- **Extensible wrt.** VARIABLES when $CTR ∈ [≥, >]$ and $\minval(VARIABLES.var) ≥ 0$.
- **Extensible wrt.** VARIABLES when $CTR ∈ [<, ≤]$ and $\maxval(VARIABLES.var) ≤ 0$.
- **Aggregate:** VARIABLES(union), CTR(id), VAR(+).

**See also**

- common keyword: `sum, sum_squares_ctr (sum)`.

**Keywords**

- characteristic of a constraint: `sum`.
- constraint type: predefined constraint, arithmetic constraint.
### 5.350 sum_free

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>[403]</td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>sum_free(S)</td>
</tr>
<tr>
<td><strong>Argument</strong></td>
<td>S : svar</td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>Impose for all pairs of values (not necessarily distinct) i, j of the set S the fact that the sum i + j is not an element of S.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>( {1, 3, 5, 9} )</td>
</tr>
</tbody>
</table>

The sum_free\( \{1, 3, 5, 9\} \) constraint holds since:

- \( 1 + 1 = 2 \notin S \), \( 1 + 3 = 4 \notin S \), \( 1 + 5 = 6 \notin S \), \( 1 + 9 = 10 \notin S \).
- \( 3 + 3 = 6 \notin S \), \( 3 + 5 = 8 \notin S \), \( 3 + 9 = 12 \notin S \).
- \( 5 + 5 = 10 \notin S \), \( 5 + 9 = 14 \notin S \).

| **Usage** | The sum_free constraint was introduced by W.-J. van Hoeve and A. Sabharwal in order to model in a concise way Schur problems. |
| **Algorithm** | W.-J. van Hoeve and A. Sabharwal have proposed an algorithm that enforces bound-consistency for the sum_free constraint in [403]. |
| **Keywords** | constraint arguments: unary constraint, constraint involving set variables. constraint type: predefined constraint. filtering: bound-consistency. problems: Schur number. |
5.351 sum_of_increments

**DESCRIPTION**

Given a collection of variables `VARIABLES` which can only be assigned non-negative values, and a variable `LIMIT`, enforce the condition

\[
\sum_{i=2}^{|[VARIABLES]|} \max(VARIABLES[i].var - VARIABLES[i - 1].var, 0) \leq \text{LIMIT}.
\]

`VARIABLES[1].var` stands from the fact that we assume an additional implicit 0 before the first variable (i.e., `VARIABLES[1].var = \max(\text{VARIABLES[1].var} - 0, 0)`).

**Purpose**

The `sum_of_increments` constraint holds since we have that

\[
4 + \max(4 - 4, 0) + \max(3 - 4, 0) + \max(6 - 4, 0) \leq 7.
\]

**Example**

\[\langle 4, 4, 3, 4, 6 \rangle, 7\]

**Typical**

- `|VARIABLES| > 2`
- `range(VARIABLES.var) > 1`
- `maxval(VARIABLES.var) > 0`
- `LIMIT > 0`

**Symmetries**

- One and the same constant can be added to `VARIABLES.var` and to `LIMIT`.
- Items of `VARIABLES` can be reversed.
- `LIMIT` can be increased.

**Arg. properties**

- Prefix-contractible wrt. `VARIABLES`.
- Suffix-contractible wrt. `VARIABLES`.

**Usage**

The `sum_of_increments` was initially motivated by the problem of decomposing a matrix of non-negative integers into a positive linear combination of matrices consisting of only zeros and ones, where the ones occur consecutively in each row.

**Algorithm**

A \(O(|VARIABLES|)\) bound-consistency filtering algorithm for the `sum_of_increments` constraint is described in [81].
Reformulation  

The following reformulations are provided in [81]. Assuming $\text{VARIABLES}[0].\text{var}$ is defined as 0 (i.e., a zero is added before the first variable of the VARIABLES collection) we have:

- $\sum_{i=1}^{|	ext{VARIABLES}|} S_i \leq \text{LIMIT}$ with $D_i = \text{VARIABLES}[i].\text{var} - \text{VARIABLES}[i-1].\text{var}$ and $S_i = \max(D_i, 0) \ (1 \leq i \leq |	ext{VARIABLES}|)$.

- $\sum_{i=1}^{|	ext{VARIABLES}|} S_i \leq \text{LIMIT}$ with $\text{VARIABLES}[i].\text{var} - \text{VARIABLES}[i-1].\text{var} \leq S_i$ and $S_i \in [0, \text{LIMIT}] \ (1 \leq i \leq |	ext{VARIABLES}|)$.

Keywords  

- **characteristic of a constraint**: difference, sum.
- **constraint type**: predefined constraint.
- **filtering**: bound-consistency.
5.352  sum_of_weights_of_distinct_values

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[38]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>sum_of_weights_of_distinct_values(VARIABLES, VALUES, COST)</td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td>swdv.</td>
<td></td>
</tr>
</tbody>
</table>

Arguments

VARIABLES : collection(var−dvar)
VALUES    : collection(val−int, weight−int)
COST      : dvar

Restrictions

required(VARIABLES, var)
|VALUES| > 0
required(VVALUES, [val, weight])
VALUES.weight ≥ 0
distinct(VVALUES, val)
in_attr(VVALUES, var, VALUES, val)
COST ≥ 0

Purpose

All variables of the VARIABLES collection take a value in the VALUES collection. In addition COST is the sum of the weight attributes associated with the distinct values taken by the variables of VARIABLES.

Example

\[
\begin{pmatrix}
\langle 1, 6, 1 \rangle, \\
\langle val−1, weight−5, \\
\langle val−2, weight−3, \\
\langle val−6, weight−7, \\
12
\end{pmatrix}
\]

The sum_of_weights_of_distinct_values constraint holds since its last argument COST = 12 is equal to the sum 5 + 7 of the weights of the values 1 and 6 that occur within the \( \langle 1, 6, 1 \rangle \) collection.

Typical

| VARIABLES| > 1
range(VARIABLES.var) > 1
|VALUES| > 1
range(VVALUES.weight) > 1
VALUES.weight > 0

Symmetries

- Items of VARIABLES are permissible.
- All occurrences of two distinct values of VARIABLES.var can be swapped.
- Items of VALUES are permissible.
- All occurrences of two distinct values in VARIABLES.var or VALUES.val can be swapped; all occurrences of a value in VARIABLES.var or VALUES.val can be renamed to any unused value.
Arg. properties

Functional dependency: COST determined by VARIABLES and VALUES.

See also

attached to cost variant: nvalue (all values have a weight of 1).
common keyword: global_cardinality_with_costs, minimum_weight_alldifferent, weighted_partial_alldiff (weighted assignment).

Keywords

application area: assignment.
constraint arguments: pure functional dependency.
constraint type: relaxation.
filtering: cost filtering constraint.
modelling: functional dependency.
problems: domination, weighted assignment, facilities location problem.
Since we use the `PRODUCT` arc generator, the number of sources of the final graph cannot exceed the number of sources of the initial graph. Since the initial graph contains `|VARIABLES|` sources, this number is an upper bound of the number of sources of the final graph. Therefore we can rewrite `NSOURCE = |VARIABLES|` to `NSOURCE ≥ |VARIABLES|` and simplify `NSOURCE` to `NSOURCE`.

Parts (A) and (B) of Figure 5.602 respectively show the initial and final graph associated with the Example slot. Since we use the `NSOURCE` graph property, the source vertices of the final graph are shown in a double circle. Since we also use the `SUM` graph property we show the vertices from which we compute the total cost in a box.

<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td><code>PRODUCT ↦ collection(variables, values)</code></td>
</tr>
<tr>
<td>Arc arity</td>
<td>2</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td><code>variables.var = values.val</code></td>
</tr>
<tr>
<td>Graph property(ies)</td>
<td>• <code>NSOURCE</code> =</td>
</tr>
<tr>
<td></td>
<td>• <code>SUM</code>(</td>
</tr>
</tbody>
</table>

**Signature**

Figure 5.602: Initial and final graph of the `sum_of_weights_of_distinct_values` constraint
5.353 sum_set

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>H. Cambazard</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>sum_set(SV, VALUES, CTR, VAR)</td>
<td></td>
</tr>
</tbody>
</table>
| Arguments   | SV : svar  
VALUES : collection(val–int, coef–int)  
CTR : atom  
VAR : dvar |
| Restrictions| required(V VALUES, [val, coef])  
distinct(V VALUES, val)  
VALUES.coef ≥ 0  
CTR ∈ [=, ≠, <, ≥, >, ≤] |
| Purpose     | Let SUM denote the sum of the coef attributes of the VALUES collection for which the corresponding values val occur in the set SV. Enforce the following constraint to hold: SUM CTR VAR. |
| Example     | \( \{2, 3, 6\}, \begin{array}{c} val - 2 \text{ coef} - 7, \\ val - 9 \text{ coef} - 1, \\ val - 5 \text{ coef} - 7, \\ val - 6 \text{ coef} - 2 \end{array}, =, 9 \) |
| Typical     | \[ |VALUES| > 1 \]
VALUES.coef > 0  
CTR ∈ [=, <, ≥, >, ≤] |
| Symmetry    | Items of VALUES are permutable. |
| Systems     | weights in Gecode. |
| See also    | common keyword: sum, sum.ctr(sum). |
| Keywords    | characteristic of a constraint: sum.  
constraint arguments: binary constraint, constraint involving set variables.  
constraint type: arithmetic constraint. |
Graph model

Parts (A) and (B) of Figure 5.603 respectively show the initial and final graph associated with the Example slot.

Figure 5.603: Initial and final graph of the sum_set constraint
5.354  sum_squares_ctr

**DESCRIPTION**

**Origin**
Arithmetic constraint.

**Constraint**

\[
\text{sum_squares ctr}(\text{VARm}, \text{CTR}, \text{VAR})
\]

**Synonyms**

\[
\text{sum_squares}, \text{sum_of_squares}, \text{sum_of_squares ctr}.
\]

**Arguments**

- **VARIABLES**: \text{collection}(\text{var}−\text{dvar})
- **CTR**: \text{atom}
- **VAR**: \text{dvar}

**Restrictions**

\[
\text{required}(\text{VARm}.\text{var})
\]

\[
\text{CTR} \in \{=, \neq, <, \geq, >, \leq\}
\]

**Purpose**

Constraint the sum of the squares of a set of domain variables. More precisely, let \( S \) denote the sum of the squares of the variables of the \text{VARIABLES} collection (when the collection is empty the corresponding sum is equal to 0). Enforce the following constraint to hold: \( S \text{ CTR } \text{VAR} \).

**Example**

\[
((1, 1, 4), =, 18)
\]

The \text{sum_squares_ctr} constraint holds since the condition \( 1^2 + 1^2 + 4^2 = 18 \) is satisfied.

**Typical**

\[
\begin{align*}
|\text{VARIABLES}| & > 1 \\
\text{range}(\text{VARIABLES}.\text{var}) & > 1 \\
\text{CTR} & \in \{=, <, \geq, >, \leq\}
\end{align*}
\]

**Symmetry**

Items of \text{VARIABLES} are permutable.

**Arg. properties**

- **Contractible** wrt. \text{VARIABLES} when \( \text{CTR} \in [<, \leq] \).
- **Extensible** wrt. \text{VARIABLES} when \( \text{CTR} \in [\geq, >] \).
- **Aggregate**: \text{VARIABLES}(union), \text{CTR}(id), \text{VAR}(+) .

**See also**

common keyword: \text{sum_ctr}, \text{sum_cubes_ctr}(\text{sum}).

**Keywords**

characteristic of a constraint: sum.

constraint type: predefined constraint, arithmetic constraint.
### 5.355 symmetric

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td></td>
<td>[131]</td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>symmetric(NODES)</td>
<td></td>
</tr>
<tr>
<td><strong>Argument</strong></td>
<td>NODES : collection(index=int, succ=svar)</td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td>required(NODES,[index, succ])</td>
<td>NODES.index ≥ 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NODES.index ≤</td>
</tr>
<tr>
<td></td>
<td></td>
<td>distinct(NODES, index)</td>
</tr>
</tbody>
</table>

**Purpose**

Consider a digraph $G$ described by the NODES collection. Select a subset of arcs of $G$ so that the corresponding graph is symmetric (i.e., if there is an arc from $i$ to $j$, there is also an arc from $j$ to $i$).

**Example**

$$
\begin{pmatrix}
\text{index - 1} & \text{succ} = \{1, 2, 3\}, \\
\text{index - 2} & \text{succ} = \{1, 3\}, \\
\text{index - 3} & \text{succ} = \{1, 2\}, \\
\text{index - 4} & \text{succ} = \{5, 6\}, \\
\text{index - 5} & \text{succ} = \{4\}, \\
\text{index - 6} & \text{succ} = \{4\}
\end{pmatrix}
$$

The symmetric constraint holds since the NODES collection depicts a symmetric graph.

**Typical**

$|\text{NODES}| > 2$

**Symmetry**

Items of NODES are permutable.

**Algorithm**

The filtering algorithm for the symmetric constraint is given in [131, page 87]. It removes (respectively imposes) the arcs $(i, j)$ for which the arc $(j, i)$ is not present (respectively is present). It has an overall complexity of $O(n + m)$ where $n$ and $m$ respectively denote the number of vertices and the number of arcs of the initial graph.

**See also**

- common keyword: connected(symmetric).
- used in graph description: in_set.

**Keywords**

- constraint arguments: constraint involving set variables.
- constraint type: graph constraint.
- final graph structure: symmetric.
Graph model

Part (A) of Figure 5.604 shows the initial graph from which we start. It is derived from the set associated with each vertex. Each set describes the potential values of the \texttt{succ} attribute of a given vertex. Part (B) of Figure 5.604 gives the final graph associated with the \textbf{Example} slot.

Figure 5.604: Initial and final graph of the \texttt{symmetric} set constraint
5.356 symmetric_alldifferent

DESCRIPTION | LINKS | GRAPH
---|---|---

**Origin**

[325]

**Constraint**

\texttt{symmetric\_alldifferent(NODES)}

**Synonyms**

\texttt{symmetric\_alldiff, symmetric\_alldistinct, symm\_alldifferent, symm\_alldiff, symm\_alldistinct, one\_factor, two\_cycle.}

**Argument**

\texttt{NODES : collection(index\_int, succ\_dvar)}

**Restrictions**

\| \begin{align*}
|\text{NODES}| \mod 2 &= 0 \\
\text{required} & (\text{NODES}, [\text{index}, \text{succ}]) \\
\text{NODES}.\text{index} & \geq 1 \\
\text{NODES}.\text{index} & \leq |\text{NODES}| \\
\text{distinct} & (\text{NODES}, \text{index}) \\
\text{NODES}.\text{succ} & \geq 1 \\
\text{NODES}.\text{succ} & \leq |\text{NODES}|
\end{align*}
\| -- (1)

**Purpose**

All variables associated with the \text{succ} attribute of the \text{NODES} collection should be pair-wise distinct. In addition enforce the following condition: if variable \text{NODES}[i].\text{succ} takes value \text{j} then variable \text{NODES}[j].\text{succ} takes value \text{i}. This can be interpreted as a graph-covering problem where one has to cover a digraph \text{G} with circuits of length two in such a way that each vertex of \text{G} belongs to one single circuit.

**Example**

\[
\begin{pmatrix}
\text{index}−1 & \text{succ}−3, \\
\text{index}−2 & \text{succ}−4, \\
\text{index}−3 & \text{succ}−1, \\
\text{index}−4 & \text{succ}−2
\end{pmatrix}
\]

The \text{symmetric\_alldifferent} constraint holds since:

\begin{itemize}
  \item \text{NODES}[1].\text{succ} = 3 \iff \text{NODES}[3].\text{succ} = 1,
  \item \text{NODES}[2].\text{succ} = 4 \iff \text{NODES}[4].\text{succ} = 2.
\end{itemize}

**Typical**

\text{\| |\text{NODES}| \geq 4 |}

**Symmetry**

Items of \text{NODES} are \text{permutable}.

**Usage**

As it was reported in [325, page 420], this constraint is useful to express matches between persons or between teams. The \text{symmetric\_alldifferent} constraint also appears implicitly in the \text{cycle cover problem} and corresponds to the four conditions given in section 1 \text{Modeling the Cycle Cover Problem} of [288].
Remark

This constraint is referenced under the name one_factor in [194] as well as in [384]. From a modelling point of view this constraint can be expressed with the cycle constraint [39] where one imposes the additional condition that each cycle has only two nodes.

Algorithm

A filtering algorithm for the symmetric_alldifferent constraint was proposed by J.-C. Régis in [325]. It achieves arc-consistency and its running time is dominated by the complexity of finding all edges that do not belong to any maximum cardinality matching in an undirected n-vertex, m-edge graph, i.e., $O(m \cdot n)$.

Reformulation

The symmetric_alldifferent(NODES) constraint can be expressed in term of a conjunction of $|\text{NODES}|^2$ reified constraints of the form NODES[i].succ = NODES[j].succ = i $(1 \leq i, j \leq |\text{NODES}|)$. The symmetric_alldifferent constraint can also be reformulated as an inverse constraint as shown below:

$$
\text{symmetric\_alldifferent}\left(\begin{array}{c}
\text{index} - 1 & \text{succ} - s_1, \\
\text{index} - 2 & \text{succ} - s_2, \\
\vdots & \vdots \\
\text{index} - n & \text{succ} - s_n \\
\end{array}\right)
$$

$$
\text{inverse}\left(\begin{array}{c}
\text{index} - 1 & \text{succ} - s_1, \text{pred} - s_1, \\
\text{index} - 2 & \text{succ} - s_2, \text{pred} - s_2, \\
\vdots & \vdots \\
\text{index} - n & \text{succ} - s_n, \text{pred} - s_n \\
\end{array}\right)
$$

See also

- common keyword: alldifferent, cycle, inverse (permutation).
- implies: symmetric_alldifferent, except_0.
- implies (items to collection): lex_alldifferent.
- related: roots.

Keywords

- application area: sport timetabling.
- characteristic of a constraint: all different, disequality.
- combinatorial object: permutation, matching.
- constraint type: graph constraint, timetabling constraint, graph partitioning constraint.
- filtering: arc-consistency.
- final graph structure: circuit.
- modelling: cycle.
Arc input(s): NODES
Arc generator: $\text{CLIQUE}(\neq) \rightsquigarrow \text{collection}(\text{nodes1}, \text{nodes2})$
Arc arity: 2
Arc constraint(s):
- $\text{nodes1}.\text{succ} = \text{nodes2}.\text{index}$
- $\text{nodes2}.\text{succ} = \text{nodes1}.\text{index}$
Graph property(ies): $\text{NARC} = |\text{NODES}|$

Graph model

In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices.

Parts (A) and (B) of Figure 5.605 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the arcs of the final graph are stressed in bold.

![Graph](image)

Figure 5.605: Initial and final graph of the symmetric_alldifferent constraint

Signature

Since all the index attributes of the NODES collection are distinct, and because of the first condition $\text{nodes1}.\text{succ} = \text{nodes2}.\text{index}$ of the arc constraint, each vertex of the final graph has at most one successor. Therefore the maximum number of arcs of the final graph is equal to the maximum number of vertices $|\text{NODES}|$ of the final graph. So we can rewrite $\text{NARC} = |\text{NODES}|$ to $\text{NARC} \geq |\text{NODES}|$ and simplify $\text{NARC}$ to $\text{NARC}$.
5.357 symmetric_alldifferent_except_0

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from symmetric_alldifferent</td>
</tr>
<tr>
<td>Constraint</td>
<td>symmetric_alldifferent_except_0(NODES)</td>
</tr>
<tr>
<td>Synonyms</td>
<td>symmetric_alldiff_except_0, symm_alldifferent_except_0, symm_alldistinct_except_0, symm_alldiff_except_0, symm_alldistinct_except_0, symm_alldistinct_except_0</td>
</tr>
<tr>
<td>Argument</td>
<td>NODES : collection(index−int, succ−dvar)</td>
</tr>
</tbody>
</table>
| Restrictions| required(NODES,[index, succ])
NODSI. index ≥ 1
NODSI. index ≤ |NODES|
distinct(NODES,index)
NODSI. succ ≥ 0
NODSI. succ ≤ |NODES|

Enforce the following three conditions:
1. ∀i ∈ [1,|NODES|], ∀j ∈ [1,|NODES|], (j ≠ i): NODES[i].succ = 0 ∨ NODES[j].succ = 0 ∨ NODES[i].succ ≠ NODES[j].succ.
2. ∀i ∈ [1,|NODES|] : NODES[i].succ ≠ i.
3. NODES[i].succ = j ∧ j ≠ i ∧ j ≠ 0 ⇔ NODES[j].succ = i ∧ i ≠ j ∧ i ≠ 0.

Example

\[
\begin{pmatrix}
\text{index} - 1 & \text{succ} - 3,
\text{index} - 2 & \text{succ} - 0,
\text{index} - 3 & \text{succ} - 4,
\text{index} - 4 & \text{succ} - 0
\end{pmatrix}
\]

The symmetric_alldifferent_except_0 constraint holds since:
- NODES[1].succ = 3 ⇔ NODES[3].succ = 1,
- NODES[2].succ = 0 and value 2 is not assigned to any variable.
- NODES[4].succ = 0 and value 4 is not assigned to any variable.

Given 3 successor variables that have to be assigned a value in interval [0, 3], the solutions of the symmetric_alldifferent_except_0((index−1 succ−s1, index−2 succ−s2, index−3 succ−s3)) constraint are (1, 0, 2, 0, 3), (1, 0, 2, 3, 3), (1, 2, 2, 1, 3, 0) and (1, 3, 2, 0, 3, 4).

Given 4 successor variables that have to be assigned a value in interval [0, 3], the solutions of the symmetric_alldifferent_except_0((index−1 succ−s1, index−2 succ−s2, index−3 succ−s3, index−4 succ−s4)) constraint are (1, 0, 2, 0, 3, 0, 4, 0), (1, 0, 2, 3, 4, 4, 3), (1, 0, 2, 3, 3, 2, 4, 0), (1, 0, 2, 4, 3, 0, 4, 2), (1, 2, 2, 1, 3, 0, 4, 0), (1, 2, 2, 1, 3, 4, 3), (1, 3, 2, 0, 3, 1, 4, 0), (1, 3, 2, 4, 3, 1, 4, 2), (1, 4, 2, 0, 3, 0, 4, 1), (1, 4, 2, 3, 3, 2, 4, 1).
Typical

\[ |\text{NODES}| \geq 4 \]
\[ \text{minval}(\text{NODES}.\text{succ}) = 0 \]

Symmetry

Items of \text{NODES} are permutable.

Usage

Within the context of sport scheduling, \( \text{NODES}[i].\text{succ} = j \) \( (i \neq 0, j \neq 0, i \neq j) \) is interpreted as the fact that team \( i \) plays against team \( j \), while \( \text{NODES}[i].\text{succ} = 0 \) \( (i \neq 0) \) is interpreted as the fact that team \( i \) does not play at all.

See also

implied by: symmetric\_alldifferent.

Keywords

application area: sport timetabling.
characteristic of a constraint: joker value.
constraint type: predefined constraint, timetabling constraint.
5.358  **symmetric_cardinality**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from global_cardinality by W. Kocjan.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>symmetric_cardinality(VARS, VALS)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VARS : collection(idvar=int, var=svar, l=int, u=int)</td>
<td>VALS : collection(idval=int, val=svar, l=int, u=int)</td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(VARS, [idvar, var, l, u])</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>Put in relation two sets: for each element of one set gives the corresponding elements of the other set to which it is associated. In addition, it constraints the number of elements associated with each element to be in a given interval.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>idvar</th>
<th>var</th>
<th>val</th>
<th>l</th>
<th>u</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>-3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>-4</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The symmetric_cardinality constraint holds since:

- 3 ∈ VARS[1].var ⇔ 1 ∈ VALS[3].val,
- 1 ∈ VARS[2].var ⇔ 2 ∈ VALS[1].val,
- 1 ∈ VARS[3].var ⇔ 3 ∈ VALS[1].val,
- 2 ∈ VARS[3].var ⇔ 3 ∈ VALS[2].val,
- $1 \in \text{VARS}[4].\text{var} \Leftrightarrow 4 \in \text{VALS}[1].\text{val},$
- $3 \in \text{VARS}[4].\text{var} \Leftrightarrow 4 \in \text{VALS}[3].\text{val},$
- The number of elements of $\text{VARS}[1].\text{var} = \{3\}$ belongs to interval $[0, 1],$
- The number of elements of $\text{VARS}[2].\text{var} = \{1\}$ belongs to interval $[1, 2],$
- The number of elements of $\text{VARS}[3].\text{var} = \{1, 2\}$ belongs to interval $[1, 2],$
- The number of elements of $\text{VARS}[4].\text{var} = \{1, 3\}$ belongs to interval $[2, 3],$
- The number of elements of $\text{VALS}[1].\text{val} = \{2, 3, 4\}$ belongs to interval $[3, 4],$
- The number of elements of $\text{VALS}[2].\text{val} = \{3\}$ belongs to interval $[1, 1],$
- The number of elements of $\text{VALS}[3].\text{val} = \{1, 4\}$ belongs to interval $[1, 2],$
- The number of elements of $\text{VALS}[4].\text{val} = \emptyset$ belongs to interval $[0, 1].$

**Typical**

| $|\text{VARS}| > 1$
| $|\text{VALS}| > 1$

**Symmetries**

- Items of $\text{VARS}$ are **permutable**.
- Items of $\text{VALS}$ are **permutable**.

**Usage**

The most simple example of applying `symmetric gcc` is a variant of personnel assignment problem, where one person can be assigned to perform between $n$ and $m$ ($n \leq m$) jobs, and every job requires between $p$ and $q$ ($p \leq q$) persons. In addition every job requires different kind of skills. The previous problem can be modelled as follows:

- For each person we create an item of the $\text{VARS}$ collection,
- For each job we create an item of the $\text{VALS}$ collection,
- There is an arc between a person and the particular job if this person is qualified to perform it.

**Remark**

The `symmetric gcc` constraint generalises the `global cardinality` constraint by allowing a variable to take more than one value.

**Algorithm**

A flow-based arc-consistency algorithm for the `symmetric cardinality` constraint is described in [223].

**See also**

- **common keyword**: `link set to booleans` (`constraint involving set variables`).
- **generalisation**: `symmetric gcc` (`fixed interval replaced by variable`).
- **root concept**: `global cardinality`.
- **used in graph description**: `in set`.

**Keywords**

- **application area**: assignment.
- **combinatorial object**: relation.
- **constraint arguments**: constraint involving set variables.
- **constraint type**: decomposition, timetabling constraint.
- **filtering**: flow.
Arc input(s)  

\textit{VARS VALS}

Arc generator  

\textit{PRODUCT} \rightarrow \textit{collection}(\text{vars, vals})

Arc arity  

2

Arc constraint(s)  

\begin{itemize}
  \item \text{in}_\text{set}(\text{vars.idvar, vals.val}) \leftrightarrow \text{in}_\text{set}(\text{vals.idval, vars.var})
  \item \text{vars.} l \leq \text{card}_\text{set}(\text{vars.var})
  \item \text{vars.} u \geq \text{card}_\text{set}(\text{vars.var})
  \item \text{vals.} l \leq \text{card}_\text{set}(\text{vals.val})
  \item \text{vals.} u \geq \text{card}_\text{set}(\text{vals.val})
\end{itemize}

Graph property(ies)  

\textbf{NARC} = |\text{VARS}| \times |\text{VALS}|

Graph model  

The graph model used for the \textit{symmetric_cardinality} is similar to the one used in the \textit{domain_constraint} or in the \textit{link_set_to_booleans} constraints: we use an equivalence in the arc constraint and ask all arc constraints to hold.

Parts (A) and (B) of Figure 5.606 respectively show the initial and final graph associated with the \textbf{Example} slot. Since we use the \textbf{NARC} graph property, all the arcs of the final graph are stressed in bold.

\begin{figure}[h]
  \centering
  \includegraphics[width=\textwidth]{figure.png}
  \caption{Initial and final graph of the \textit{symmetric_cardinality} constraint}
\end{figure}

Signature  

Since we use the \textit{PRODUCT} arc generator on the collections VARS and VALS, the number of arcs of the initial graph is equal to |VARS| \times |VALS|. Therefore the maximum number of arcs of the final graph is also equal to |VARS| \times |VALS| and we can rewrite \textbf{NARC} = |VARS| \times |VALS| to \textbf{NARC} \geq |VARS| \times |VALS|. So we can simplify \textbf{NARC} to \textbf{NARC}. 


5.359 symmetric_gcc

**DESCRIPTION**

- **Origin**: Derived from *global_cardinality* by W. Kocjan.

- **Constraint**: `symmetric_gcc(VARS, VALS)`

- **Synonym**: `sgcc`.

**Arguments**

- **VARS**: `collection(idvar: int, var: svar, nocc: dvar)`
- **VALS**: `collection(idval: int, val: svar, nocc: dvar)`

**Restrictions**

- `required(VARS, [idvar, var, nocc])`
  - `|VARS| ≥ 1`
  - `VARS.idvar ≥ 1`
  - `VARS.idvar ≤ |VARS|`
  - `distinct(VARS, idvar)`
  - `VARS.nocc ≥ 0`
  - `VARS.nocc ≤ |VALS|`
  - `required(VALS, [idval, val, nocc])`
  - `|VALS| ≥ 1`
  - `VALS.idval ≥ 1`
  - `VALS.idval ≤ |VALS|`
  - `distinct(VALS, idval)`
  - `VALS.nocc ≥ 0`
  - `VALS.nocc ≤ |VARS|`

**Purpose**

Put in relation two sets: for each element of one set gives the corresponding elements of the other set to which it is associated. In addition, enforce a cardinality constraint on the number of occurrences of each value.

**Example**

```
\[
\begin{pmatrix}
  \text{idvar} = 1 & \text{var} = \{3\} & \text{nocc} = 1, \\
  \text{idvar} = 2 & \text{var} = \{1\} & \text{nocc} = 1, \\
  \text{idvar} = 3 & \text{var} = \{1, 2\} & \text{nocc} = 2, \\
  \text{idvar} = 4 & \text{var} = \{1, 3\} & \text{nocc} = 2, \\
  \text{idval} = 1 & \text{val} = \{2, 3, 4\} & \text{nocc} = 3, \\
  \text{idval} = 2 & \text{val} = \{3\} & \text{nocc} = 1, \\
  \text{idval} = 3 & \text{val} = \{1, 4\} & \text{nocc} = 2, \\
  \text{idval} = 4 & \text{val} = \emptyset & \text{nocc} = 0
\end{pmatrix}
\]
```

The `symmetric_gcc` constraint holds since:

- `3 ∈ VARS[1].var ⇔ 1 ∈ VALS[3].val`,
- `1 ∈ VARS[2].var ⇔ 2 ∈ VALS[1].val`,
- `1 ∈ VARS[3].var ⇔ 3 ∈ VALS[1].val`,
- `2 ∈ VARS[3].var ⇔ 3 ∈ VALS[2].val`,
- etc.
• $1 \in \text{VARS}_4[.\text{var} \leftrightarrow 4 \in \text{VALS}_1[.\text{val}]$,
• $3 \in \text{VARS}_4[.\text{var} \leftrightarrow 4 \in \text{VALS}_3[.\text{val}]$,
• The number of elements of $\text{VARS}_1[.\text{var} = \{3\}$ is equal to 1,
• The number of elements of $\text{VARS}_2[.\text{var} = \{1\}$ is equal to 1,
• The number of elements of $\text{VARS}_3[.\text{var} = \{1, 2\}$ is equal to 2,
• The number of elements of $\text{VARS}_4[.\text{var} = \{1, 3\}$ is equal to 2,
• The number of elements of $\text{VALS}_1[.\text{val} = \{2, 3, 4\}$ is equal to 3,
• The number of elements of $\text{VALS}_2[.\text{val} = \{3\}$ is equal to 1,
• The number of elements of $\text{VALS}_3[.\text{val} = \{1, 4\}$ is equal to 2,
• The number of elements of $\text{VALS}_4[.\text{val} = \emptyset$ is equal to 0.

<table>
<thead>
<tr>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
</tr>
<tr>
<td>$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symmetries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items of $\text{VARS}$ are permutable.</td>
</tr>
<tr>
<td>Items of $\text{VALS}$ are permutable.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>The most simple example of applying \textit{symmetric_gcc} is a variant of personnel assignment problem, where one person can be assigned to perform between $n$ and $m$ ($n \leq m$) jobs, and every job requires between $p$ and $q$ ($p \leq q$) persons. In addition every job requires different kind of skills. The previous problem can be modelled as follows:</td>
</tr>
<tr>
<td>For each person we create an item of the $\text{VARS}$ collection,</td>
</tr>
<tr>
<td>For each job we create an item of the $\text{VALS}$ collection,</td>
</tr>
<tr>
<td>There is an arc between a person and the particular job if this person is qualified to perform it.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>The \textit{symmetric_gcc} constraint generalises the \textit{global_cardinality} constraint by allowing a variable to take more than one value. It corresponds to a variant of the $\textit{symmetric_cardinality}$ constraint described in [223] where the occurrence variables of the $\text{VARS}$ and $\text{VALS}$ collections are replaced by fixed intervals.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>See also</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{common keyword:} link_set_to_booleans (\textit{constraint involving set variables}).</td>
</tr>
<tr>
<td>\textbf{root concept:} \textit{global_cardinality}.</td>
</tr>
<tr>
<td>\textbf{specialisation:} \textit{symmetric_cardinality} (\textit{variable replaced by fixed interval}).</td>
</tr>
<tr>
<td>\textbf{used in graph description:} in_set.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{application area:} assignment.</td>
</tr>
<tr>
<td>\textbf{combinatorial object:} relation.</td>
</tr>
<tr>
<td>\textbf{constraint arguments:} constraint involving set variables.</td>
</tr>
<tr>
<td>\textbf{constraint type:} decomposition, timetabling constraint.</td>
</tr>
<tr>
<td>\textbf{filtering:} flow.</td>
</tr>
</tbody>
</table>
Arc input(s)  
VARS VALS

Arc generator  
\( PRODUCT \mapsto collection(vars, vals) \)

Arc arity  
2

Arc constraint(s)  
- \( in\_set(idvar, vals) \leftrightarrow in\_set(idval, var) \)
- \( \text{var.nocc} = \text{card}\_set(var) \)
- \( \text{vals.nocc} = \text{card}\_set(vals) \)

Graph property(ies)  
\[ \text{NARC} = |VARS| \times |VALS| \]

Graph model  
The graph model used for the symmetric gcc is similar to the one used in the domain constraint or in the link set to booleans constraints: we use an equivalence in the arc constraint and ask all arc constraints to hold.

Parts (A) and (B) of Figure 5.607 respectively show the initial and final graph. Since we use the NARC graph property, all the arcs of the final graph are stressed in bold.

![Graph Model](image)

Figure 5.607: Initial and final graph of the symmetric gcc constraint

Signature  
Since we use the \( PRODUCT \) arc generator on the collections VARS and VALS, the number of arcs of the initial graph is equal to \( |VARS| \times |VALS| \). Therefore the maximum number of arcs of the final graph is also equal to \( |VARS| \times |VALS| \) and we can rewrite \( \text{NARC} = |VARS| \times |VALS| \) to \( \text{NARC} \geq |VARS| \times |VALS| \). So we can simplify \( \text{NARC} \) to \( \text{NARC} \).
**5.360 temporal_path**

**DESCRIPTION**

**Constraint**

```
temporal_path(NPATH, NODES)
```

**Arguments**

- `NPATH : dvar`
- `NODES : collection(index−int, succ−dvar, start−dvar, end−dvar)`

**Restrictions**

- `NPATH ≥ 1`
- `NPATH ≤ |NODES|`
- `required(NODES, [index, succ, start, end])`
- `|NODES| > 0`
- `NODES.index ≥ 1`
- `NODES.index ≤ |NODES|`
- `distinct(NODES, index)`
- `NODES.succ ≥ 1`
- `NODES.succ ≤ |NODES|`
- `NODES.start ≤ NODES.end`

**Purpose**

Let $G$ be the digraph described by the `NODES` collection. Partition $G$ with a set of disjoint paths such that each vertex of the graph belongs to a single path. In addition, for all pairs of consecutive vertices of a path we have a precedence constraint that enforces the end associated with the first vertex to be less than or equal to the start related to the second vertex.

**Example**

```
\[
\begin{pmatrix}
\text{index} - 1 & \text{succ} - 2 & \text{start} - 0 & \text{end} - 1,
\text{index} - 2 & \text{succ} - 6 & \text{start} - 3 & \text{end} - 5,
\text{index} - 3 & \text{succ} - 4 & \text{start} - 0 & \text{end} - 3,
\text{index} - 4 & \text{succ} - 5 & \text{start} - 4 & \text{end} - 6,
\text{index} - 5 & \text{succ} - 7 & \text{start} - 7 & \text{end} - 8,
\text{index} - 6 & \text{succ} - 6 & \text{start} - 7 & \text{end} - 9,
\text{index} - 7 & \text{succ} - 7 & \text{start} - 9 & \text{end} - 10
\end{pmatrix}
\]
```

The `temporal_path` constraint holds since:

- The items of the `NODES` collection represent the two (`NPATH = 2`) paths $1 \rightarrow 2 \rightarrow 6$ and $3 \rightarrow 4 \rightarrow 5 \rightarrow 7$.
- As illustrated by Figure 5.608, all precedences between adjacent vertices of a same path hold: each item $i$ ($1 \leq i \leq 7$) of the `NODES` collection is represented by a rectangle starting and ending at instants `NODES[i].start` and `NODES[i].end`; the number within each rectangle designates the index of the corresponding item within the `NODES` collection.

**Typical**

```
NPATH < |NODES|
|NODES| > 1
NODES.start < NODES.end
```
**Symmetries**

- Items of NODES are permutable.
- One and the same constant can be added to the start and end attributes of all items of NODES.

**Arg. properties**

Functional dependency: NPATH determined by NODES.

**Remark**

This constraint is related to the path constraint of Ilog Solver. It can also be directly expressed with the cycle [39] constraint of CHIP by using the diff nodes and the origin parameters. A generic model based on linear programming that handles paths, trees and cycles is presented in [226].

**Reformulation**

The temporal path(NPATH, NODES) constraint can be expressed in term of a conjunction of one path constraint, |NODES| element constraints, and |NODES| inequalities constraints:

- We pass to the path constraint the number of path variable NPATH as well as the items of the NODES collection form which we remove the start and end attributes.
- To the i-th (1 ≤ i ≤ |NODES|) item of the NODES collection, we create a variable Startsucc and an element(NODES[i].succ, ⟨T_{i,1}, T_{i,2}, ..., T_{i,|NODES|}, Startsucc⟩) constraint, where T_{i,j} = NODES[i].start if i ≠ j and T_{i,i} = NODES[i].end otherwise.
- Finally to the i-th (1 ≤ i ≤ |NODES|) item of the NODES collection, we also create an inequality constraint NODES[i].end ≤ Startsucc. Note that, since T_{i,i} was initialised to NODES[i].end, the inequality NODES[i].end ≤ T_{i,i} holds when i = j.

With respect to the Example slot we get the following conjunction of constraints:

```
predpath(2, ⟨index − 1 succ − 2, index − 2 succ − 6, index − 3 succ − 4, index − 4 succ − 5, index − 5 succ − 7, index − 6 succ − 6, index − 7 succ − 7⟩),
```

```
element(2, ⟨1, 3, 0, 4, 7, 7, 9, 3⟩),
element(6, ⟨1, 5, 0, 4, 7, 7, 9, 7⟩),
element(4, ⟨1, 5, 3, 4, 7, 7, 9, 4⟩),
element(5, ⟨1, 5, 3, 6, 7, 7, 9, 7⟩),
element(7, ⟨1, 5, 3, 6, 8, 7, 9, 9⟩),
element(6, ⟨1, 5, 3, 6, 8, 9, 9, 9⟩),
element(7, ⟨1, 5, 3, 6, 8, 9, 10, 10⟩),
```

```
1 ≤ 3, 5 ≤ 7, 3 ≤ 4, 6 ≤ 7, 8 ≤ 9, 9 ≤ 9, 10 ≤ 10.
```

Figure 5.608: The two paths of the Example slot represented as two sequences of rectangles
See also

common keyword: path\_from\_to (path).
implies (items to collection): atleast\_nvector.
specialisation: path (time dimension removed).

Keywords

combinatorial object: path.
constraint type: graph constraint, graph partitioning constraint.
final graph structure: connected component.
modelling: sequence dependent set-up, functional dependency.
modelling exercises: sequence dependent set-up.
Arc input(s) NODES
Arc generator $CLIQUE \rightarrow collection(\text{nodes}_1, \text{nodes}_2)$
Arc arity 2
Arc constraint(s)
- $\text{nodes}_1.\text{succ} = \text{nodes}_2.\text{index}$
- $\text{nodes}_1.\text{succ} = \text{nodes}_1.\text{index} \lor \text{nodes}_1.\text{end} \leq \text{nodes}_2.\text{start}$
- $\text{nodes}_1.\text{start} \leq \text{nodes}_1.\text{end}$
- $\text{nodes}_2.\text{start} \leq \text{nodes}_2.\text{end}$
Graph property(ies)
- $\text{MAX}\_\text{ID} \leq 1$
- $\text{NCC} = \text{NPATH}$
- $\text{NVERTEX} = |\text{NODES}|$

Graph model
The arc constraint is a conjunction of four conditions that respectively correspond to:
- A constraint that links the successor variable of a first vertex to the index attribute of a second vertex,
- A precedence constraint that applies on one vertex and its distinct successor,
- One precedence constraint between the start and the end of the vertex that corresponds to the departure of an arc,
- One precedence constraint between the start and the end of the vertex that corresponds to the arrival of an arc.

We use the following three graph properties in order to enforce the partitioning of the graph in distinct paths:
- The first property $\text{MAX}\_\text{ID} \leq 1$ enforces that each vertex has no more than one predecessor ($\text{MAX}\_\text{ID}$ does not consider loops),
- The second property $\text{NCC} = \text{NPATH}$ ensures that we have the required number of paths,
- The third property $\text{NVERTEX} = |\text{NODES}|$ enforces that, for each vertex, the start is not located after the end.

Parts (A) and (B) of Figure 5.609 respectively show the initial and final graph associated with the Example slot. Since we use the $\text{MAX}\_\text{ID}$, the $\text{NCC}$ and the $\text{NVERTEX}$ graph properties we display the following information on the final graph:
- We show with a double circle a vertex that has the maximum number of predecessors.
- We show the two connected components corresponding to the two paths.
- We put in bold the vertices.
Figure 5.609: Initial and final graph of the temporal path constraint
5.361 tour

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[4]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>tour(NODES)</td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>atour, cycle.</td>
<td></td>
</tr>
<tr>
<td>Argument</td>
<td>NODES : collection(index=int, succ=svar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>Enforce to cover an undirected graph G described by the NODES collection with a Hamiltonian cycle.</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
|             | \(\begin{pmatrix}
\text{index} - 1 & \text{succ} = \{2, 4\}, \\
\text{index} - 2 & \text{succ} = \{1, 3\}, \\
\text{index} - 3 & \text{succ} = \{2, 4\}, \\
\text{index} - 4 & \text{succ} = \{1, 3\}
\end{pmatrix}\) | |
|             | The tour constraint holds since its NODES argument depicts the following Hamiltonian cycle visiting successively the vertices 1, 2, 3 and 4. | |
| Symmetry    | Items of NODES are permutable. | |
| Algorithm   | When the number of vertices is odd (i.e., \(|\text{NODES}| \) is odd) a necessary condition is to have a bipartite graph (see the Algorithm slot of the bipartite constraint). | |
| See also    | common keyword: circuit (graph partitioning constraint, Hamiltonian), cycle (graph constraint), link_set_to_booleans (constraint involving set variables). | |
| Keywords    | characteristic of a constraint: undirected graph. constraint arguments: constraint involving set variables. constraint type: graph constraint. filtering: linear programming. problems: Hamiltonian. | |
Arc input(s) NODES
Arc generator $\text{CLIQUE}(\neq) \mapsto \text{collection}(\text{nodes1}, \text{nodes2})$
Arc arity 2
Arc constraint(s) $\text{in}_{\text{set}}(\text{nodes2}.\text{index}, \text{nodes1}.\text{succ}) \iff \text{in}_{\text{set}}(\text{nodes1}.\text{index}, \text{nodes2}.\text{succ})$

Graph property(ies) $\text{NARC} = |\text{NODES}| \ast |\text{NODES}| - |\text{NODES}|$

Arc input(s) NODES
Arc generator $\text{CLIQUE}(\neq) \mapsto \text{collection}(\text{nodes1}, \text{nodes2})$
Arc arity 2
Arc constraint(s) $\text{in}_{\text{set}}(\text{nodes2}.\text{index}, \text{nodes1}.\text{succ})$

Graph property(ies)
- $\text{MIN_NSCC} = |\text{NODES}|$
- $\text{MIN_ID} = 2$
- $\text{MAX_ID} = 2$
- $\text{MIN_OD} = 2$
- $\text{MAX_OD} = 2$

Graph model
The first graph property enforces the subsequent condition: If we have an arc from the $i^{th}$ vertex to the $j^{th}$ vertex then we have also an arc from the $j^{th}$ vertex to the $i^{th}$ vertex. The second graph property enforces the following constraints:

- We have one strongly connected component containing $|\text{NODES}|$ vertices,
- Each vertex has exactly two predecessors and two successors.

Part (A) of Figure 5.610 shows the initial graph from which we start. It is derived from the set associated with each vertex. Each set describes the potential values of the $\text{succ}$ attribute of a given vertex. Part (B) of Figure 5.610 gives the final graph associated with the Example slot. The tour constraint holds since the final graph corresponds to a Hamiltonian cycle.

Signature
Since the maximum number of vertices of the final graph is equal to $|\text{NODES}|$, we can rewrite the graph property $\text{MIN_NSCC} = |\text{NODES}|$ to $\text{MIN_NSCC} \geq |\text{NODES}|$ and simplify $\text{MIN_NSCC}$ to $\text{MIN_NSCC}$. 
Figure 5.610: Initial and final graph of the tour set constraint
5.362 track

DESCRIPTION | LINKS | GRAPH
--- | --- | ---

**Origin**

[255]

**Constraint**

track\((\text{NTRAIL, TASKS})\)

**Arguments**

\(\text{NTRAIL} : \text{int}\)
\(\text{TASKS} : \text{collection(trail-int, origin-dvar, end-dvar)}\)

**Restrictions**

\(\text{NTRAIL} > 0\)
\(\text{NTRAIL} \leq |\text{TASKS}|\)
\(\text{required(\text{TASKS, [trail, origin, end]})}\)
\(\text{TASKS.origin} \leq \text{TASKS.end}\)

**Purpose**

The track constraint enforces that, at each point in time overlapped by at least one task, the number of distinct values of the trail attribute of the set of tasks that overlap that point, is equal to NTRAIL.

**Example**

\[
\begin{pmatrix}
\text{trail-1, origin-1, end-2, } \\
\text{trail-2, origin-1, end-2, } \\
2, \\
\text{trail-1, origin-2, end-4, } \\
\text{trail-2, origin-2, end-3, } \\
\text{trail-2, origin-3, end-4, }
\end{pmatrix}
\]

Figure 5.611 represents the tasks of the example: to the \(i^{th}\) task of the TASKS collection corresponds a rectangle labelled by \(i\). The track constraint holds since:

- The first and second tasks both overlap instant 1 and have a respective trail of 1 and 2. This makes two distinct values for the trail attribute at instant 1.
- The third and fourth tasks both overlap instant 2 and have a respective trail of 1 and 2. This makes two distinct values for the trail attribute at instant 2.
- The third and fifth tasks both overlap instant 3 and have a respective trail of 1 and 2. This makes two distinct values for the trail attribute at instant 3.

Figure 5.611: Tasks associated with the example of the Example slot
Typical

\[ NTRAIL < \lvert TASKS \rvert \]
\[ \lvert TASKS \rvert > 1 \]
\[ range(TASKS.trail) > 1 \]
\[ TASKS.origin < TASKS.end \]

Symmetries

- Items of TASKS are permutable.
- All occurrences of two distinct values of TASKS.trail can be swapped; all occurrences of a value of TASKS.trail can be renamed to any unused value.
- One and the same constant can be added to the origin and end attributes of all items of TASKS.

Reformulation

The track constraint can be expressed in terms of a set of refined constraints and of \( 2 \cdot \lvert TASKS \rvert \) nvalue constraints:

1. For each pair of tasks TASKS\([i], TASKS\([j]\) (\(i, j \in [1, \lvert TASKS \rvert]\)) of the TASKS collection we create a variable \( T_{ij}^{\text{origin}} \) which is set to the trail attribute of task TASKS\([j]\) if task TASKS\([j]\) overlaps the origin attribute of task TASKS\([i]\), and to the trail attribute of task TASKS\([i]\) otherwise:
   - If \( i = j \):
     - \( T_{ij}^{\text{origin}} = \text{TASKS}[i].\text{trail} \).
   - If \( i \neq j \):
     - \( T_{ij}^{\text{origin}} = \text{TASKS}[i].\text{trail} \lor T_{ij}^{\text{origin}} = \text{TASKS}[j].\text{trail} \).
     - \( ((\text{TASKS}[j].\text{origin} \leq \text{TASKS}[i].\text{origin} \land \text{TASKS}[j].\text{end} > \text{TASKS}[i].\text{origin}) \land (T_{ij}^{\text{origin}} = \text{TASKS}[j].\text{trail}) \lor ((\text{TASKS}[j].\text{origin} > \text{TASKS}[i].\text{origin} \lor \text{TASKS}[j].\text{end} \leq \text{TASKS}[i].\text{origin}) \land (T_{ij}^{\text{origin}} = \text{TASKS}[i].\text{trail})) \)

2. For each task TASKS\([i]\) (\(i \in [1, \lvert TASKS \rvert]\)) we impose the number of distinct trails associated with the tasks that overlap the origin of task TASKS\([i]\) (TASKS\([i]\) overlaps its own origin) to be equal to NTRAIL:
   \[ \text{nvalue}(NTRAIL, \langle T_{i1}^{\text{origin}}, T_{i2}^{\text{origin}}, \ldots, T_{i\lvert TASKS \rvert}^{\text{origin}} \rangle). \]

3. For each pair of tasks TASKS\([i], TASKS\([j]\) (\(i, j \in [1, \lvert TASKS \rvert]\)) of the TASKS collection we create a variable \( T_{ij}^{\text{end}} \) which is set to the trail attribute of task TASKS\([j]\) if task TASKS\([j]\) overlaps the end attribute of task TASKS\([i]\), and to the trail attribute of task TASKS\([i]\) otherwise:
   - If \( i = j \):
     - \( T_{ij}^{\text{end}} = \text{TASKS}[i].\text{trail} \).
   - If \( i \neq j \):
     - \( T_{ij}^{\text{end}} = \text{TASKS}[i].\text{trail} \lor T_{ij}^{\text{end}} = \text{TASKS}[j].\text{trail} \).
     - \( ((\text{TASKS}[j].\text{origin} \leq \text{TASKS}[i].\text{end} - 1 \land \text{TASKS}[j].\text{end} > \text{TASKS}[i].\text{end} - 1) \land (T_{ij}^{\text{end}} = \text{TASKS}[j].\text{trail}) \lor ((\text{TASKS}[j].\text{origin} > \text{TASKS}[i].\text{end} - 1 \lor \text{TASKS}[j].\text{end} \leq \text{TASKS}[i].\text{end} - 1) \land (T_{ij}^{\text{end}} = \text{TASKS}[i].\text{trail})) \)

4. For each task TASKS\([i]\) (\(i \in [1, \lvert TASKS \rvert]\)) we impose the number of distinct trails associated with the tasks that overlap the end of task TASKS\([i]\) (TASKS\([i]\) overlaps its own end) to be equal to NTRAIL:
   \[ \text{nvalue}(NTRAIL, \langle T_{i1}^{\text{end}}, T_{i2}^{\text{end}}, \ldots, T_{i\lvert TASKS \rvert}^{\text{end}} \rangle). \]
With respect to the **Example** slot we get the following conjunction of *nvalue* constraints:

- The *nvalue*(2, ⟨1, 2, 1, 1, 1⟩) constraint corresponding to the *trail* attributes of the tasks that overlap the origin of the first task (i.e., instant 1) that has a trail of 1.
- The *nvalue*(2, ⟨1, 2, 2, 2, 2⟩) constraint corresponding to the *trail* attributes of the tasks that overlap the origin of the second task (i.e., instant 1) that has a trail of 2.
- The *nvalue*(2, ⟨1, 1, 2, 1⟩) constraint corresponding to the *trail* attributes of the tasks that overlap the origin of the third task (i.e., instant 2) that has a trail of 1.
- The *nvalue*(2, ⟨2, 1, 2, 2, 2⟩) constraint corresponding to the *trail* attributes of the tasks that overlap the origin of the fourth task (i.e., instant 2) that has a trail of 1.
- The *nvalue*(2, ⟨1, 1, 1, 1⟩) constraint corresponding to the *trail* attributes of the tasks that overlap the last instant of the first task (i.e., instant 1) that has a trail of 1.
- The *nvalue*(2, ⟨1, 2, 2, 2, 2⟩) constraint corresponding to the *trail* attributes of the tasks that overlap the last instant of the second task (i.e., instant 1) that has a trail of 2.
- The *nvalue*(2, ⟨1, 1, 1, 1⟩) constraint corresponding to the *trail* attributes of the tasks that overlap the last instant of the third task (i.e., instant 3) that has a trail of 1.
- The *nvalue*(2, ⟨2, 2, 1, 2, 2⟩) constraint corresponding to the *trail* attributes of the tasks that overlap the last instant of the fourth task (i.e., instant 2) that has a trail of 2.
- The *nvalue*(2, ⟨2, 2, 1, 2, 2⟩) constraint corresponding to the *trail* attributes of the tasks that overlap the last instant of the fifth task (i.e., instant 3) that has a trail of 2.

**See also**

- **common keyword:** coloured_cumulative (*resource constraint*).
- **used in graph description:** *nvalue*.

**Keywords**

- **characteristic of a constraint:** derived collection.
- **constraint type:** timetabling constraint, resource constraint, temporal constraint.
Derived Collection

\[
\text{col} \left( \begin{array}{c}
\text{item} (\text{origin} \rightarrow \text{TASKS}.\text{origin}, \\
\text{end} \rightarrow \text{TASKS}.\text{end}, \\
\text{point} \rightarrow \text{TASKS}.\text{origin} \\
\text{item} (\text{origin} \rightarrow \text{TASKS}.\text{origin}, \\
\text{end} \rightarrow \text{TASKS}.\text{end}, \\
\text{point} \rightarrow \text{TASKS}.\text{end} - 1)
\end{array} \right)
\]

\[
\text{TIME_POINTS} \rightarrow \text{collection} (\text{origin} \rightarrow \text{dvar}, \text{end} \rightarrow \text{dvar}, \text{point} \rightarrow \text{dvar}),
\]

Arc input(s) \hspace{0.5cm} \text{TASKS}

Arc generator \hspace{0.5cm} \text{SELF} \rightarrow \text{collection} (\text{tasks})

Arc arity \hspace{0.5cm} 1

Arc constraint(s) \hspace{0.5cm} \text{tasks}.\text{origin} \leq \text{tasks}.\text{end}

Graph property(ies) \hspace{0.5cm} \text{NARC} = |\text{TASKS}|

Arc input(s) \hspace{0.5cm} \text{TIME_POINTS} \text{TASKS}

Arc generator \hspace{0.5cm} \text{PRODUCT} \rightarrow \text{collection} (\text{time_points}, \text{tasks})

Arc arity \hspace{0.5cm} 2

Arc constraint(s) \hspace{0.5cm}
- \text{time_points}.\text{end} > \text{time_points}.\text{origin}
- \text{tasks}.\text{origin} \leq \text{time_points}.\text{point}
- \text{time_points}.\text{point} < \text{tasks}.\text{end}

Sets

\[
\text{SUCC} \mapsto \\
\left[ \begin{array}{c}
\text{source}, \\
\text{variables} \rightarrow \text{col} \left( \begin{array}{c}
\text{VARIES} \rightarrow \text{collection} (\text{var} \rightarrow \text{dvar}), \\
\text{item} (\text{var} \rightarrow \text{TASKS}.\text{trail})
\end{array} \right)
\end{array} \right]
\]

Constraint(s) on sets \hspace{0.5cm} \text{nvalue(NTRAIL,variables)}

Graph model

Parts (A) and (B) of Figure 5.612 respectively show the initial and final graph of the second graph constraint of the Example slot.

Signature

Consider the first graph constraint. Since we use the \text{SELF} arc generator on the \text{TASKS} collection, the maximum number of arcs of the final graph is equal to |\text{TASKS}|. Therefore we can rewrite \text{NARC} = |\text{TASKS}| to \text{NARC} \geq |\text{TASKS}| and simplify \text{NARC} to \text{NARC}. 
Figure 5.612: Initial and final graph of the track constraint
5.363 tree

**DESCRIPTION**

<table>
<thead>
<tr>
<th>Origin</th>
<th>N. Beldiceanu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint</td>
<td>tree(NTREES, NODES)</td>
</tr>
<tr>
<td>Arguments</td>
<td></td>
</tr>
</tbody>
</table>
NTREES : dvar  
NODES : collection(index−int, succ−dvar) |
| Restrictions | 
NTREES ≥ 1  
NTREES ≤ |NODES|  
required(NODES,[index, succ])  
NODES.index ≥ 1  
NODES.index ≤ |NODES|  
distinct(NODES, index)  
NODES.succ ≥ 1  
NODES.succ ≤ |NODES| |
| Purpose | Cover a digraph G by a set of trees in such a way that each vertex of G belongs to one distinct tree. The edges of the trees are directed from their leaves to their respective roots. |
| Example | 
\[
\begin{pmatrix}
\text{index} - 1 & \text{succ} - 1, \\
\text{index} - 2 & \text{succ} - 5, \\
\text{index} - 3 & \text{succ} - 5, \\
\text{index} - 4 & \text{succ} - 7, \\
\text{index} - 5 & \text{succ} - 1, \\
\text{index} - 6 & \text{succ} - 1, \\
\text{index} - 7 & \text{succ} - 7, \\
\text{index} - 8 & \text{succ} - 5
\end{pmatrix}
\]

The tree constraint holds since the graph associated with the items of the NODES collection corresponds to two trees (i.e., NTREES = 2): each tree respectively involves the vertices \{1, 2, 3, 5, 6, 8\} and \{4, 7\}. They are depicted by Figure 5.613. |

**Links**

**Graph**

![Graph](image)

Figure 5.613: The two trees associated with the example

**Typical**

| NTREES < |NODES|  
| | | |
| NODES| > 2 |
Symmetry

Items of NODES are permutable.

Arg. properties

Functional dependency: NTREES determined by NODES.

Remark

Given a complete digraph of \( n \) vertices as well as an unrestricted number of trees NTREES, the total number of solutions of the corresponding tree constraint corresponds to the sequence A000272 of the On-Line Encyclopedia of Integer Sequences [370].

Extension of the tree constraint to the minimum spanning tree constraint is described in [132, 329, 332].

Algorithm

An arc-consistency filtering algorithm for the tree constraint is described in [40]. This algorithm is based on a necessary and sufficient condition that we now depict.

To any tree constraint we associate the digraph \( G = (V, E) \), where:

- To each item NODES[i] of the NODES collection corresponds a vertex \( v_i \) of \( G \).
- For every pair of items (NODES[i], NODES[j]) of the NODES collection, where \( i \) and \( j \) are not necessarily distinct, there is an arc from \( v_i \) to \( v_j \) in \( E \) if \( j \) is a potential value of NODES[i].succ.

A strongly connected component \( C \) of \( G \) is called a sink component if all the successors of all vertices of \( C \) belong to \( C \). Let MINTREES and MAXTREES respectively denote the number of sink components of \( G \) and the number of vertices of \( G \) with a loop.

The tree constraint has a solution if and only if:

- Each sink component of \( G \) contains at least one vertex with a loop,
- The domain of NTREES has at least one value within interval [MINTREES, MAXTREES].

Inspired by the idea of using dominators used in [205] for getting a linear time algorithm for computing strong articulation points of a digraph \( G \), the worst case complexity of the algorithm proposed in [40] was also enhanced in a similar way by J.-G. Fages and X. Lorca [144].

Reformulation

The tree constraint can be expressed in term of (1) a set of \( |\text{NODES}|^2 \) reified constraints for avoiding circuit between more than one node and of (2) \( |\text{NODES}| \) reified constraints and of one sum constraint for counting the trees:

1. For each vertex NODES[i] (\( i \in [1,|\text{NODES}|] \)) of the NODES collection we create a variable \( R_i \) that takes its value within interval [1,|NODES|]. This variable represents the rank of vertex NODES[i] within a solution. It is used to prevent the creation of circuit involving more than one vertex as explained now. For each pair of vertices NODES[i], NODES[j] (\( i, j \in [1,|\text{NODES}|] \)) of the NODES collection we create a reified constraint of the form NODES[i].succ = NODES[j].index \& i \neq j \Rightarrow R_i < R_j.

The purpose of this constraint is to express the fact that, if there is an arc from vertex NODES[i] to another vertex NODES[j], then \( R_i \) should be strictly less than \( R_j \).

2. For each vertex NODES[i] (\( i \in [1,|\text{NODES}|] \)) of the NODES collection we create a 0-1 variable \( B_i \) and state the following reified constraint NODES[i].succ = NODES[i].index \iff B_i in order to force variable \( B_i \) to be set to value 1 if and only if there is a loop on vertex NODES[i]. Finally we create a constraint NTREES = \( B_1 + B_2 + \ldots + B_{|\text{NODES}|} \) for stating the fact that the number of trees is equal to the number of loops of the graph.
 Systems

tree in Choco.

See also

common keyword: cycle, graph_crossing, map (graph partitioning constraint), proper_forest (connected component, tree).

implied by: binary_tree.

implies (items to collection): atleast_nvector.

related: balance_tree (counting number of trees versus controlling how balanced the trees are), global_cardinality_low_up_no_loop, global_cardinality_no_loop (can be used for restricting number of children since discard loops associated with tree roots).

shift of concept: stable_compatibility, tree_range, tree_resource.

specialisation: binary_tree (no limit on the number of children replaced by at most two children), path (no limit on the number of children replaced by at most one child).

uses in its reformulation: tree_range, tree_resource.

Keywords

constraint type: graph constraint, graph partitioning constraint.

filtering: strong articulation point, arc-consistency.

final graph structure: connected component, tree, one_succe.

modelling: functional dependency.
Arc input(s)  
NODES

Arc generator  
$\text{CLIQUE} \mapsto \text{collection}(\text{nodes}_1, \text{nodes}_2)$

Arc arity  
2

Arc constraint(s)  
$\text{nodes}_1.\text{succ} = \text{nodes}_2.\text{index}$

Graph property(ies)  
- $\text{MAX}\_\text{NSCC} \leq 1$
- $\text{NCC} = \text{NTREES}$

Graph model  
We use the graph property $\text{MAX}\_\text{NSCC} \leq 1$ in order to specify the fact that the size of the largest strongly connected component should not exceed one. In fact each root of a tree is a strongly connected component with one single vertex. The second graph property $\text{NCC} = \text{NTREES}$ enforces the number of trees to be equal to the number of connected components.

Parts (A) and (B) of Figure 5.614 respectively show the initial and final graph associated with the Example slot. Since we use the NCC graph property, we display the two connected components of the final graph. Each of them corresponds to a tree. The tree constraint holds since all strongly connected components of the final graph have no more than one vertex and since $\text{NTREES} = \text{NCC} = 2$.

![Diagrams](A) ![Diagrams](B)

Figure 5.614: Initial and final graph of the tree constraint
### 5.364 tree_range

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from <code>tree</code>.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td><code>tree_range(NTREES, R, NODES)</code></td>
<td></td>
</tr>
</tbody>
</table>
| Arguments | `NTREES : dvar`
| | `R : dvar`
| | `NODES : collection(index−int, succ−dvar)` |
| Restrictions | `NTREES ≥ 0`
| | `R ≥ 0`
| | `R < |NODES|`
| | `|NODES| > 0`
| | `required(NODES, [index, succ])`
| | `NODES.index ≥ 1`
| | `NODES.index ≤ |NODES|`
| | `distinct(NODES, index)`
| | `NODES.succ ≥ 1`
| | `NODES.succ ≤ |NODES|` |
| Purpose | Cover the digraph $G$ described by the `NODES` collection with `NTREES` trees in such a way that each vertex of $G$ belongs to one distinct tree. $R$ is the difference between the longest and the shortest paths (from a leaf to a root) of the final graph. |

#### Example

```
\begin{align*}
&\{\text{index} - 1, \text{succ} - 1, \\
&\text{index} - 2, \text{succ} - 5, \\
&\text{index} - 3, \text{succ} - 5, \\
&\text{index} - 4, \text{succ} - 7, \\
&\text{index} - 5, \text{succ} - 1, \\
&\text{index} - 6, \text{succ} - 1, \\
&\text{index} - 7, \text{succ} - 7, \\
&\text{index} - 8, \text{succ} - 5\}
\end{align*}
```

The `tree_range` constraint holds since the graph associated with the items of the `NODES` collection corresponds to two trees (i.e., `NTREES = 2`): each tree respectively involves the vertices `{1, 2, 3, 5, 8}` and `{4, 7}`. Furthermore $R = 1$ is set to the difference between the longest path (for instance $2 \rightarrow 5 \rightarrow 1$) and the shortest path (for instance $4 \rightarrow 7$) from a leaf to a root. Figure 5.615 provides the two trees associated with the example.

| Typical | `NTREES < |NODES|`
| | `|NODES| > 2` |

| Symmetry | Items of `NODES` are `permutable`. |
Arg. properties

- Functional dependency: \( \text{NTREES} \) determined by \( \text{NODES} \).
- Functional dependency: \( \text{R} \) determined by \( \text{NODES} \).

Reformulation

By introducing a distance variable \( D_i \), an occurrence variable \( O_i \) and a leave variable \( L_i \) (\( 1 \leq i \leq |\text{NODES}| \)) for each item \( i \) of the \( \text{NODES} \) collection, where:

- \( D_i \) represents the number of vertices from \( i \) to the root of the corresponding tree,
- \( O_i \) gives the number of occurrences of value \( i \) within variables \( \text{NODES}[1].\text{succ}, \text{NODES}[2].\text{succ}, \ldots, \text{NODES}[n].\text{succ} \),
- \( L_i \) is set to 1 if item \( i \) corresponds to a leave (i.e., \( O_i > 0 \)) and 0 otherwise,

the \text{tree_range}(\text{NTREES}, \text{R}, \text{NODES}) constraint can be expressed in term of a conjunction of one \text{tree} constraint, \( |\text{NODES}| \) \text{element} constraints, \( |\text{NODES}| \) \text{linear} constraints, one \text{global_cardinality} constraint, \( |\text{NODES}| \) \text{reified} constraints, one \text{open_minimum}, one \text{maximum} and one \text{linear} constraint, where:

- The \text{tree} constraint models the fact that we have a forest of \text{NTREES} trees.
- Each \text{element} constraint provides the link between the attribute \text{succ} of the \( i \)-th item and the distance variable \( D_{\text{NODES}[i].\text{succ}} \) associated with item \( \text{NODES}[i].\text{succ} \).
- Each linear constraint associated with the \( i \)-th item states that the difference between the distance variable \( D_i \) and the distance variable \( D_{\text{NODES}[i].\text{succ}} \) is equal to 1.
- The \text{global_cardinality} constraint provides the number of occurrences \( O_i \) of value \( i \) (\( 1 \leq i \leq |\text{NODES}| \)) within variables \( \text{NODES}[1].\text{succ}, \text{NODES}[2].\text{succ}, \ldots, \text{NODES}[|\text{NODES}|].\text{succ} \). Note that, when \( O_i \) is equal to 0, the corresponding \( i \)-th item is a leave of one of the \text{NTREES} trees.
- Each reified constraint of the form \( L_i \Leftrightarrow O_i > 0 \) makes the link between the \( i \)-th occurrence variable \( O_i \) and the \( i \)-th leave variable \( L_i \).
- The \text{open_minimum} constraint computes the minimum distance \text{MIN} from the leaves to the corresponding roots. The leave variable \( L_i \) is used in order to select only the distance variables corresponding to leaves.
- The \text{maximum} constraint computes the maximum distance \text{MAX} from the vertices to the roots. Since the maximum is achieved by a leaf we do not need to focus just on the leaves as it was the case for the minimum distance \text{MIN}.
- The linear constraint \( \text{MAX} - \text{MIN} = \text{R} \) links together argument \( \text{R} \) to the minimum and maximum distances.

With respect to the \textbf{Example} slot we get the following conjunction of constraints:

\begin{align*}
\text{tree}(2, \{(index - 1 \text{ succ} - 1, index - 2 \text{ succ} - 5),} \\
\quad \{(index - 3 \text{ succ} - 5, index - 4 \text{ succ} - 7),} \\
\end{align*}

\begin{figure}[h]
\centering
\begin{tikzpicture}
\filldraw (0,0) circle (1pt) node[above] {1} -- (1,0) node[above] {7} -- (0,0) -- (1,0) -- (2,0) node[below] {2} -- (1,0) -- (3,0) node[below] {3} -- (1,0) -- (5,0) node[below] {5} -- (4,0) node[below] {4} -- (1,0) -- (6,0) node[below] {6} -- (1,0) -- (7,0) node[below] {8} -- (1,0) -- (0,0);
\end{tikzpicture}
\caption{The two trees associated with the example}
\end{figure}
index – 5 succ – 1, index – 6 succ – 1,  
index – 7 succ – 7, index – 8 succ – 5),  
\( \text{domain}(D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8), 0, 8) \),  
\( \text{DS}_1 \in [0, 8] \), \( \text{element}(1, (0, D_2, D_3, D_4, D_5, D_6, D_7, D_8), DS_1) \), \( D_1 = 0 = 1 \),  
\( \text{DS}_2 \in [0, 8] \), \( \text{element}(5, (1, 0, D_3, D_4, D_5, D_6, D_7, D_8), DS_2) \), \( D_2 = D_3 = 1 \),  
\( \text{DS}_3 \in [0, 8] \), \( \text{element}(5, (1, D_2, 0, D_4, D_5, D_6, D_7, D_8), DS_3) \), \( D_3 = D_4 = 1 \),  
\( \text{DS}_4 \in [0, 8] \), \( \text{element}(7, (1, D_2, D_3, 0, D_6, D_7, D_8), DS_4) \), \( D_4 = D_7 = 1 \),  
\( \text{DS}_5 \in [0, 8] \), \( \text{element}(1, (1, D_2, D_3, D_4, 0, D_6, D_7, D_8), DS_5) \), \( D_5 = 1 = 1 \),  
\( \text{DS}_6 \in [0, 8] \), \( \text{element}(1, (1, 3, 3, D_4, 2, 0, D_7, D_8), DS_6) \), \( D_6 = 1 = 1 \),  
\( \text{DS}_7 \in [0, 8] \), \( \text{element}(7, (1, 3, 3, D_4, 2, 2, 0, D_8), DS_7) \), \( D_7 = 0 = 1 \),  
\( \text{DS}_8 \in [0, 8] \), \( \text{element}(5, (1, 3, 3, 2, 2, 1, 0), DS_8) \), \( D_8 = 2 = 1 \),  
\( \text{global_cardinality}((1, 5, 5, 7, 1, 1, 7, 5), (\text{val} – 1 \text{noccurrence} = 3,  
\text{val} – 2 \text{noccurrence} = 0,  
\text{val} – 3 \text{noccurrence} = 0,  
\text{val} – 4 \text{noccurrence} = 0,  
\text{val} – 5 \text{noccurrence} = 3,  
\text{val} – 6 \text{noccurrence} = 0,  
\text{val} – 7 \text{noccurrence} = 2,  
\text{val} – 8 \text{noccurrence} = 0)),  
(1 \leftrightarrow 3 > 0, 0 \leftrightarrow 0 > 0, 0 \leftrightarrow 0 > 0, 0 \leftrightarrow 0 > 0,  
1 \leftrightarrow 3 > 0, 0 \leftrightarrow 0 > 0, 1 \leftrightarrow 2 > 0, 0 \leftrightarrow 0 > 0,  
\text{open_minimum}(\text{MIN}, \langle \text{var} – 3 \text{bool} = 1, \text{var} – 0 \text{bool} = 0,  
\text{var} – 0 \text{bool} = 0, \text{var} – 0 \text{bool} = 0,  
\text{var} – 3 \text{bool} = 1, \text{var} – 0 \text{bool} = 0,  
\text{var} – 2 \text{bool} = 1, \text{var} – 0 \text{bool} = 0 \rangle),  
\text{maximum}(\text{MAX}, (1, 3, 3, 2, 2, 1, 3)),  
\text{MAX} – \text{MIN} = R = 1.  
\)
Arc input(s) NODES
Arc generator $CLIQUE\rightarrow\{(\text{nodes}_1, \text{nodes}_2)\}$
Arc arity 2
Arc constraint(s) $\text{nodes}_1.\text{succ} = \text{nodes}_2.\text{index}$
Graph property(ies) • $\text{MAX}_\text{NSCC} \leq 1$
• $\text{NCC} = \text{NTREES}$
• $\text{RANGE}_\text{DRG} = R$

Graph model

Parts (A) and (B) of Figure 5.616 respectively show the initial and final graph associated with the Example slot. Since we use the RANGE_DRG graph property, we respectively display the longest and shortest paths of the final graph with a bold and a dash line.

Figure 5.616: Initial and final graph of the tree_range constraint
5.365  tree_resource

**DESCRIPTION**

Origin
Derived from tree.

Constraint

\[ \text{tree_resource} (\text{RESOURCE}, \text{TASK}) \]

Arguments

\[
\begin{align*}
\text{RESOURCE} & : \text{collection}(\text{id} - \text{int}, \text{nb} \_ \text{task} - \text{dvar}) \\
\text{TASK} & : \text{collection}(\text{id} - \text{int}, \text{father} - \text{dvar}, \text{resource} - \text{dvar})
\end{align*}
\]

Restrictions

\[
\begin{align*}
|\text{RESOURCE}| > 0 & \\
\text{required} (\text{RESOURCE}, [\text{id}, \text{nb} \_ \text{task}]) & \\
\text{RESOURCE}.\text{id} \geq 1 & \\
\text{RESOURCE}.\text{id} \leq |\text{RESOURCE}| & \\
\text{distinct} (\text{RESOURCE}, \text{id}) & \\
\text{RESOURCE}.\text{nb} \_ \text{task} \geq 0 & \\
\text{RESOURCE}.\text{nb} \_ \text{task} \leq |\text{TASK}| & \\
\text{required} (\text{TASK}, [\text{id}, \text{father}, \text{resource}]) & \\
\text{TASK}.\text{id} > |\text{RESOURCE}| & \\
\text{TASK}.\text{id} \leq |\text{RESOURCE}| + |\text{TASK}| & \\
\text{distinct} (\text{TASK}, \text{id}) & \\
\text{TASK}.\text{father} \geq 1 & \\
\text{TASK}.\text{father} \leq |\text{RESOURCE}| + |\text{TASK}| & \\
\text{TASK}.\text{resource} \geq 1 & \\
\text{TASK}.\text{resource} \leq |\text{RESOURCE}|
\end{align*}
\]

**Purpose**

Cover a digraph \( G \) in such a way that each vertex belongs to one distinct tree. Each tree is made up from one resource vertex and several task vertices. The resource vertices correspond to the roots of the different trees. For each resource a domain variable \( \text{nb} \_ \text{task} \) indicates how many task-vertices belong to the corresponding tree. For each task a domain variable \( \text{resource} \) gives the identifier of the resource that can handle the task.

**Example**

\[
\begin{align*}
\langle \text{id} - 1 \text{ nb} \_ \text{task} - 4, \text{id} - 2 \text{ nb} \_ \text{task} - 0, \text{id} - 3 \text{ nb} \_ \text{task} - 1 \rangle, \\
\langle \text{id} - 4 \text{ father} - 8 \text{ resource} - 1, \\
\text{id} - 5 \text{ father} - 3 \text{ resource} - 3, \\
\text{id} - 6 \text{ father} - 8 \text{ resource} - 1, \\
\text{id} - 7 \text{ father} - 1 \text{ resource} - 1, \\
\text{id} - 8 \text{ father} - 1 \text{ resource} - 1 \rangle
\end{align*}
\]

The tree_resource constraint holds since the graph associated with the items of the RESOURCE and the TASK collections corresponds to 3 trees (i.e., \( |\text{RESOURCE}| = 3 \)); each tree respectively involves the vertices \( \{1, 4, 6, 7, 8\}, \{2\} \) and \( \{3, 5\} \). They are depicted by Figure 5.617, where resource and task vertices are respectively coloured in blue and pink.

**Typical**

\[
|\text{RESOURCE}| > 0 \\
|\text{TASK}| > |\text{RESOURCE}|
\]
Symmetries

- Items of RESOURCE are permutable.
- Items of TASK are permutable.

Reformulation

The tree_resource(RESOURCE, TASK) constraint can be expressed in term of a conjunction of one tree constraint, |TASK| element constraints and one global_cardinality constraint:

- The tree constraint expresses the fact that we have a well formed tree.
- The element constraint is used for expressing the link between the father attribute of an item of the TASK collection and its corresponding resource attribute.
- The global_cardinality constraint is used to link the resource attribute of the items of the TASK collection with the nb_task attribute of the items of the RESOURCE collection.

With respect to the Example slot we get the following conjunction of constraints:

- tree(3, (index - 1 succ - 1, index - 2 succ - 2, index - 3 succ - 3, index - 4 succ - 8, index - 5 succ - 3, index - 6 succ - 8, index - 7 succ - 1, index - 8 succ - 1)),
- element(8, (1, 2, 3, 1, 3, 1, 1, 1), 1),
- element(3, (1, 2, 3, 1, 3, 1, 1, 1), 3),
- element(8, (1, 2, 3, 1, 3, 1, 1, 1), 1),
- element(1, (1, 2, 3, 1, 3, 1, 1, 1), 1),
- element(1, (1, 2, 3, 1, 3, 1, 1, 1), 1),
- global_cardinality((1, 3, 1, 1, 1),
  (val - 1 noccurrence - 4, val - 2 noccurrence - 0, val - 3 noccurrence - 1)).

See also

- root concept: tree.
- used in reformulation: element, global_cardinality, tree.

Keywords

- characteristic of a constraint: derived collection.
- constraint type: graph constraint, resource constraint, graph partitioning constraint.
- final graph structure: tree, connected component.

![Figure 5.617: The three trees associated with the example](image-url)
Derived Collection

\[
\text{col} = \{\text{RESOURCE\_TASK} \rightarrow \text{collection}(\text{index} - \text{int}, \text{succ} - \text{dvar}, \text{name} - \text{dvar}), \text{item} \rightarrow \{\text{index} - \text{RESOURCE\_id}, \text{succ} - \text{RESOURCE\_id}, \text{name} - \text{RESOURCE\_id}\}, \text{item} \rightarrow \{\text{index} - \text{TASK\_id}, \text{succ} - \text{TASK\_father}, \text{name} - \text{TASK\_resource}\}\}
\]

Arc input(s)

\text{RESOURCE\_TASK}

Arc generator

\(\text{CLIQUE} \rightarrow \text{collection}(\text{resource\_task1, resource\_task2})\)

Arc arity

2

Arc constraint(s)

\begin{itemize}
  \item \text{resource\_task1.succ} = \text{resource\_task2.index}
  \item \text{resource\_task1.name} = \text{resource\_task2.name}
\end{itemize}

Graph property(ies)

\begin{itemize}
  \item \text{MAX\_NSCC} \leq 1
  \item \text{NCC} = |\text{RESOURCE}|
  \item \text{NVERTEX} = |\text{RESOURCE}| + |\text{TASK}|
\end{itemize}

For all items of \text{RESOURCE}:

Graph model

For the second graph constraint, part (A) of Figure 5.618 shows the initial graphs associated with resources 1, 2 and 3 of the Example slot. For the second graph constraint, part (B) of Figure 5.618 shows the corresponding final graphs associated with resources 1, 2 and 3. Since we use the \text{NVERTEX} graph property, the vertices of the final graphs are stressed in bold. To each resource corresponds a tree of respectively 4, 0 and 1 task-vertices.

Signature

Since the initial graph of the first graph constraint contains |\text{RESOURCE}| + |\text{TASK}| vertices, the corresponding final graph cannot have more than |\text{RESOURCE}| + |\text{TASK}| vertices. Therefore we can rewrite the graph property \text{NVERTEX} = |\text{RESOURCE}| + |\text{TASK}| to \text{NVERTEX} \geq |\text{RESOURCE}| + |\text{TASK}| and simplify \text{NVERTEX} to \text{NVERTEX}.
Figure 5.618: Initial and final graph of the tree_resource constraint
## 5.366 twin

**DESCRIPTION**

Pairs of variables related by hidden **element** constraints sharing the same table.

**LINKS**

**Constraint**

\[
\text{twin(PAIRS)}
\]

**Argument**

\[
\text{PAIRS : collection}(x - \text{dvar}, y - \text{dvar})
\]

**Restrictions**

\[
\begin{align*}
\text{required}(\text{PAIRS}, x) \\
\text{required}(\text{PAIRS}, y) \\
|\text{PAIRS}| > 0
\end{align*}
\]

**Purpose**

Enforce the condition \(\text{PAIRS}[i].x = u \land \text{PAIRS}[i].y = v \ (i \in [1, |\text{PAIRS}|]) \Rightarrow \forall j \in [1, |\text{PAIRS}|] : \text{PAIRS}[j].x = u \Leftrightarrow \text{PAIRS}[j].y = v.\)

**Example**

\[
\begin{pmatrix}
    x - 1 & y - 8, \\
    x - 9 & y - 6, \\
    x - 1 & y - 8, \\
    x - 5 & y - 0, \\
    x - 6 & y - 7, \\
    x - 9 & y - 6
\end{pmatrix}
\]

The **twin** constraint holds since 1 is paired with 8, 9 is paired with 6, 5 is paired with 0, 6 is paired with 7.

**Typical**

\[
\begin{align*}
|\text{PAIRS}| > 1 \\
|\text{PAIRS}| > \text{nval}(\text{PAIRS}.x) \\
|\text{PAIRS}| > \text{nval}(\text{PAIRS}.y) \\
\text{nval}(\text{PAIRS}.x) > 1 \\
\text{nval}(\text{PAIRS}.y) > 1 \\
\text{nval}(\text{PAIRS}.x) = \text{nval}(\text{PAIRS}.y) \\
\text{nval}(\text{PAIRS}.x) < |\text{PAIRS}| \\
\text{nval}(\text{PAIRS}.y) < |\text{PAIRS}|
\end{align*}
\]

**Arg. properties**

Contractible wrt. **PAIRS**.

**See also**

related: **element** *(pairs linked by an element with the same table)*.

**Keywords**

characteristic of a constraint: pair. constraint type: predefined constraint.
5.367 \textbf{two\_layer\_edge\_crossing}

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>two_layer_edge_crossing</td>
<td>(NCROSS, VERTICES_LAYER1, VERTICES_LAYER2, EDGES)</td>
</tr>
<tr>
<td>Arguments</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NCROSS</td>
<td>dvar</td>
<td></td>
</tr>
<tr>
<td>VERTICES_LAYER1</td>
<td>collection(id-int,pos-dvar)</td>
<td></td>
</tr>
<tr>
<td>VERTICES_LAYER2</td>
<td>collection(id-int,pos-dvar)</td>
<td></td>
</tr>
<tr>
<td>EDGES</td>
<td>collection(id-int,vertex1-int,vertex2-int)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NCROSS (\geq 0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>required(VERTICES_LAYER1, [id,pos])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VERTICES_LAYER1.id (\geq 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VERTICES_LAYER1.id (\leq</td>
<td>\text{VERTICES_LAYER1}</td>
<td>)</td>
</tr>
<tr>
<td>distinct(VERTICES_LAYER1, id)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>distinct(VERTICES_LAYER1, pos)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>required(VERTICES_LAYER2, [id,pos])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VERTICES_LAYER2.id (\geq 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VERTICES_LAYER2.id (\leq</td>
<td>\text{VERTICES_LAYER2}</td>
<td>)</td>
</tr>
<tr>
<td>distinct(VERTICES_LAYER2, id)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>distinct(VERTICES_LAYER2, pos)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>required(EDGES, [id,vertex1,vertex2])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDGES.id (\geq 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDGES.id (\leq</td>
<td>\text{EDGES}</td>
<td>)</td>
</tr>
<tr>
<td>distinct(EDGES, id)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDGES.vertex1 (\geq 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDGES.vertex1 (\leq</td>
<td>\text{VERTICES_LAYER1}</td>
<td>)</td>
</tr>
<tr>
<td>EDGES.vertex2 (\geq 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDGES.vertex2 (\leq</td>
<td>\text{VERTICES_LAYER2}</td>
<td>)</td>
</tr>
<tr>
<td>Purpose</td>
<td></td>
<td>NCROSS is the number of line-segments intersections.</td>
</tr>
<tr>
<td>Example</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.619 provides a picture of the example, where one can see the two line-segments intersections. Each line-segment of Figure 5.619 is labelled with its identifier and corresponds to an item of the EDGES collection. The two vertices on top of Figure 5.619
correspond to the items of the VERTICES_LAYER1 collection, while the three other vertices are associated with the items of VERTICES_LAYER2.

Figure 5.619: Intersection between line-segments joining two layers

Typical

|VERTICES_LAYER1| > 1
|VERTICES_LAYER2| > 1
|EDGES| ≥ |VERTICES_LAYER1|
|EDGES| ≥ |VERTICES_LAYER2|

Symmetries

- Arguments are permutable w.r.t. permutation (NCROSS) (VERTICES_LAYER1, VERTICES_LAYER2) (EDGES).
- Items of VERTICES_LAYER1 are permutable.
- Items of VERTICES_LAYER2 are permutable.

Arg. properties

Functional dependency: NCROSS determined by VERTICES_LAYER1, VERTICES_LAYER2 and EDGES.

Remark

The two-layer edge crossing minimisation problem was proved to be NP-hard in [170].

See also

common keyword: crossing, graph_crossing (line-segments intersection).

Keywords

characteristic of a constraint: derived collection.
constraint arguments: pure functional dependency.
geometry: geometrical constraint, line-segments intersection.
miscellaneous: obscure.
modelling: functional dependency.
**Derived Collection**

\[
\text{col} \left( \text{collection} \left( \text{edges extremities}-layer1 \text{-dvar}, \text{layer2}-\text{dvar} \right), \right.
\begin{array}{l}
\text{item} \left( \text{layer1} \rightarrow \text{edges vertex1} \left( \text{vertices layer1}, \text{pos, id} \right) \right), \\
\text{layer2} \rightarrow \text{edges vertex2} \left( \text{vertices layer2}, \text{pos, id} \right) \end{array}
\right)
\]

**Arc input(s)**

EDGES_EXTREMITIES

**Arc generator**

\( \text{CLIQUE}(<) \rightarrow \text{collection}(\text{edges extremities1}, \text{edges extremities2}) \)

**Arc arity**

2

**Arc constraint(s)**

\[\bigvee \left( \begin{array}{l}
\text{edges extremities1.layer1} < \text{edges extremities2.layer1}, \\
\text{edges extremities1.layer2} > \text{edges extremities2.layer2}
\end{array} \right) \]

**Graph property(ies)**

\( \text{NARC} = \text{NCROSS} \)

**Graph model**

As usual for the two-layer edge crossing problem [186], [21], positions of the vertices on each layer are represented as a permutation of the vertices. We generate a derived collection that, for each edge, contains the position of its extremities on both layers. In the arc generator we use the restriction \(<\) in order to generate one single arc for each pair of segments. This is required, since otherwise we would count more than once a line-segments intersection.

Parts (A) and (B) of Figure 5.620 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

![Figure 5.620](image)

Figure 5.620: Initial and final graph of the two_layer_edge_crossing constraint
### 5.368 two_orth_are_in_contact

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[338], used for defining orths_are_connected.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>two_orth_are_in_contact(ORTHOTOPE1,ORTHOTOPE2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>ORTHOTOPE : $\text{collection}(\text{ori-dvar}, \text{siz-dvar}, \text{end-dvar})$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>ORTHOTOPE1 : ORTHOTOPE \hspace{1cm} ORTHOTOPE2 : ORTHOTOPE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>$</td>
<td>\text{ORTHOTOPE}</td>
<td>&gt; 0$ \hspace{1cm} \text{require_at_least}(2, \text{ORTHOTOPE}, [\text{ori}, \text{siz}, \text{end}])$ \hspace{1cm} \text{ORTHOTOPE}.siz $&gt; 0$ \hspace{1cm} \text{ORTHOTOPE}.ori $\leq$ \text{ORTHOTOPE}.end \hspace{1cm} $</td>
</tr>
</tbody>
</table>

#### Purpose

Enforce the following conditions on two orthotopes $O_1$ and $O_2$:

- For all dimensions $i$, except one dimension, the projections of $O_1$ and $O_2$ on $i$ have a non-empty intersection.
- For all dimensions $i$, the distance between the projections of $O_1$ and $O_2$ on $i$ is equal to 0.

#### Example

\[
\begin{align*}
\langle \text{ori} - 1 & \text{ siz} - 3 \text{ end} - 4, \text{ori} - 5 \text{ siz} - 2 \text{ end} - 7 \rangle, \\
\langle \text{ori} - 3 & \text{ siz} - 2 \text{ end} - 5, \text{ori} - 2 \text{ siz} - 3 \text{ end} - 5 \rangle
\end{align*}
\]

Figure 5.621 shows the two rectangles of the example. The two_orth_are_in_contact constraint holds since the two rectangles are in contact: the contact is depicted by a pink line-segment.

![Figure 5.621: Two rectangles that are in contact](image_url)
Typical

$|\text{ORTHOTOPE}| > 1$

Symmetries

- Arguments are permutable w.r.t. permutation (ORTHOTOPE1, ORTHOTOPE2).
- Items of ORTHOTOPE1 and ORTHOTOPE2 are permutable (same permutation used).

Used in

orths are_connected.

See also

implies: two_orth_do_not_overlap.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint,

constraint network structure: Berge-acyclic constraint network.

constraint type: logic.

filtering: arc-consistency.

gometry: geometrical constraint, touch, contact, non-overlapping, orthotope.
Arc input(s) ORTHOTOPE1 ORTHOTOPE2
Arc generator \textit{PRODUCT}(=) \rightarrow \textit{collection}(\text{orthotope1, orthotope2})
Arc arity 2
Arc constraint(s) • orthotope1.end > orthotope2.ori
• orthotope2.end > orthotope1.ori

Graph property(ies) \textbf{NARC} = |ORTHOTOPE1| − 1

Arc input(s) ORTHOTOPE1 ORTHOTOPE2
Arc generator \textit{PRODUCT}(=) \rightarrow \textit{collection}(\text{orthotope1, orthotope2})
Arc arity 2
Arc constraint(s) \max \left( 0, \max(\text{orthotope1.ori, orthotope2.ori}) − \min(\text{orthotope1.end, orthotope2.end}) \right) = 0

Graph property(ies) \textbf{NARC} = |ORTHOTOPE1|

Graph model

Parts (A) and (B) of Figure 5.622 respectively show the initial and final graph associated with the first graph constraint of the \textbf{Example} slot. Since we use the \textbf{NARC} graph property, the unique arc of the final graph is stressed in bold. It corresponds to the fact that the projection in dimension 1 of the two rectangles of the example overlap.

Figure 5.622: Initial and final graph of the \texttt{two\_orth\_are\_in\_contact} constraint

Signature

Consider the second graph constraint. Since we use the arc generator \textit{PRODUCT}(=) on the collections ORTHOTOPE1 and ORTHOTOPE2, and because of the restriction \(|ORTHOTOPE1| = |ORTHOTOPE2|\), the maximum number of arcs of the corresponding final graph is equal to \(|ORTHOTOPE1|\). Therefore we can rewrite the graph property \textbf{NARC} = |ORTHOTOPE1| to \textbf{NARC} \geq |ORTHOTOPE1| and simplify \textbf{NARC} to \textbf{NARC}. 
Automaton

Figure 5.623 depicts the automaton associated with the two_orth_are_in_contact constraint. Let ORI1, SIZ1, and END1, respectively be the ori, the siz and the end attributes of the \(i\)th item of the ORTHOTOPE1 collection. Let ORI2, SIZ2, and END2, respectively be the ori, the siz and the end attributes of the \(i\)th item of the ORTHOTOPE2 collection. To each sextuple \((\text{ORI}_1, \text{SIZ}_1, \text{END}_1, \text{ORI}_2, \text{SIZ}_2, \text{END}_2)\) corresponds a signature variable \(S_i\), which takes its value in \(\{0, 1, 2\}\), as well as the following signature constraint:

\[
\begin{align*}
((\text{SIZ}_1 > 0) \land (\text{SIZ}_2 > 0) \land (\text{END}_1 > \text{ORI}_2) \land (\text{END}_2 > \text{ORI}_1)) & \iff S_i = 0 \\
((\text{SIZ}_1 > 0) \land (\text{SIZ}_2 > 0) \land (\text{END}_1 = \text{ORI}_2 \lor \text{END}_2 = \text{ORI}_1)) & \iff S_i = 1.
\end{align*}
\]

Figure 5.623: Automaton of the two_orth_are_in_contact constraint

Figure 5.624: Hypergraph of the reformulation corresponding to the automaton of the two_orth_are_in_contact constraint
5.369 two_orth_column

<table>
<thead>
<tr>
<th>DESCRIBTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Used for defining <code>diffn.column</code>.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td><code>two_orth_column(ORTHOTOPE1, ORTHOTOPE2, DIM)</code></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td><code>ORTHOTOPE : collection(ori−dvar, siz−dvar, end−dvar)</code></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td><code>ORTHOTOPE1 : ORTHOTOPE</code></td>
<td><code>ORTHOTOPE2 : ORTHOTOPE</code></td>
</tr>
<tr>
<td>Restrictions</td>
<td>`</td>
<td>ORTHOTOPE</td>
</tr>
</tbody>
</table>

**Purpose**

Let $P_1$ and $P_2$ respectively denote the projections of $ORTHOTOPE1$ and $ORTHOTOPE2$ in dimension $DIM$. If $P_1$ and $P_2$ overlap then the size of their intersection is equal to the size of $ORTHOTOPE1$ in dimension $DIM$, as well as to the size of $ORTHOTOPE2$ in dimension $DIM$.

**Example**

\[
\begin{align*}
(\langle \text{ori}−1, \text{siz}−3, \text{end}−4, \text{ori}−1, \text{siz}−1, \text{end}−2 \rangle, \\
(\langle \text{ori}−4, \text{siz}−2, \text{end}−6, \text{ori}−1, \text{siz}−3, \text{end}−4 \rangle, 1)
\end{align*}
\]

Figure 5.625: Initial and final graph of the `two_orth_column` constraint
Typical

$|\text{ORTHOTOPE}| > 1$

Symmetry

Arguments are permutable w.r.t. permutation (ORTHOTOPE1, ORTHOTOPE2) (DIM).

Used in

diffn_column.

See also

implies: two_orth_include.
related: diffn(an extension of the diffn constraint).

Keywords

constraint type: logic.
geometry: geometrical constraint, positioning constraint, orthotope, guillotine cut.
<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>ORTHOTOPE1 ORTHOTOPE2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>PRODUCT(=) → \text{collection}(\text{orthotope1, orthotope2})</td>
</tr>
<tr>
<td>Arc arity</td>
<td>2</td>
</tr>
</tbody>
</table>
| Arc constraint(s) | \( \land \left( \begin{array}{l}
\text{orthotope1.key} = \text{DIM}, \\
\text{orthotope1.ori} < \text{orthotope2.end}, \\
\text{orthotope2.ori} < \text{orthotope1.end}, \\
\text{orthotope1.siz} > 0, \\
\text{orthotope2.siz} > 0 \\
\min(\text{orthotope1.end, orthotope2.end}) - \\
\max(\text{orthotope1.ori, orthotope2.ori}) = , \\
\text{orthotope1.siz} = \text{orthotope2.siz}
\end{array} \right) \) |
| Graph property(ies) | NARC = 1 |
5.370  two_orth_do_not_overlap

### Origin
Used for defining *diffn*.

### Constraint
\[
\text{two_orth_do_not_overlap}(\text{ORTHO TOPE1}, \text{ORTHO TOPE2})
\]

### Type
**ORTHOTOPE**: collection(ori−dvar, siz−dvar, end−dvar)

### Arguments
ORTHO TOPE1 : ORTHOTOPE
ORTHO TOPE2 : ORTHOTOPE

### Restrictions
\[
|\text{ORTHOTOPE}| > 0 \\
\text{require_at_least}(2, \text{ORTHOTOPE}, [\text{ori, siz, end}]) \\
\text{ORTHOTOPE}.\text{siz} \geq 0 \\
\text{ORTHOTOPE}.\text{ori} \leq \text{ORTHOTOPE}.\text{end} \\
|\text{ORTHOTOPE1}| = |\text{ORTHOTOPE2}| \\
\text{orth_link_ori_siz_end}(\text{ORTHOTOPE1}) \\
\text{orth_link_ori_siz_end}(\text{ORTHOTOPE2})
\]

### Purpose
For two orthotopes \(O_1\) and \(O_2\) enforce that there exists at least one dimension \(i\) such that the projections on \(i\) of \(O_1\) and \(O_2\) do not overlap.

### Example
\[
\left(\langle \text{ori} − 2, \text{siz} − 2, \text{end} − 4, \text{ori} − 1, \text{siz} − 3, \text{end} − 4\rangle, \langle \text{ori} − 4, \text{siz} − 4, \text{end} − 8, \text{ori} − 3, \text{siz} − 3, \text{end} − 6\rangle \right)
\]

Figure 5.626 represents the respective position of the two rectangles of the example. The coordinates of the leftmost lowest corner of each rectangle are stressed in bold. The **two_orth_do_not_overlap** constraint holds since the two rectangles do not overlap.

![Figure 5.626: The two rectangles of the example](image)

### Typical
\[
|\text{ORTHOTOPE}| > 1
\]
Symmetries

- Arguments are permutable w.r.t. permutation (ORTHOTOPE1, ORTHOTOPE2).
- Items of ORTHOTOPE1 and ORTHOTOPE2 are permutable (same permutation used).
- ORTHOTOPE1.siz can be decreased to any value $\geq 0$.
- ORTHOTOPE2.siz can be decreased to any value $\geq 0$.

Used in
diffn.

See also
implied by: two_orth_are_in_contact.

Keywords

- characteristic of a constraint: automaton, automaton without counters, reified automaton constraint,
- constraint network structure: Berge-acyclic constraint network.
- constraint type: logic.
- filtering: arc-consistency, constructive disjunction.
- final graph structure: bipartite, no loop.
- geometry: geometrical constraint, non-overlapping, orthotope.
Arc input(s) | ORTHOTOPE1 ORTHOTOPE2
---|---
Arc generator | \( \text{SYMmetric\_PRODUCT}(=) \mapsto \text{collection}(\text{orthotope1, orthotope2}) \)
Arc arity | 2
Arc constraint(s) | orthotope1.end \( \leq \) orthotope2.ori \( \lor \) orthotope1.siz = 0
Graph property(ies) | \( \text{NARC} \geq 1 \)
Graph class | • BIPARTITE
• NO LOOP

**Graph model**

We build an initial graph where each arc corresponds to the fact that, either the projection of an orthotope on a given dimension is empty, either it is located before the projection in the same dimension of the other orthotope. Finally we ask that at least one arc constraint remains in the final graph.

Parts (A) and (B) of Figure 5.627 respectively show the initial and final graph associated with the Example slot. Since we use the NARC graph property, the unique arc of the final graph is stressed in bold. It corresponds to the fact that the projection in dimension 1 of the first orthotope is located before the projection in dimension 1 of the second orthotope. Therefore the two orthotopes do not overlap.

![Figure 5.627: Initial and final graph of the two orth do not overlap constraint](image)
Automaton

Figure 5.628 depicts the automaton associated with the two_orth_do_not_overlap constraint. Let ORI1, SIZ1, and END1, respectively be the ori, the siz and the end attributes of the $i^{th}$ item of the ORTHOTOPE1 collection. Let ORI2, SIZ2, and END2, respectively be the ori, the siz and the end attributes of the $i^{th}$ item of the ORTHOTOPE2 collection. To each sextuple $(ORI1_i, SIZ1_i, END1_i, ORI2_i, SIZ2_i, END2_i)$ corresponds a 0-1 signature variable $S_i$ as well as the following signature constraint: $(SIZ1_i > 0) \land (SIZ2_i > 0) \land (END1_i > ORI2_i) \land (END2_i > ORI1_i) \Leftrightarrow S_i.$

![Automaton Diagram](image1)

Figure 5.628: Automaton of the two_orth_do_not_overlap constraint

![Hypergraph Diagram](image2)

Figure 5.629: Hypergraph of the reformulation corresponding to the automaton of the two_orth_do_not_overlap constraint
5.371  two_orth_include

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Used for defining <code>diffn.include</code>.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td><code>two_orth_include(ORTHOTOPE1, ORTHOTOPE2, DIM)</code></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td><code>ORTHOTOPE : collection(ori-dvar, siz-dvar, end-dvar)</code></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td><code>ORTHOTOPE1 : ORTHOTOPE</code>&lt;br&gt;<code>ORTHOTOPE2 : ORTHOTOPE</code>&lt;br&gt;<code>DIM : int</code></td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>`</td>
<td>ORTHOTOPE</td>
</tr>
<tr>
<td>Purpose</td>
<td>Let ( P_1 ) and ( P_2 ) respectively denote the projections of <code>ORTHOTOPE1</code> and <code>ORTHOTOPE2</code> in dimension <code>DIM</code>. If ( P_1 ) and ( P_2 ) overlap then, either ( P_1 ) is included in ( P_2 ), either ( P_2 ) is included in ( P_1 ).</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td><code>[(ori - 1 siz - 3 end - 4, ori - 1 siz - 1 end - 2), (ori - 1 siz - 2 end - 3, ori - 2 siz - 3 end - 5), 1]</code></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.630: Initial and final graph of the `two_orth_include` constraint
<p>| Typical | $|\text{ORTHOTOPE}| &gt; 1$ |
|---|---|
| Symmetry | Arguments are \textit{permutable} \textit{w.r.t.} permutation ($\text{ORTHOTOPE}_1, \text{ORTHOTOPE}_2$) (DIM). |
| Used in | \textit{diffn} \textit{include}. |
| See also | \textit{implied by: two_orth_column}. \textit{related: diffn (an extension of the diffn constraint)}. |
| Keywords | \textit{constraint type:} logic. \textit{geometry:} geometrical constraint, positioning constraint, orthotope. |</p>
<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>ORTHOTOPE1 ORTHOTOPE2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>$PRODUCT(=) \rightarrow \text{collection}(\text{orthotope1}, \text{orthotope2})$</td>
</tr>
<tr>
<td>Arc arity</td>
<td>2</td>
</tr>
</tbody>
</table>
| Arc constraint(s)        | $\land$
\[
\begin{align*}
&\text{orthotope1.key} = \text{DIM}, \\
&\text{orthotope1_ori} < \text{orthotope2.end}, \\
&\text{orthotope2_ori} < \text{orthotope1.end}, \\
&\text{orthotope1.siz} > 0, \\
&\text{orthotope2.siz} > 0 \\
&\min(\text{orthotope1.end, orthotope2.end}) = \\
&\max(\text{orthotope1_ori, orthotope2_ori}) \\
&\min(\text{orthotope1.siz, orthotope2.siz}) \\
\end{align*}
\]  
| Graph property(ies)      | $\text{NARC} = 1$     |
5.372 used_by

<table>
<thead>
<tr>
<th>Description</th>
<th>Links</th>
<th>Graph</th>
<th>Automaton</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>N. Beldiceanu</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>used_by(VARIABLES1, VARIABLES2)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| **Arguments** | VARIABLES1 : `collection(var-dvar)`  
                VARIABLES2 : `collection(var-dvar)` |       |           |
| **Restrictions** | `|VARIABLES1| ≥ |VARIABLES2|`  
                                 `required(VARIABLES1.var)`  
                                 `required(VARIABLES2.var)` |       |           |
| **Purpose** | All the values of the variables of collection VARIABLES2 are used by the variables of collection VARIABLES1. |       |           |

**Example**

The used_by constraint holds since, for each value occurring within the collection VARIABLES2 = `⟨1, 1, 2, 5⟩`, its number of occurrences within VARIABLES1 = `⟨1, 9, 1, 5, 2, 1⟩` is greater than or equal to its number of occurrences within VARIABLES2:

- Value 1 occurs 3 times within `⟨1, 9, 1, 5, 2, 1⟩` and 2 times within `⟨1, 1, 2, 5⟩`.
- Value 2 occurs 1 times within `⟨1, 9, 1, 5, 2, 1⟩` and 1 times within `⟨1, 1, 2, 5⟩`.
- Value 5 occurs 1 times within `⟨1, 9, 1, 5, 2, 1⟩` and 1 times within `⟨1, 1, 2, 5⟩`.

**Typical**

- `|VARIABLES1| > 1`  
  `range(VARIABLES1.var) > 1`  
  `|VARIABLES2| > 1`  
  `range(VARIABLES2.var) > 1`  

**Symmetries**

- Items of VARIABLES1 are **permutable**.
- Items of VARIABLES2 are **permutable**.
- All occurrences of two distinct values in VARIABLES1.var or VARIABLES2.var can be **swapped**; all occurrences of a value in VARIABLES1.var or VARIABLES2.var can be **renamed** to any unused value.
Arg. properties

- Contractible wrt. VARIABLES2.
- Extensible wrt. VARIABLES1.
- Aggregate: VARIABLES1(union), VARIABLES2(union).

Algorithm

As described in [45] we can pad VARIABLES2 with dummy variables such that its cardinality will be equal to that cardinality of VARIABLES1. The domain of a dummy variable contains all of the values. Then, we have a same constraint between the two sets. Direct arc-consistency and bound-consistency algorithms based on a flow model are also proposed in [45, 47, 213].

Reformulation

The used_by((var - U1 var - U2, ..., var - U|VARIABLES1|), (var - V1 var - V2, ..., var - V|VARIABLES2|)) constraint can be expressed in term of a conjunction of |VARIABLES2| reified constraints of the form:

\[ \sum_{1 \leq j \leq |VARIABLES1|} (V_i = U_j) \geq \sum_{1 \leq j \leq |VARIABLES2|} (V_i = V_j) \ (i \in [1, |VARIABLES2|]). \]

Used in

int.value_precede_chain, k_used_by.

See also

generalisation: used_by_interval (variable replaced by variable/constant),
used_by_modulo (variable replaced by variable \mod constant),
used_by_partition (variable replaced by variable \in partition).

implied by: same.

implies: uses.

soft variant: soft_used_by_var (variable-based violation measure).

system of constraints: k_used_by.

Keywords

characteristic of a constraint: sort based reformulation, automaton, automaton with array of counters.
combinatorial object: multiset.
constraint arguments: constraint between two collections of variables.
filtering: flow, arc-consistency, bound-consistency, DFS-bottleneck.
modelling: inclusion.
Arc input(s)  VARIABLES1 VARIABLES2
Arc generator  \( PRODUCT \rightarrow \) collection(variables1,variables2)
Arc arity  2
Arc constraint(s)  variables1.var = variables2.var
Graph property(ies)  
• for all connected components: \( \text{NSOURCE} \geq \text{NSINK} \)
• \( \text{NSINK} = |\text{VARIABLES2}| \)

Graph model
Parts (A) and (B) of Figure 5.631 respectively show the initial and final graph associated with the Example slot. Since we use the \( \text{NSOURCE} \) and \( \text{NSINK} \) graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. Note that the vertex corresponding to the variable assigned to value 9 was removed from the final graph since there is no arc for which the associated equality constraint holds. The used by constraint holds since:

• For each connected component of the final graph the number of sources is greater than or equal to the number of sinks.
• The number of sinks of the final graph is equal to \( |\text{VARIABLES2}| \).

Signature
Since the initial graph contains only sources and sinks, and since sources of the initial graph cannot become sinks of the final graph, we have that the maximum number of sinks of the final graph is equal to \( |\text{VARIABLES2}| \). Therefore we can rewrite \( \text{NSINK} = |\text{VARIABLES2}| \) to \( \text{NSINK} \geq |\text{VARIABLES2}| \) and simplify \( \text{NSINK} \) to \( \text{NSINK} \).
Figure 5.631: Initial and final graph of the `used_by` constraint
Automaton

Figure 5.632 depicts the automaton associated with the used_by constraint. To each item of the collection VARIABLES1 corresponds a signature variable $S_i$ that is equal to 0. To each item of the collection VARIABLES2 corresponds a signature variable $S_{i+|VARIABLES1|}$ that is equal to 1.

Figure 5.632: Automaton of the used_by constraint
5.373 used_by_interval

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Derived from used_by.</td>
<td></td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>used_by_interval(VARIABLES1, VARIABLES2, SIZE_INTERVAL)</td>
<td></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>VARIABLES1 : collection(var−dvar)</td>
<td>VARIABLES2 : collection(var−dvar)</td>
</tr>
<tr>
<td><strong>Restrictions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>Let ( N_i ) (respectively ( M_i )) denote the number of variables of the collection VARIABLES1 (respectively VARIABLES2) that take a value in the interval ( [\text{SIZE_INTERVAL} \cdot i, \text{SIZE_INTERVAL} \cdot i + \text{SIZE_INTERVAL} - 1] ). For all integer ( i ) we have ( M_i &gt; 0 \Rightarrow N_i \geq M_i ).</td>
<td></td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>( \left{ \right. )</td>
<td></td>
</tr>
</tbody>
</table>
| | \left. \begin{array}{l}
\text{var} - 1, \\
\text{var} - 9, \\
\text{var} - 1, \\
\text{var} - 8, \\
\text{var} - 6, \\
\text{var} - 2 \\
\left. \left(1, 0, 7, 7\right), 3 \right. \right\} |       |

In the example, the third argument \( \text{SIZE_INTERVAL} = 3 \) defines the following family of intervals \( [3 \cdot k, 3 \cdot k + 2] \), where \( k \) is an integer. Consequently the values of the collection VARIABLES2 = \( \langle 1, 0, 7, 7 \rangle \) are respectively located within intervals \( [0, 2], [0, 2], [6, 8], [6, 8] \). Therefore intervals \( [0, 2] \) and \( [6, 8] \) are respectively used 2 and 2 times.

Similarly, the values of the collection VARIABLES1 = \( \langle 1, 9, 1, 8, 6, 2 \rangle \) are respectively located within intervals \( [0, 2], [9, 11], [0, 2], [6, 8], [6, 8], [0, 2] \). Therefore intervals \( [0, 2], [6, 8] \) and \( [9, 11] \) are respectively used 3, 2 and 1 times.

Consequently, the used_by_interval constraint holds since, for each interval associated with the collection VARIABLES2 = \( \langle 1, 0, 7, 7 \rangle \), its number of occurrences within VARIABLES1 = \( \langle 1, 9, 1, 8, 6, 2 \rangle \) is greater than or equal to its number of occurrences within VARIABLES2:

- Interval \( [0, 2] \) occurs 3 times within \( \langle 1, 9, 1, 8, 6, 2 \rangle \) and 2 times within \( \langle 1, 0, 7, 7 \rangle \).
- Interval \( [6, 8] \) occurs 2 times within \( \langle 1, 9, 1, 8, 6, 2 \rangle \) and 2 times within \( \langle 1, 0, 7, 7 \rangle \).
Typical

$|\text{VARIABLES1}| > 1$
$\text{range}(\text{VARIABLES1}.\text{var}) > 1$
$|\text{VARIABLES2}| > 1$
$\text{range}(\text{VARIABLES2}.\text{var}) > 1$
$\text{SIZE}_{\text{INTERVAL}} > 1$
$\text{SIZE}_{\text{INTERVAL}} < \text{range}(\text{VARIABLES1}.\text{var})$
$\text{SIZE}_{\text{INTERVAL}} < \text{range}(\text{VARIABLES2}.\text{var})$

Symmetries

- Items of \text{VARIABLES1} are \textbf{permutable}.
- Items of \text{VARIABLES2} are \textbf{permutable}.
- An occurrence of a value of \text{VARIABLES1}.\text{var} that belongs to the $k$-th interval, of size \text{SIZE}_{\text{INTERVAL}}, can be \textbf{replaced} by any other value of the same interval.
- An occurrence of a value of \text{VARIABLES2}.\text{var} that belongs to the $k$-th interval, of size \text{SIZE}_{\text{INTERVAL}}, can be \textbf{replaced} by any other value of the same interval.

Arg. properties

- \textbf{Contractible} wrt. \text{VARIABLES2}.
- \textbf{Extensible} wrt. \text{VARIABLES1}.
- \textbf{Aggregate}: \text{VARIABLES1(union)}, \text{VARIABLES2(union)}, \text{SIZE}_{\text{INTERVAL}}(\text{id}).

Reformulation

The \text{used_by_interval}((\text{var} - U_1 \text{var} - U_2, ..., \text{var} - U_\text{|VARIABLES1|}), (\text{var} - V_1 \text{var} - V_2, ..., \text{var} - V_\text{|VARIABLES2|}), \text{SIZE}_{\text{INTERVAL}}) constraint can be expressed by introducing $|\text{VARIABLES1}| + |\text{VARIABLES2}|$ \textit{quotient} variables

$U_i = \text{SIZE}_{\text{INTERVAL}} \cdot P_i + R_i, R_i \in [0, \text{SIZE}_{\text{INTERVAL}} - 1] (i \in [1, |\text{VARIABLES1}|])$,

$V_i = \text{SIZE}_{\text{INTERVAL}} \cdot Q_i + S_i, S_i \in [0, \text{SIZE}_{\text{INTERVAL}} - 1] (i \in [1, |\text{VARIABLES2}|])$,

in term of a conjunction of $|\text{VARIABLES2}|$ \textit{reified} constraints of the form:

$\sum_{1 \leq j \leq |\text{VARIABLES1}|}(Q_i = P_j) \geq \sum_{1 \leq j \leq |\text{VARIABLES2}|}(Q_i = Q_j) (i \in [1, |\text{VARIABLES2}|])$.

Used in

\textit{k_used_by_interval}.

See also

\textit{implied by}: \textit{same_interval}.

\textit{soft variant}: \textit{soft_used_by_interval_var} (\textit{variable-based violation measure}).

\textit{specialisation}: \textit{used_by} (\textit{variable/constant replaced by variable}).

\textit{system of constraints}: \textit{k_used_by_interval}.

Keywords

- characteristic of a constraint: sort based reformulation.
- \textbf{constraint arguments}: constraint between two collections of variables.
- \textbf{modelling}: inclusion, interval.
Arc input(s)  VARIABLES1 VARIABLES2
Arc generator  $PRODUCT \rightarrow \text{collection}(\text{variables1}, \text{variables2})$
Arc arity  2
Arc constraint(s)  variables1.var/\text{SIZE\_INTERVAL} = variables2.var/\text{SIZE\_INTERVAL}
Graph property(ies)  
  • for all connected components: $\text{NSOURCE} \geq \text{NSINK}$
  • $\text{NSINK} = |\text{VARIABLES2}|$

Graph model

Parts (A) and (B) of Figure 5.633 respectively show the initial and final graph associated with the Example slot. Since we use the $\text{NSOURCE}$ and $\text{NSINK}$ graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. Note that the vertex corresponding to the variable that takes value 9 was removed from the final graph since there is no arc for which the associated equivalence constraint holds. The used by interval constraint holds since:

  • For each connected component of the final graph the number of sources is greater than or equal to the number of sinks.
  • The number of sinks of the final graph is equal to $|\text{VARIABLES2}|$.

Signature

Since the initial graph contains only sources and sinks, and since sources of the initial graph cannot become sinks of the final graph, we have that the maximum number of sinks of the final graph is equal to $|\text{VARIABLES2}|$. Therefore we can rewrite $\text{NSINK} = |\text{VARIABLES2}|$ to $\text{NSINK} \geq |\text{VARIABLES2}|$ and simplify $\text{NSINK}$ to $\text{NSINK}$.
Figure 5.633: Initial and final graph of the used_by_interval constraint
5.374 used_by_modulo

**DESCRIPTION**

Derived from used_by.

**LINKS**

used_by_modulo(VARIABLES1, VARIABLES2, M)

**ARGUMENTS**

VARIABLES1 : collection(var−dvar)
VARIABLES2 : collection(var−dvar)
M : int

**RESTRICTIONS**

|VARIABLES1| ≥ |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)
M > 0

**PURPOSE**

For each integer $R$ in $[0, M - 1]$, let $N1_R$ (respectively $N2_R$) denote the number of variables of VARIABLES1 (respectively VARIABLES2) that have $R$ as a rest when divided by $M$. For all $R$ in $[0, M - 1]$ we have $N2_R > 0 \Rightarrow N1_R \geq N2_R$.

**EXAMPLE**

\[
\begin{pmatrix}
\text{var} - 1, \\
\text{var} - 9, \\
\text{var} - 4, \\
\text{var} - 5, \\
\text{var} - 2, \\
\text{var} - 1
\end{pmatrix}, \langle 7, 1, 2, 5 \rangle, 3
\]

The values of the collection VARIABLES2 = $\langle 7, 1, 2, 5 \rangle$ are respectively associated with the equivalence classes $7 \mod 3 = 1$, $1 \mod 3 = 1$, $2 \mod 3 = 2$, $5 \mod 3 = 2$. Therefore the equivalence classes 1 and 2 are respectively used 2 and 2 times.

Similarly, the values of the collection VARIABLES1 = $\langle 1, 9, 4, 5, 2, 1 \rangle$ associated with the equivalence classes $1 \mod 3 = 1$, $9 \mod 3 = 0$, $4 \mod 3 = 1$, $5 \mod 3 = 2$, $2 \mod 3 = 2$, $1 \mod 3 = 1$. Therefore the equivalence classes 0, 1 and 2 are respectively used 1, 3 and 2 times.

Consequently, the used_by_modulo constraint holds since, for each equivalence class associated with the collection VARIABLES2 = $\langle 7, 1, 2, 5 \rangle$, its number of occurrences within VARIABLES1 = $\langle 1, 9, 4, 5, 2, 1 \rangle$ is greater than or equal to its number of occurrences within VARIABLES2:

- The equivalence class 1 occurs 3 times within $\langle 1, 9, 4, 5, 2, 1 \rangle$ and 2 times within $\langle 7, 1, 2, 5 \rangle$.
- The equivalence class 2 occurs 2 times within $\langle 1, 9, 4, 5, 2, 1 \rangle$ and 2 times within $\langle 7, 1, 2, 5 \rangle$. 
Typical

\[ |\text{VARIABLES1}| > 1 \]
\[ \text{range}(\text{VARIABLES1}.\text{var}) > 1 \]
\[ |\text{VARIABLES2}| > 1 \]
\[ \text{range}(\text{VARIABLES2}.\text{var}) > 1 \]
\[ M > 1 \]
\[ M < \text{maxval}(\text{VARIABLES1}.\text{var}) \]
\[ M < \text{maxval}(\text{VARIABLES2}.\text{var}) \]

Symmetries

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value \( u \) of VARIABLES1.var can be replaced by any other value \( v \) such that \( v \) is congruent to \( u \) modulo \( M \).
- An occurrence of a value \( u \) of VARIABLES2.var can be replaced by any other value \( v \) such that \( v \) is congruent to \( u \) modulo \( M \).

Arg. properties

- Contractible wrt. VARIABLES2.
- Extensible wrt. VARIABLES1.
- Aggregate: VARIABLES1(union), VARIABLES2(union), M(id).

Used in

\[ k\text{\_used\_by\_modulo}. \]

See also

implied by: same modulo.
soft variant: soft\_used\_by\_modulo\_var (variable-based violation measure).
specialisation: used by (variable mod constant replaced by variable).
system of constraints: k\_used\_by\_modulo.

Keywords

characteristic of a constraint: modulo, sort based reformulation.
constraint arguments: constraint between two collections of variables.
modelling: inclusion.
### Graph model

Parts (A) and (B) of Figure 5.634 respectively show the initial and final graph associated with the Example slot. Since we use the NSOURCE and NSINK graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. Note that the vertex corresponding to the variable that takes value 9 was removed from the final graph since there is no arc for which the associated equivalence constraint holds. The used by modulo constraint holds since:

- For each connected component of the final graph the number of sources is greater than or equal to the number of sinks.
- The number of sinks of the final graph is equal to |VARIABLES2|.

### Signature

Since the initial graph contains only sources and sinks, and since sources of the initial graph cannot become sinks of the final graph, we have that the maximum number of sinks of the final graph is equal to |VARIABLES2|. Therefore we can rewrite NSINK = |VARIABLES2| to NSINK ≥ |VARIABLES2| and simplify NSINK to NSINK.
Figure 5.634: Initial and final graph of the used_by_modulo constraint
5.375  used_by_partition

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Derived from used_by.</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>used_by_partition(VARIABLES1, VARIABLES2, PARTITIONS)</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>VALUES : collection(val=int)</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VARIABLES1 : collection(var=dvar)</td>
<td>VARIABLES2 : collection(var=dvar)</td>
</tr>
<tr>
<td>Restrictions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>For each integer i in [1,</td>
<td>PARTITIONS]</td>
</tr>
</tbody>
</table>

Example

\[
\begin{pmatrix}
\text{val} = 1, \\
\text{val} = 9, \\
\text{var} = 1, \\
\text{var} = 6, \\
\text{var} = 2, \\
\text{var} = 3, \\
\text{p} = \langle 1, 3 \rangle, \\
\text{p} = \langle 4 \rangle, \\
\text{p} = \langle 2, 6 \rangle
\end{pmatrix}
\]

The different values of the collection VARIABLES2 = \langle 1, 3, 6, 6 \rangle are respectively associated with the partitions p \langle 1, 3 \rangle, p \langle 1, 3 \rangle, p \langle 2, 6 \rangle, and p \langle 2, 6 \rangle. Therefore partitions p \langle 1, 3 \rangle and p \langle 2, 6 \rangle are respectively used 2 and 2 times.

Similarly, the different values of the collection VARIABLES1 = \langle 1, 9, 1, 6, 2, 3 \rangle (except value 9, which does not occur in any partition) are respectively associated with the partitions p \langle 1, 3 \rangle, p \langle 1, 3 \rangle, p \langle 2, 6 \rangle, p \langle 2, 6 \rangle, and p \langle 1, 3 \rangle. Therefore partitions p \langle 1, 3 \rangle and p \langle 2, 6 \rangle are respectively used 3 and 2 times.
Consequently, the used_by_partition constraint holds since, for each partition associated with the collection VARIABLES2 = \langle 1, 3, 6, 6 \rangle, its number of occurrences within VARIABLES1 = \langle 1, 9, 1, 6, 2, 3 \rangle is greater than or equal to its number of occurrences within VARIABLES2:

- Partition p = \langle 1, 3 \rangle occurs 3 times within \langle 1, 9, 1, 6, 2, 3 \rangle and 2 times within \langle 1, 3, 6, 6 \rangle.
- Partition p = \langle 2, 6 \rangle occurs 2 times within \langle 1, 9, 1, 6, 2, 3 \rangle and 2 times within \langle 1, 3, 6, 6 \rangle.

**Typical**

\[
\begin{align*}
|\text{VARIABLES1}| & > 1 \\
\text{range}(\text{VARIABLES1}.\text{var}) & > 1 \\
|\text{VARIABLES2}| & > 1 \\
\text{range}(\text{VARIABLES2}.\text{var}) & > 1 \\
|\text{VARIABLES1}| & > |\text{PARTITIONS}| \\
|\text{VARIABLES2}| & > |\text{PARTITIONS}|
\end{align*}
\]

**Symmetries**

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- Items of PARTITIONS are permutable.
- Items of PARTITIONS.p are permutable.
- An occurrence of a value of VARIABLES1.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.
- An occurrence of a value of VARIABLES2.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.

**Arg. properties**

- Contractible wrt. VARIABLES2.
- Extensible wrt. VARIABLES1.
- Aggregate: VARIABLES1(union), VARIABLES2(union), PARTITIONS(id).

**Used in**

k_used_by_partition.

**See also**

implied by: same_partition.
soft variant: soft_used_by_partition_var (variable-based violation measure).
specialisation: used_by (variable ∈ partition replaced by variable).
system of constraints: k_used_by_partition.
used in graph description: in_same_partition.

**Keywords**

characteristic of a constraint: partition, sort based reformulation.
constraint arguments: constraint between two collections of variables.
modelling: inclusion.
Arc input(s) | VARIABLES1 VARIABLES2
---|---
Arc generator | \( PRODUCT \rightarrow \text{collection}(\text{variables1}, \text{variables2}) \)
Arc arity | 2
Arc constraint(s) | \( \text{in\_same\_partition}(\text{variables1}\_\text{var}, \text{variables2}\_\text{var}, \text{PARTITIONS}) \)
Graph property(ies) | • for all connected components: \( \text{NSOURCE} \geq \text{NSINK} \)
| • \( \text{NSINK} = |\text{VARIABLES2}| \)

### Graph model

Parts (A) and (B) of Figure 5.635 respectively show the initial and final graph associated with the Example slot. Since we use the \( \text{NSOURCE} \) and \( \text{NSINK} \) graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. Note that the vertex corresponding to the variable that takes value 9 was removed from the final graph since there is no arc for which the associated equivalence constraint holds. The used\_by\_partition constraint holds since:

- For each connected component of the final graph the number of sources is greater than or equal to the number of sinks.
- The number of sinks of the final graph is equal to \( |\text{VARIABLES2}| \).

### Signature

Since the initial graph contains only sources and sinks, and since sources of the initial graph cannot become sinks of the final graph, we have that the maximum number of sinks of the final graph is equal to \( |\text{VARIABLES2}| \). Therefore we can rewrite \( \text{NSINK} = |\text{VARIABLES2}| \) to \( \text{NSINK} \geq |\text{VARIABLES2}| \) and simplify \( \text{NSINK} \) to \( \text{NSINK} \).
Figure 5.635: Initial and final graph of the `used_by_partition` constraint
5.376  uses

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[59]</td>
<td></td>
</tr>
</tbody>
</table>

Constraint  

\[ \text{uses} \left( \text{VARIABLES1}, \text{VARIABLES2} \right) \]

Arguments

- \( \text{VARIABLES1} : \text{collection} \left( \text{var} - \text{dvar} \right) \)
- \( \text{VARIABLES2} : \text{collection} \left( \text{var} - \text{dvar} \right) \)

Restrictions

\[
\begin{align*}
\min(1, |\text{VARIABLES1}|) & \geq \min(1, |\text{VARIABLES2}|) \\
\text{required}(\text{VARIABLES1}.\text{var}) & \\
\text{required}(\text{VARIABLES2}.\text{var}) & \\
\end{align*}
\]

Purpose

The set of values assigned to the variables of the collection of variables \( \text{VARIABLES2} \) is included within the set of values assigned to the variables of the collection of variables \( \text{VARIABLES1} \).

Example

\[
\left( \langle 3, 3, 4, 6 \rangle, \langle 3, 4, 4, 4 \rangle \right)
\]

The \( \text{uses} \) constraint holds since the set of values \( \{3, 4\} \) assigned to the items of collection \( \{3, 4, 4, 4\} \) is included within the set of values \( \{3, 4, 6\} \) occurring within \( \langle 3, 3, 4, 6 \rangle \).

Typical

\[
\begin{align*}
|\text{VARIABLES1}| & > 1 \\
\text{range}(\text{VARIABLES1}.\text{var}) & > 1 \\
|\text{VARIABLES2}| & > 1 \\
\text{range}(\text{VARIABLES2}.\text{var}) & > 1 \\
|\text{VARIABLES1}| & \leq |\text{VARIABLES2}| \\
\end{align*}
\]

Symmetries

- Items of \( \text{VARIABLES1} \) are permutable.
- Items of \( \text{VARIABLES2} \) are permutable.
- All occurrences of two distinct values in \( \text{VARIABLES1}.\text{var} \) or \( \text{VARIABLES2}.\text{var} \) can be swapped; all occurrences of a value in \( \text{VARIABLES1}.\text{var} \) or \( \text{VARIABLES2}.\text{var} \) can be renamed to any unused value.

Arg. properties

- Contractible wrt. \( \text{VARIABLES2} \).
- Extensible wrt. \( \text{VARIABLES1} \).
- Aggregate: \( \text{VARIABLES1(union)}, \text{VARIABLES2(union)} \).

Remark

It was shown in [59] that, finding out whether a uses constraint has a solution or not is NP-hard. This was achieved by reduction from 3-SAT.
See also

generalisation: common.
implied by: used by.
related: roots.

Keywords

complexity: 3-SAT.
constraint arguments: constraint between two collections of variables.
final graph structure: acyclic, bipartite, no loop.
modelling: inclusion.
### Graph model

Parts (A) and (B) of Figure 5.636 respectively show the initial and final graph associated with the Example slot. Since we use the NSINK graph property, the sink vertices of the final graph are stressed with a double circle. Note that all the vertices corresponding to the variables that take values 9 or 2 were removed from the final graph since there is no arc for which the associated equality constraint holds.

![Graph Diagram](image)

**Figure 5.636: Initial and final graph of the uses constraint**

### Table

<table>
<thead>
<tr>
<th>Arc input(s)</th>
<th>VARIABLES1 VARIABLES2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc generator</td>
<td>$PRODUCT \rightarrow \text{collection}(\text{variables1}, \text{variables2})$</td>
</tr>
<tr>
<td>Arc arity</td>
<td>2</td>
</tr>
<tr>
<td>Arc constraint(s)</td>
<td>variables1.var = variables2.var</td>
</tr>
<tr>
<td>Graph property(ies)</td>
<td>NSINK =</td>
</tr>
</tbody>
</table>
| Graph class           | • ACYCLIC  
                         • BIPARTITE  
                         • NO_LOOP |

---

**Arc input(s)**

VARIABLES1, VARIABLES2

**Arc generator**

$PRODUCT \rightarrow \text{collection}(\text{variables1}, \text{variables2})$

**Arc arity**

2

**Arc constraint(s)**

variables1.var = variables2.var

**Graph property(ies)**

NSINK = |VARIABLES2|

**Graph class**

- ACYCLIC
- BIPARTITE
- NO_LOOP

---

**Graph model**

Parts (A) and (B) of Figure 5.636 respectively show the initial and final graph associated with the Example slot. Since we use the NSINK graph property, the sink vertices of the final graph are stressed with a double circle. Note that all the vertices corresponding to the variables that take values 9 or 2 were removed from the final graph since there is no arc for which the associated equality constraint holds.

![Graph Diagram](image)

**Figure 5.636: Initial and final graph of the uses constraint**
5.377 valley

**DESCRIPTION**

Origin

Derived from *inflexion*.

**LINKS**

**AUTOMATON**

**Constraint**

valley(N, VARIABLES)

**Arguments**

\[
\begin{align*}
N &: \text{dvar} \\
\text{VARIABLES} &: \text{collection(var−dvar)}
\end{align*}
\]

**Restrictions**

\[
\begin{align*}
N \geq 0 \\
2 \times N &\leq \max(|\text{VARIABLES}| - 1, 0) \\
\text{required} &\text{(VARIABLES, var)}
\end{align*}
\]

**Purpose**

A variable \(V_k\) (1 < \(k\) < \(m\)) of the sequence of variables VARIABLES = \(V_1, \ldots, V_m\) is a valley if and only if there exists an \(i\) (1 < \(i\) ≤ \(k\)) such that \(V_{i-1} > V_i\) and \(V_i = V_{i+1} = \ldots = V_k\) and \(V_k < V_{k+1}\). 

\(N\) is the total number of valleys of the sequence of variables VARIABLES.

**Example**

\[
\begin{pmatrix}
\text{var} - 1, \\
\text{var} - 1, \\
\text{var} - 4, \\
\text{var} - 8, \\
\text{var} - 8, \\
\text{var} - 2, \\
\text{var} - 7, \\
\text{var} - 1
\end{pmatrix}
\]

The valley constraint holds since the sequence 1 1 4 8 8 2 7 1 contains one valley that corresponds to the variable that is assigned to value 2.

**Figure 5.637:** The sequence and its unique valley
Typical

| VARIABLES | > 2
range(VARIABLES.var) > 1

Symmetries

- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

Contractible wrt. VARIABLES when N = 0.

Usage

Useful for constraining the number of valleys of a sequence of domain variables.

Remark

Since the arity of the arc constraint is not fixed, the valley constraint cannot be currently described. However, this would not hold anymore if we were introducing a slot that specifies how to merge adjacent vertices of the final graph.

See also

common keyword: deepest_valley, inflexion(sequence).
comparison swapped: peak.
related: no_peak.
specialisation: no_valley (the variable counting the number of valleys is set to 0 and removed).

Keywords

characteristic of a constraint: automaton, automaton with counters.
combinatorial object: sequence.
constraint network structure: sliding cyclic(1) constraint network(2).
Automaton

Figure 5.638 depicts the automaton associated with the valley constraint. To each pair of consecutive variables (VAR\_i, VAR\_i+1) of the collection VARIABLES corresponds a signature variable S\_i. The following signature constraint links VAR\_i, VAR\_i+1 and S\_i: (VAR\_i < VAR\_i+1 ⇔ S\_i = 0) ∧ (VAR\_i = VAR\_i+1 ⇔ S\_i = 1) ∧ (VAR\_i > VAR\_i+1 ⇔ S\_i = 2).

Figure 5.638: Automaton of the valley constraint

Figure 5.639: Hypergraph of the reformulation corresponding to the automaton of the valley constraint
5.378  \texttt{vec\_eq\_tuple}

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Used for defining \texttt{in_relation}.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>\texttt{vec_eq_tuple(VARIABLES,TUPLE)}</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VARIABLES : collection(var,-dvar)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TUPLE : collection(val,-int)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>\texttt{required(VARIABLES,var)}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>\texttt{required(TUPLE,val)}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>\texttt{</td>
<td>VARIABLES</td>
</tr>
</tbody>
</table>

| Purpose | Enforce a vector of domain variables to be equal to a tuple of values. |

| Example | (\langle 5, 3, 3 \rangle, \langle 5, 3, 3 \rangle) |

The \texttt{vec\_eq\_tuple} constraint holds since the first, the second and the third items of \texttt{VARIABLES} = \langle 5, 3, 3 \rangle are respectively equal to the first, the second and the third items of \texttt{TUPLE} = \langle 5, 3, 3 \rangle.

| Typical | \texttt{|VARIABLES| > 1} |
|         | \texttt{range(VARIABLES.var) > 1} |
|         | \texttt{range(TUPLE.val) > 1} |

| Symmetries | \begin{itemize} \item Arguments are \texttt{permutable} w.r.t. permutation \texttt{(VARIABLES,TUPLE)}. \item Items of \texttt{VARIABLES} and \texttt{TUPLE} are \texttt{permutable} \texttt{(same permutation used)}. \end{itemize} |

| Arg. properties | \texttt{Contractible} \texttt{wrt. VARIABLES and TUPLE (remove items from same position)}. |

| Used in | \texttt{in\_relation}. |

| See also | \texttt{generalisation: lex\_equal} \texttt{(integer replaced by variable in second argument)}. |
|          | \texttt{implies: lex\_equal}. |

| Keywords | \texttt{characteristic of a constraint: tuple}. |
|          | \texttt{constraint type: value constraint}. |
|          | \texttt{filtering: arc-consistency}. |
Arc input(s)  VARIABLES TUPLE
Arc generator  \textit{PRODUCT}(=) \rightarrow \textit{collection}\{\textit{variables}, \textit{tuple}\}
Arc arity  2
Arc constraint(s)  \textit{variables}.\textit{var} = \textit{tuple}.\textit{val}
Graph property(ies)  \textbf{NARC} = |\textit{VARIABLES}|

Graph model

Parts (A) and (B) of Figure 5.640 respectively show the initial and final graph associated with the \textbf{Example} slot. Since we use the \textbf{NARC} graph property, the arcs of the final graph are stressed in bold.

Figure 5.640: Initial and final graph of the \texttt{vec_eq_tuple} constraint

Signature

Since we use the arc generator \textit{PRODUCT}(=) on the collections \textit{VARIABLES} and \textit{TUPLE}, and because of the restriction |\textit{VARIABLES}| = |\textit{TUPLE}|, the maximum number of arcs of the final graph is equal to |\textit{VARIABLES}|. Therefore we can rewrite the graph property \textbf{NARC} = |\textit{VARIABLES}| to \textbf{NARC} \geq |\textit{VARIABLES}| and simplify \textbf{NARC} to \textbf{NARC}. 
### 5.379 visible

<table>
<thead>
<tr>
<th>Description</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Origin</strong></td>
<td>Extension of <em>accessibility</em> parameter of <em>diffn</em>.</td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>visible(K, DIMS, FROM, OBJECTS, SBOXES)</td>
</tr>
</tbody>
</table>
| **Types** | VARIABLES : collection(v–dvar)  
INTERS : collection(v–int)  
POSITIVES : collection(v–int)  
DIMDIR : collection(dim–int, dir–int) |
| **Arguments** | K : int  
DIMS : sint  
FROM : DIMDIR  
OBJECTS : collection  
  (oid–int,  
  sid–dvar,  
  x – VARIABLES,  
  start–dvar,  
  duration–dvar,  
  end–dvar)  
SBOXES : collection  
  (sid–int, t – INTEGERS, l – POSITIVES, f – DIMDIR) |
Restrictions

<table>
<thead>
<tr>
<th>VARIABLES</th>
<th>≥ 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGERS</td>
<td>≥ 1</td>
</tr>
<tr>
<td>POSITIVES</td>
<td>≥ 1</td>
</tr>
</tbody>
</table>

required(VARIABLES, v)

<table>
<thead>
<tr>
<th>VARIABLES</th>
<th>= K</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGERS</td>
<td>= K</td>
</tr>
</tbody>
</table>

required(POSITIVES, v)

<table>
<thead>
<tr>
<th>POSITIVES</th>
<th>= K</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITIVES. v</td>
<td>&gt; 0</td>
</tr>
</tbody>
</table>

required(DIMDIR, [dim, dir])

<table>
<thead>
<tr>
<th>DIMDIR</th>
<th>&gt; 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIMDIR</td>
<td>≤ K + K</td>
</tr>
</tbody>
</table>

distinct(DIMDIR, [])

<table>
<thead>
<tr>
<th>DIMDIR</th>
<th>≥ 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIMDIR</td>
<td>&lt; K</td>
</tr>
<tr>
<td>DIMDIR</td>
<td>≥ 0</td>
</tr>
<tr>
<td>DIMDIR</td>
<td>≤ 1</td>
</tr>
<tr>
<td>K</td>
<td>≥ 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIMS</th>
<th>≥ 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIMS</td>
<td>&lt; K</td>
</tr>
</tbody>
</table>

distinct(OBJECTS, oid)

required(OBJECTS, [oid, sid, x])

require_at_least(2, OBJECTS, [start, duration, end])

<table>
<thead>
<tr>
<th>OBJECTS</th>
<th>.oid</th>
<th>≥ 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECTS</td>
<td>.sid</td>
<td>≥ 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OBJECTS</th>
<th>.sid</th>
<th>≤</th>
<th>OBJECTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECTS</td>
<td>.duration</td>
<td>≥ 0</td>
<td></td>
</tr>
</tbody>
</table>

| SBAXES | ≥ 1 |

required(SBAXES, [sid, t, l])

<table>
<thead>
<tr>
<th>SBAXES</th>
<th>.sid</th>
<th>≥ 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBAXES</td>
<td>.sid</td>
<td>≤</td>
</tr>
</tbody>
</table>

do_not_overlap(SBAXES)

Holds if and only if:

1. The difference between the end in time and the start in time of each object is equal to its duration in time.

2. Given a collection of potential observations places FROM, where each observation place is specified by a dimension (i.e., an integer between 0 and \( k - 1 \)) and by a direction (i.e., an integer between 0 and 1), and given for each shifted box of SBAXES a set of visible faces, enforce that at least one visible face of each shifted box associated with an object \( o \in \) OBJECTS should be entirely visible from at least one observation place of FROM at time \( o.start \) as well as at time \( o.end - 1 \). This notion is defined in a more formal way in the Remark slot.

Purpose
The five previous examples correspond respectively to parts (I), (II), (III) and (IV) of Figure 5.642 and to Figure 5.643. Before explaining these five examples Figure 5.641 first illustrates the notion of observations places and of visible faces.

<table>
<thead>
<tr>
<th>Example</th>
<th>Observation Place</th>
<th>Visible Faces</th>
</tr>
</thead>
<tbody>
<tr>
<td>2, {0, 1},</td>
<td>{dim = 0 dir = 1},</td>
<td>{dim = 0 dir = 1},</td>
</tr>
<tr>
<td>{oid = 1 sid = 1 x = {1, 2} start = 8 duration = 8 end = 16, }</td>
<td>{oid = 1 sid = 1 x = {1, 2} start = 1 duration = 8 end = 16, }</td>
<td></td>
</tr>
<tr>
<td>{oid = 2 sid = 2 x = {4, 2} start = 1 duration = 15 end = 16 },</td>
<td>{oid = 2 sid = 2 x = {4, 2} start = 1 duration = 15 end = 16 },</td>
<td></td>
</tr>
<tr>
<td>{sid = 1 t = (0, 0) l = {1, 2} f = {dim = 0 dir = 1}, }</td>
<td>{sid = 1 t = (0, 0) l = {1, 2} f = {dim = 0 dir = 1}, }</td>
<td></td>
</tr>
<tr>
<td>{sid = 2 t = (0, 0) l = {2, 3} f = {dim = 0 dir = 1} }</td>
<td>{sid = 2 t = (0, 0) l = {2, 3} f = {dim = 0 dir = 1} }</td>
<td></td>
</tr>
</tbody>
</table>
Figure 5.641: Entirely visible faces (depicted by a thick line) of rectangles 1, 2, 3, 4, 5, 6 and 7 from the four observation places \((\text{dim} = 0, \text{dir} = 1)\), \((\text{dim} = 0, \text{dir} = 0)\), \((\text{dim} = 1, \text{dir} = 1)\) and \((\text{dim} = 1, \text{dir} = 0)\) (depicted by an arrow)
We first need to introduce a number of definitions in order to illustrate the notion of visibility.

**Definition 1.** Consider two distinct objects $o$ and $o'$ of the visible constraint (i.e., $o, o' \in$ OBJECTS) as well as an observation place defined by the pair $(\text{dim}, \text{dir}) \in \text{FROM}$. The object $o$ is masked by the object $o'$ according to the observation place $(\text{dim}, \text{dir})$ if there exist two shifted boxes $s$ and $s'$ respectively associated with $o$ and $o'$ such that conditions A, B, C, D and E all hold:

- (A) $o'.\text{duration} > 0 \land o'.\text{duration} > 0 \land o.\text{end} > o'.\text{start} \land o'.\text{end} > o.\text{start}$ (i.e., the time intervals associated with $o$ and $o'$ intersect).
- (B) Discarding dimension $\text{dim}$, $s$ and $s'$ intersect in all dimensions specified by $\text{DIMS}$ (i.e., objects $o$ and $o'$ are in vis-à-vis).
- (C) If $\text{dir} = 0$
  
  then $o.x[\text{dim}] + s.t[\text{dim}] \geq o'.x[\text{dim}] + s'.t[\text{dim}] + s'.l[\text{dim}]$
  
  else $o'.x[\text{dim}] + s'.t[\text{dim}] \geq o.x[\text{dim}] + s.t[\text{dim}] + s.l[\text{dim}]$ (i.e., in dimension $\text{dim}$ $o$ and $o'$ are ordered in the wrong way according to direction $\text{dir}$).
- (D) $o.\text{start} > o'.\text{start} \lor o.\text{end} < o'.\text{end}$ (i.e., instants $o.\text{start}$ or $o.\text{end}$ are located within interval $[o'.\text{start}, o'.\text{end}]$; we consider also condition A).
- (E) The observation place $(\text{dim}, \text{dir})$ occurs within the list of visible faces associated with the face attribute $f$ of the shifted box $s$ (i.e., the pair $(\text{dim}, \text{dir})$ is a potentially visible face of $o$).

**Definition 2.** Consider an object $o$ of the collection OBJECTS as well as a possible observation place defined by the pair $(\text{dim}, \text{dir})$. The object $o$ is masked according to the observation place $(\text{dim}, \text{dir})$ if and only if at least one of the following conditions holds:

- No shifted box associated with $o$ has the pair $(\text{dim}, \text{dir})$ as one of its potentially visible face.
- The object $o$ is masked according to the possible observation place $(\text{dim}, \text{dir})$ by another object $o'$.

Figures 5.642 and 5.643 respectively illustrate Definition 1 in the context of an observation place (depicted by a triangle) equal to the pair $(\text{dim} = 0, \text{dir} = 1)$. Note that, in the context of Figure 5.643, as the $\text{DIMS}$ parameter of the visible constraint only mentions dimension 0 (and not dimension 1), one object may be masked by another object even if the two objects do not intersect in any dimension: i.e., only their respective ordering in the dimension $\text{dim} = 0$ as well as their positions in time matter.

**Definition 3.** Consider an object $o$ of the collection OBJECTS as well as a possible observation place defined by the pair $(\text{dim}, \text{dir})$. The object $o$ is masked according to the observation place $(\text{dim}, \text{dir})$ if and only if at least one of the following conditions holds:

- No shifted box associated with $o$ has the pair $(\text{dim}, \text{dir})$ as one of its potentially visible face.
- The object $o$ is masked according to the possible observation place $(\text{dim}, \text{dir})$ by another object $o'$.

**Definition 4.** An object of the collection OBJECTS constraint is masked according to a set of possible observation places FROM if it is masked according to each observation place of FROM.
Figure 5.642: Illustration of Definition 1: (I,II) the case where an object $o$ is masked by an object $o'$ according to dimensions $\{0,1\}$ and to the observation place $\langle \dim = 0, \text{dir} = 1 \rangle$ because (A) $o$ and $o'$ intersect in time, (B) $o$ and $o'$ intersect in dimension 1, (C) $o$ and $o'$ are not well ordered according to the observation place, (D) there exists an instant where $o'$ if present (but not $o$) and (E) $\langle \dim = 0, \text{dir} = 1 \rangle$ is a potentially visible face of $o$; (III,IV) the case where an object $o$ is not masked by an object $o'$ according to the observation place $\langle \dim = 0, \text{dir} = 1 \rangle$. 

(A) $o$ and $o'$ intersect in time,
(B) $o$ and $o'$ intersect in dimension 1,
(C) $o$ and $o'$ are not well ordered according to the observation place,
(D) there exists an instant where $o'$ if present (but not $o$) and
(E) $\langle \dim = 0, \text{dir} = 1 \rangle$ is a potentially visible face of $o$; (II) the case where an object $o$ is masked by an object $o'$ according to dimensions $\{0,1\}$ and to the observation place $\langle \dim = 0, \text{dir} = 1 \rangle$ because (A) $o$ and $o'$ intersect in time, (B) $o$ and $o'$ intersect in dimension 1, (C) $o$ and $o'$ are not well ordered according to the observation place, (D) there exists an instant where $o'$ if present (but not $o$) and (E) $\langle \dim = 0, \text{dir} = 1 \rangle$ is a potentially visible face of $o$; (III) the case where an object $o$ is not masked by an object $o'$ according to the observation place $\langle \dim = 0, \text{dir} = 1 \rangle$. 

(A) Even though $o$ and $o'$ intersect in time,
(B) and even though $o$ and $o'$ intersect in dimension 1,
(C) even though, in dimension 1, $o$ starts after the end of $o$,
(D) and even though $\langle \dim = 0, \text{dir} = 1 \rangle$ is a potentially visible face of $o$.

condition (B) does not hold.

condition does not hold.

condition (B) does not hold.

condition (C) does not hold.

condition (C) does not hold.
Figure 5.643: Illustration of Definition 1: the case where an object $o$ is masked by an object $o'$ according to dimension 0 and to the observation place $\langle \text{dim} = 0, \text{dir} = 1 \rangle$ because: (A) $o$ and $o'$ intersect in time, (C) $o$ and $o'$ are not well ordered according to the observation place and (D) there exists an instant where $o'$ if present (but not $o$) and (E) $\langle \text{dim} = 0, \text{dir} = 1 \rangle$ is a potentially visible face of $o$. 

(A) $o$ and $o'$ intersect in time,
(C) in dimension 0, $o'$ starts after the end of $o$,
(D) the end in time of $o$ is located before the end in time of $o'$,
(E) $\langle \text{dim} = 0, \text{dir} = 1 \rangle$ is a potentially visible face of $o$. 

visible[2, [0], \langle \text{dim} = 0, \text{dir} = 1 \rangle, 
\langle \text{oid} = o \text{ sid} = 1 \text{ x} = \langle 2, 1 \rangle \text{ start} = 1 \text{ duration} = 8 \text{ end} = 9, 
\langle \text{oid} = o' \text{ sid} = 2 \text{ x} = \langle 4, 3 \rangle \text{ start} = 1 \text{ duration} = 15 \text{ end} = 16, 
\langle \text{sid} = 1 \text{ l} = \langle 0, 0 \rangle \text{ f} = \langle \text{dim} = 0 \text{ dir} = 1 \rangle, 
\text{sid} = 2 \text{ l} = \langle 0, 0 \rangle \text{ f} = \langle \text{dim} = 0 \text{ dir} = 1 \rangle \rangle)$ 

$o$ is masked by $o'$ according to $\langle \text{dim} = 0, \text{dir} = 1 \rangle$ since:
(A) $o$ and $o'$ intersect in time,
(C) in dimension 0, $o'$ starts after the end of $o$,
(D) the end in time of $o$ is located before the end in time of $o'$;
(E) $\langle \text{dim} = 0, \text{dir} = 1 \rangle$ is a potentially visible face of $o$. 

\(time\ interval \ [9, 15]\) 

\(time\ interval \ [1, 9]\)
We are now in position to define the visible constraint.

**Definition 5.** Given a $\text{visible}(K, \text{DIMS}, \text{FROM}, \text{OBJECTS}, \text{SBOXES})$ constraint, the visible constraint holds if none of the objects of $\text{OBJECTS}$ is masked according to the dimensions of $\text{DIMS}$ and to the set of possible observation places defined by $\text{FROM}$. 
Typical

| OBJECTS | > 1

Symmetries

- Items of OBJECTS are permutable.
- Items of SBOXES are permutable.

Usage

We now give several typical concrete uses of the visible constraint, which all mention the diffst as well as the visible constraints:

- Figure 5.644 corresponds to a ship loading problem where containers are piled within a ship by a crane each time the ship visits a given harbour. In this context we have first to express the fact that a container can only be placed on top of an already placed container and second, that a container can only be taken away if no container is placed on top of it. These two conditions are expressed by one single visible constraint for which the DIMS parameter mentions all three dimensions of the placement space and the FROM parameter mentions the pair \(\langle \text{dim} = 2, \text{dir} = 1 \rangle\) as its unique observation place. In addition we also use a diffst constraint for expressing non-overlapping.

\[
\begin{align*}
\text{visible}(3, \{0,1,2\}, \langle \text{dim} = 2, \text{dir} = 1 \rangle, \\
\text{oid} = 1 & \text{ sid} = 1 x = \langle 1,1,1 \rangle \text{ start} = 0 \text{ duration} = 17 \text{ end} = 17, \\
\text{oid} = 2 & \text{ sid} = 1 x = \langle 1,1,3 \rangle \text{ start} = 0 \text{ duration} = 8 \text{ end} = 8, \\
\text{oid} = 3 & \text{ sid} = 1 x = \langle 4,1,1 \rangle \text{ start} = 0 \text{ duration} = 8 \text{ end} = 8, \\
\text{oid} = 4 & \text{ sid} = 1 x = \langle 1,1,3 \rangle \text{ start} = 8 \text{ duration} = 9 \text{ end} = 17, \\
\text{oid} = 5 & \text{ sid} = 1 x = \langle 4,1,1 \rangle \text{ start} = 8 \text{ duration} = 16 \text{ end} = 24, \\
\text{oid} = 6 & \text{ sid} = 1 x = \langle 1,1,1 \rangle \text{ start} = 17 \text{ duration} = 7 \text{ end} = 24, \\
\text{from} & = \langle 0,0,0 \rangle \text{ t} = \langle 2,4,2 \rangle \text{ f} = \langle \text{dim} = 2, \text{dir} = 1 \rangle 
\end{align*}
\]

Figure 5.644: Illustration of the ship loading problem
Figure 5.645 corresponds to a container loading/unloading problem in the context of a pick-up delivery problem where the loading/unloading takes place with respect to the front door of the container. Beside the diffn constraint used for expressing non-overlapping, we use two distinct visible constraints:

- The first visible constraint takes care of the location of the front door of the container (each object \( o \) has to be loaded/unloaded without moving around any other object, i.e., objects that are in the vis-à-vis of \( o \) according to the front door of the container). This is expressed by one single visible constraint for which the \( \text{DIMS} \) parameter mentions all three dimensions of the placement space and the \( \text{FROM} \) parameter mentions the pair \( (\text{dim} = 1, \text{dir} = 0) \) as its unique observation place.

- The second visible constraint takes care of the gravity dimension (i.e., each object that has to be loaded should not be put under another object, and reciprocally each object that has to be unloaded should not be located under another object). This is expressed by the same visible constraint that was used for the ship loading problem, i.e., a visible constraint for which the \( \text{DIMS} \) parameter mentions all three dimensions of the placement space and the \( \text{FROM} \) parameter mentions the pair \( (\text{dim} = 2, \text{dir} = 1) \) as its unique observation place.

Figure 5.646 corresponds to a pallet loading problem where one has to place six objects on a pallet. Each object corresponds to a parallelepiped that has a bar code on one of its four sides (i.e., the sides that are different from the top and the bottom of the parallelepiped). If, for some reason, an object has no bar code then we simply remove it from the objects that will be passed to the visible constraint: this is for instance the case of the sixth object. In this context the constraint to enforce (beside the non-overlapping constraint between the parallelepipeds that are assigned to a same pallet) is the fact that the bar code of each object should be visible (i.e., visible from one of the four sides of the pallet). This is expressed by the visible constraint given in Part (F) of Figure 5.646.

**Remark**

The visible constraint is a generalisation of the accessibility constraint initially introduced in the context of the diffn constraint.

**See also**

- **common keyword:** diffn (geometrical constraint),
- geost, geost_time (geometrical constraint,sweep),
- non_overlap_sboxes (geometrical constraint).

**Keywords**

- **constraint type:** decomposition, predefined constraint.
- **filtering:** sweep.
- **geometry:** geometrical constraint.
visible(3, {0,1,2}, <dim−1 dir−0>,
oid−1 sid−1 x−<1,2,3> start−0 duration−8 end−8,
oid−2 sid−2 x−<1,3,3> start−0 duration−8 end−8,
oid−3 sid−3 x−<1,1,1> start−0 duration−17 end−17,
oid−4 sid−4 x−<4,1,1> start−0 duration−17 end−17,
oid−5 sid−5 x−<1,2,3> start−8 duration−9 end−17,
oid−6 sid−6 x−<3,1,1> start−8 duration−12 end−24,
oid−7 sid−3 x−<1,1,1> start−17 duration−7 end−24>,
oid−1 t−<0,0,0> l−<2,1,1> f−<dim−1 dir−0, dim−2 dir−1>,
sid−2 t−<0,0,0> l−<2,2,2> f−<dim−1 dir−0, dim−2 dir−1>,
sid−3 t−<0,0,0> l−<2,4,2> f−<dim−1 dir−0, dim−2 dir−1>,
sid−4 t−<0,0,0> l−<2,4,1> f−<dim−1 dir−0, dim−2 dir−1>,
sid−5 t−<0,0,0> l−<2,3,1> f−<dim−1 dir−0, dim−2 dir−1>,
sid−6 t−<0,0,0> l−<1,2,2> f−<dim−1 dir−0, dim−2 dir−1>)

visible(3, {0,1,2}, <dim−2 dir−1>,
oid−1 sid−1 x−<1,2,3> start−0 duration−8 end−8,
oid−2 sid−2 x−<1,3,3> start−0 duration−8 end−8,
oid−3 sid−3 x−<1,1,1> start−0 duration−17 end−17,
oid−4 sid−4 x−<4,1,1> start−0 duration−17 end−17,
oid−5 sid−5 x−<1,2,3> start−8 duration−9 end−17,
oid−6 sid−6 x−<3,1,1> start−8 duration−12 end−24,
oid−7 sid−3 x−<1,1,1> start−17 duration−7 end−24>,
oid−1 t−<0,0,0> l−<2,1,1> f−<dim−1 dir−0, dim−2 dir−1>,
sid−2 t−<0,0,0> l−<2,2,2> f−<dim−1 dir−0, dim−2 dir−1>,
sid−3 t−<0,0,0> l−<2,4,2> f−<dim−1 dir−0, dim−2 dir−1>,
sid−4 t−<0,0,0> l−<2,4,1> f−<dim−1 dir−0, dim−2 dir−1>,
sid−5 t−<0,0,0> l−<2,3,1> f−<dim−1 dir−0, dim−2 dir−1>,
sid−6 t−<0,0,0> l−<1,2,2> f−<dim−1 dir−0, dim−2 dir−1>)

Figure 5.645: Illustration of the pick-up delivery problem
Figure 5.646: Illustration of the pallet loading problem
5.380  weighted_partial_alldiff

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>[381, page 71]</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>weighted_partial_alldiff(VARIABLES, UNDEFINED, VALUES, COST)</td>
<td></td>
</tr>
<tr>
<td>Synonyms</td>
<td>weighted_partial_alldifferent, weighted_partial_alldistinct, wpa.</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VARIABLES : collection(var-dvar)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UNDEFINED : int</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VALUES : collection(val-int, weight-int)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COST : dvar</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>required(VARIABLES, var)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[VALUES] &gt; 0</td>
</tr>
<tr>
<td></td>
<td>required(VALUES, [val, weight])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>in_attr(VARIABLES, var, VALUES, val)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>distinct(VALUES, val)</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>All variables of the VARIABLES collection that are not assigned to value UNDEFINED must have pairwise distinct values from the val attribute of the VALUES collection. In addition COST is the sum of the weight attributes associated with the values assigned to the variables of VARIABLES. Within the VALUES collection, value UNDEFINED must be explicitly defined with a weight of 0.</td>
<td></td>
</tr>
</tbody>
</table>

```
\begin{align*}
\{ & \text{var} - 4, \\
& \text{var} - 0, \\
& \{ \text{var} - 1, \\
& \text{var} - 2, \} , 0, \\
& \text{var} - 0, \\
& \text{var} - 0, \\
& \text{val} - 0 \text{ weight} - 0, \\
& \text{val} - 1 \text{ weight} - 2, \\
& \{ \text{val} - 2 \text{ weight} - 1, \\
& \text{val} - 4 \text{ weight} - 7, \} , 8, \\
& \text{val} - 5 \text{ weight} - 8, \\
& \text{val} - 6 \text{ weight} - 2 \\
\end{align*}
```

The weighted_partial_alldiff constraint holds since:

- No value, except value UNDEFINED = 0, is used more than once.
- COST = 8 is equal to the sum of the weights 2, -1 and 7 of the values 1, 2 and 4 assigned to the variables of VARIABLES = (4, 0, 1, 2, 0, 0).
Typical

\[ |\text{VARIABLES}| > 0 \]
\[ \text{atleast}(1, \text{VARIABLES}, \text{UNDEFINED}) \]
\[ |\text{VARIABLES}| \leq |\text{VALUES}| + 2 \]

Symmetries

- Items of \text{VARIABLES} are permutable.
- Items of \text{VALUES} are permutable.
- All occurrences of two distinct values in \text{VARIABLES} or \text{VALUES} that are both different from \text{UNDEFINED} can be swapped; all occurrences of a value in \text{VARIABLES} or \text{VALUES} that is different from \text{UNDEFINED} can be renamed to any unused value that is also different from \text{UNDEFINED}.

Arg. properties

Functional dependency: \text{COST} determined by \text{VARIABLES} and \text{VALUES}.

Usage

In his PhD thesis [381, pages 71–72], Sven Thiel describes the following three potential scenarios of the \text{weighted partial alldiff} constraint:

- Given a set of tasks (i.e., the items of the \text{VARIABLES} collection), assign to each task a resource (i.e., an item of the \text{VALUES} collection). Except for the resource associated with value \text{UNDEFINED}, every resource can be used at most once. The cost of a resource is independent from the task to which the resource is assigned. The cost of value \text{UNDEFINED} is equal to 0. The total cost \text{COST} of an assignment corresponds to the sum of the costs of the resources effectively assigned to the tasks. Finally we impose an upper bound on the total cost.

- Given a set of persons (i.e., the items of the \text{VARIABLES} collection), select for each person an offer (i.e., an item of the \text{VALUES} collection). Except for the offer associated with value \text{UNDEFINED}, every offer should be selected at most once. The profit associated with an offer is independent from the person that selects the offer. The profit of value \text{UNDEFINED} is equal to 0. The total benefit \text{COST} is equal to the sum of the profits of the offers effectively selected. In addition we impose a lower bound on the total benefit.

- The last scenario deals with an application to an over-constraint problem involving the \text{alldifferent} constraint. Allowing some variables to take an "undefined" value is done by setting all weights of all the values different from \text{UNDEFINED} to 1. As a consequence all variables assigned to a value different from \text{UNDEFINED} will have to take distinct values. The \text{COST} variable allows to control the number of such variables.

Remark

It was shown in [381, page 104] that, finding out whether the \text{weighted partial alldiff} constraint has a solution or not is NP-hard. This was achieved by reduction from \text{subset sum}.

Algorithm

A filtering algorithm is given in [381, pages 73–104]. After showing that, deciding whether the \text{weighted partial alldiff} has a solution is NP-complete, [381, pages 105–106] gives the following results of his filtering algorithm with respect to consistency under the 3 scenarios previously described:

- For scenario 1, if there is no restriction of the lower bound of the \text{COST} variable, the filtering algorithm achieves arc-consistency for all variables of the \text{VARIABLES} collection (but not for the \text{COST} variable itself).
• For scenario 2, if there is no restriction of the upper bound of the COST variable, the filtering algorithm achieves arc-consistency for all variables of the VARIABLES collection (but not for the COST variable itself).

• Finally, for scenario 3, the filtering algorithm achieves arc-consistency for all variables of the VARIABLES collection as well as for the COST variable.

See also

attached to cost variant: alldifferent, alldifferent_except_0.

common keyword:  
global_cardinality_with_costs (weighted assignment),
minimum_weight_alldifferent (cost filtering constraint, weighted assignment),
soft_alldifferent_var (soft constraint),
sum_of_weights_of_distinct_values (weighted assignment).

Keywords

application area: assignment.

characteristic of a constraint: all different, joker value.

complexity: subset sum.

constraint type: soft constraint, relaxation.

filtering: cost filtering constraint.

modelling: functional dependency.

problems: weighted assignment.
Arc input(s) VARIABLES VALUES
Arc generator $PRODUCT \rightarrow \textit{collection}(\text{variables, values})$
Arc arity 2
Arc constraint(s) • $\text{variables.var} \neq \text{UNDEFINED}$
• $\text{variables.var} = \text{values.val}$
Graph property(ies) • $\text{MAX_ID} \leq 1$
• $\text{SUM(VALUES, weight)} = \text{COST}$

Graph model Parts (A) and (B) of Figure 5.647 respectively show the initial and final graph associated with the Example slot. Since we also use the $\text{SUM}$ graph property we show the vertices of the final graph from which we compute the total cost in a box.

Figure 5.647: Initial and final graph of the $\text{weighted_partial_alldiff}$ constraint
## 5.381 xor

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LINKS</th>
<th>AUTOMATON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin</td>
<td>Logic</td>
<td></td>
</tr>
<tr>
<td>Constraint</td>
<td>xor(VAR, VARIABLES)</td>
<td></td>
</tr>
<tr>
<td>Synonym</td>
<td>rel.</td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>VAR : dvar, VARIABLES : collection(var−dvar)</td>
<td></td>
</tr>
<tr>
<td>Restrictions</td>
<td>VAR ≥ 0, VAR ≤ 1, (</td>
<td>VARIABLES</td>
</tr>
<tr>
<td>Purpose</td>
<td>Let VARIABLES be a collection of 0-1 variables VAR₁, VAR₂. Enforce VAR = (VAR₁ ≠ VAR₂).</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>(0, (0, 0)), (1, (0, 1)), (1, (1, 0)), (0, (1, 1))</td>
<td></td>
</tr>
<tr>
<td>Symmetry</td>
<td>Items of VARIABLES are permutable.</td>
<td></td>
</tr>
<tr>
<td>Arg. properties</td>
<td>Functional dependency: VAR determined by VARIABLES.</td>
<td></td>
</tr>
<tr>
<td>Systems</td>
<td>reifiedXor in Choco, rel in Gecode, xorbool in JaCoP, #\ in SICStus.</td>
<td></td>
</tr>
<tr>
<td>See also</td>
<td>common keyword: and, equivalent, imply, nand, nor, or (Boolean constraint).</td>
<td></td>
</tr>
<tr>
<td>Keywords</td>
<td>characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>constraint arguments: pure functional dependency.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>constraint network structure: Berge-acyclic constraint network.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>constraint type: Boolean constraint.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>filtering: arc-consistency.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>modelling: functional dependency.</td>
<td></td>
</tr>
</tbody>
</table>
Automaton

Figure 5.648 depicts the automaton associated with the xor constraint. To the first argument VAR of the xor constraint corresponds the first signature variable. To each variable VAR of the second argument VARIABLES of the xor constraint corresponds the next signature variable. There is no signature constraint.

Figure 5.648: Automaton of the xor constraint

Figure 5.649: Hypergraph of the reformulation corresponding to the automaton of the xor constraint
Appendix A

Legend for the Description

This section provides the list of restrictions, of arc generators, of graph parameters and of set generators sorted in alphabetic order with the page where they are defined.
Restrictions:

- Term₁ Comparison Term₂ p. 14
- distinct p. 11
- in_attr p. 10
- in_list p. 10
- increasing_seq p. 11
- non_increasing_size p. 12
- required p. 12
- require_at_least p. 13
- same_size p. 13

Arc generators:

- CHAIN p. 53
- CIRCUIT p. 53
- CLIQUE p. 53
- CLIQUE(C) p. 54
- CYCLE p. 54
- GRID p. 54
- LOOP p. 54
- PATH p. 54
- PATH₁ p. 55
- PATH_N p. 55
- PRODUCT p. 55
- PRODUCT(C) p. 55
- SELF p. 55
- SYMMETRIC_PRODUCT p. 55
- SYMMETRIC_PRODUCT(C) p. 55
- VOID p. 56

Graph parameters:

- DISTANCE p. 69
- MAX_DRG p. 60
- MAX_ID p. 60
- MAX_NCC p. 61
- MAX_NSCELL p. 61
- MAX_OD p. 61
- MIN_DRG p. 61
- MIN_ID p. 61
- MIN_NCC p. 62
- MIN_NSCELL p. 62
- MIN_OD p. 62
- NARC p. 62
- NARC_NO_LOOP p. 63
- NCELL p. 63
- NSCELL p. 63
- NSINK p. 63
- NSINK_NSOURCE p. 64
- NSOURCE p. 64
- NTREE p. 64
- NVERTEX p. 65
- ORDER p. 65
- PATH_FROM_TO p. 66
- PROD p. 67
- RANGE p. 67
- RANGE_DRG p. 65
- RANGE_NCC p. 65
- RANGE_NSCELL p. 65
- SUM p. 68
- SUM_WEIGHT_ARC p. 69

Set generators:

- ALL_VERTICES p. 75
- CC p. 75
- PATH_LENGTH p. 75
- PRED p. 75
- SUCC p. 75
Appendix B

Electronic Constraint Catalogue

Contents

B.1  abs_value  ............................................. 1977
B.2  all_differ_from_at_least_k_pos  .................................... 1978
B.3  all_equal  ............................................ 1981
B.4  all_incomparable  .......................................... 1983
B.5  all_min_dist  ........................................... 1986
B.6  alldifferent  ............................................ 1988
B.7  alldifferent_between_sets  ...................................... 1991
B.8  alldifferent_consecutive_values  .................................. 1993
B.9  alldifferent_cst  .......................................... 1995
B.10  alldifferent_except_0  ......................................... 1997
B.11  alldifferent_interval  .......................................... 2000
B.12  alldifferent_modulo  .......................................... 2002
B.13  alldifferent_on_intersection  .................................... 2004
B.14  alldifferent_partition  .......................................... 2007
B.15  alldifferent_same_value  ....................................... 2009
B.16  allperm  .................................................. 2011
B.17  among  ................................................. 2013
B.18  among_diff_0  ............................................. 2016
B.19  among_interval  .............................................. 2019
B.20  among_low_up  ............................................. 2022
B.21  among_modulo  ............................................. 2026
B.22  among_seq  ............................................... 2029
B.23  among_var  ............................................... 2032
B.24  and  ...................................................... 2035
B.25  arith  .................................................... 2037
B.26  arith_or  .................................................. 2040
B.27  arith_sliding  .............................................. 2045

1967
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.28</td>
<td>assign_and_counts</td>
<td>2050</td>
</tr>
<tr>
<td>B.29</td>
<td>assign_and_nvalues</td>
<td>2053</td>
</tr>
<tr>
<td>B.30</td>
<td>atleast</td>
<td>2056</td>
</tr>
<tr>
<td>B.31</td>
<td>atleast_nvalue</td>
<td>2059</td>
</tr>
<tr>
<td>B.32</td>
<td>atleast_nvector</td>
<td>2061</td>
</tr>
<tr>
<td>B.33</td>
<td>atmost</td>
<td>2063</td>
</tr>
<tr>
<td>B.34</td>
<td>atmost1</td>
<td>2065</td>
</tr>
<tr>
<td>B.35</td>
<td>atmost_nvalue</td>
<td>2066</td>
</tr>
<tr>
<td>B.36</td>
<td>atmost_nvector</td>
<td>2068</td>
</tr>
<tr>
<td>B.37</td>
<td>balance</td>
<td>2070</td>
</tr>
<tr>
<td>B.38</td>
<td>balance_cycle</td>
<td>2072</td>
</tr>
<tr>
<td>B.39</td>
<td>balance_interval</td>
<td>2074</td>
</tr>
<tr>
<td>B.40</td>
<td>balance_modulo</td>
<td>2076</td>
</tr>
<tr>
<td>B.41</td>
<td>balance_partition</td>
<td>2078</td>
</tr>
<tr>
<td>B.42</td>
<td>balance_path</td>
<td>2080</td>
</tr>
<tr>
<td>B.43</td>
<td>balance_tree</td>
<td>2083</td>
</tr>
<tr>
<td>B.44</td>
<td>between_min_max</td>
<td>2086</td>
</tr>
<tr>
<td>B.45</td>
<td>bin_packing</td>
<td>2089</td>
</tr>
<tr>
<td>B.46</td>
<td>bin_packing_capa</td>
<td>2091</td>
</tr>
<tr>
<td>B.47</td>
<td>binary_tree</td>
<td>2093</td>
</tr>
<tr>
<td>B.48</td>
<td>bipartite</td>
<td>2095</td>
</tr>
<tr>
<td>B.49</td>
<td>calendar</td>
<td>2096</td>
</tr>
<tr>
<td>B.50</td>
<td>cardinality_atleast</td>
<td>2100</td>
</tr>
<tr>
<td>B.51</td>
<td>cardinality_atmost</td>
<td>2103</td>
</tr>
<tr>
<td>B.52</td>
<td>cardinality_atmost_partition</td>
<td>2106</td>
</tr>
<tr>
<td>B.53</td>
<td>change</td>
<td>2108</td>
</tr>
<tr>
<td>B.54</td>
<td>change_continuity</td>
<td>2111</td>
</tr>
<tr>
<td>B.55</td>
<td>change_pair</td>
<td>2119</td>
</tr>
<tr>
<td>B.56</td>
<td>change_partition</td>
<td>2128</td>
</tr>
<tr>
<td>B.57</td>
<td>change_vectors</td>
<td>2130</td>
</tr>
<tr>
<td>B.58</td>
<td>circuit</td>
<td>2133</td>
</tr>
<tr>
<td>B.59</td>
<td>circuit_cluster</td>
<td>2135</td>
</tr>
<tr>
<td>B.60</td>
<td>circular_change</td>
<td>2137</td>
</tr>
<tr>
<td>B.61</td>
<td>clause_and</td>
<td>2140</td>
</tr>
<tr>
<td>B.62</td>
<td>clause_or</td>
<td>2142</td>
</tr>
<tr>
<td>B.63</td>
<td>clique</td>
<td>2144</td>
</tr>
<tr>
<td>B.64</td>
<td>colored_matrix</td>
<td>2146</td>
</tr>
<tr>
<td>B.65</td>
<td>coloured_cumulative</td>
<td>2149</td>
</tr>
<tr>
<td>B.66</td>
<td>coloured_cumulatives</td>
<td>2154</td>
</tr>
<tr>
<td>B.67</td>
<td>common</td>
<td>2160</td>
</tr>
<tr>
<td>B.68</td>
<td>common_interval</td>
<td>2162</td>
</tr>
<tr>
<td>B.69</td>
<td>common_modulo</td>
<td>2165</td>
</tr>
<tr>
<td>B.70</td>
<td>common_partition</td>
<td>2167</td>
</tr>
</tbody>
</table>
B.71 compare_and_count ...................................... 2170
B.72 cond_lex_cost ........................................... 2173
B.73 cond_lex_greater ...................................... 2175
B.74 cond_lex_greatereq ................................... 2177
B.75 cond_lex_less ........................................... 2179
B.76 cond_lex_leqq .......................................... 2181
B.77 connect_points ........................................ 2183
B.78 connected ............................................. 2186
B.79 consecutive_groups_of_ones ............................. 2187
B.80 consecutive_values ..................................... 2190
B.81 contains_sboxes ......................................... 2192
B.82 correspondence ......................................... 2195
B.83 count ..................................................... 2197
B.84 counts .................................................... 2200
B.85 coveredby_sboxes ....................................... 2203
B.86 covers_sboxes ........................................... 2207
B.87 crossing .................................................. 2211
B.88 cumulative .............................................. 2213
B.89 cumulative_convex ...................................... 2216
B.90 cumulative_product ..................................... 2218
B.91 cumulative_two_d ...................................... 2222
B.92 cumulative_with_level_of_priority ..................... 2225
B.93 cumulatives ............................................. 2228
B.94 cutset .................................................... 2232
B.95 cycle ....................................................... 2234
B.96 cycle_card_on_path ..................................... 2236
B.97 cycle_or_accessibility .................................. 2238
B.98 cycle_resource .......................................... 2240
B.99 cyclic_change ........................................... 2243
B.100 cyclic_change_joker ................................... 2247
B.101 dag ....................................................... 2252
B.102 decreasing ............................................. 2254
B.103 deepest_valley ........................................ 2256
B.104 derangement ............................................ 2258
B.105 differ_from_at_least_k_pos ............................ 2260
B.106 diffn ..................................................... 2263
B.107 diffn_column ........................................... 2267
B.108 diffn_include .......................................... 2269
B.109 discrepancy ............................................. 2272
B.110 disj ....................................................... 2274
B.111 disjoint .................................................. 2276
B.112 disjoint_sboxes ........................................ 2278
B.113 disjoint_tasks .......................................... 2282
<table>
<thead>
<tr>
<th>B.114</th>
<th>disjunctive</th>
<th>2285</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.115</td>
<td>disjunctive_or_same_end</td>
<td>2287</td>
</tr>
<tr>
<td>B.116</td>
<td>disjunctive_or_same_start</td>
<td>2289</td>
</tr>
<tr>
<td>B.117</td>
<td>distance</td>
<td>2291</td>
</tr>
<tr>
<td>B.118</td>
<td>distance_between</td>
<td>2292</td>
</tr>
<tr>
<td>B.119</td>
<td>distance_change</td>
<td>2298</td>
</tr>
<tr>
<td>B.120</td>
<td>divisible</td>
<td>2304</td>
</tr>
<tr>
<td>B.121</td>
<td>divisible or</td>
<td>2305</td>
</tr>
<tr>
<td>B.122</td>
<td>dom_reachability</td>
<td>2306</td>
</tr>
<tr>
<td>B.123</td>
<td>domain</td>
<td>2308</td>
</tr>
<tr>
<td>B.124</td>
<td>domain_constraint</td>
<td>2310</td>
</tr>
<tr>
<td>B.125</td>
<td>elem</td>
<td>2313</td>
</tr>
<tr>
<td>B.126</td>
<td>elem_from_to</td>
<td>2316</td>
</tr>
<tr>
<td>B.127</td>
<td>element</td>
<td>2319</td>
</tr>
<tr>
<td>B.128</td>
<td>element_greatereq</td>
<td>2322</td>
</tr>
<tr>
<td>B.129</td>
<td>element_lesseq</td>
<td>2325</td>
</tr>
<tr>
<td>B.130</td>
<td>element_matrix</td>
<td>2328</td>
</tr>
<tr>
<td>B.131</td>
<td>element_product</td>
<td>2332</td>
</tr>
<tr>
<td>B.132</td>
<td>element_sparse</td>
<td>2334</td>
</tr>
<tr>
<td>B.133</td>
<td>elementn</td>
<td>2338</td>
</tr>
<tr>
<td>B.134</td>
<td>elements</td>
<td>2341</td>
</tr>
<tr>
<td>B.135</td>
<td>elements_alldifferent</td>
<td>2343</td>
</tr>
<tr>
<td>B.136</td>
<td>elements_sparse</td>
<td>2345</td>
</tr>
<tr>
<td>B.137</td>
<td>eq</td>
<td>2348</td>
</tr>
<tr>
<td>B.138</td>
<td>eq_cst</td>
<td>2349</td>
</tr>
<tr>
<td>B.139</td>
<td>eq_set</td>
<td>2350</td>
</tr>
<tr>
<td>B.140</td>
<td>equal_sboxes</td>
<td>2351</td>
</tr>
<tr>
<td>B.141</td>
<td>equivalent</td>
<td>2354</td>
</tr>
<tr>
<td>B.142</td>
<td>exactly</td>
<td>2356</td>
</tr>
<tr>
<td>B.143</td>
<td>gcd</td>
<td>2358</td>
</tr>
<tr>
<td>B.144</td>
<td>geost</td>
<td>2359</td>
</tr>
<tr>
<td>B.145</td>
<td>geost_time</td>
<td>2362</td>
</tr>
<tr>
<td>B.146</td>
<td>geq</td>
<td>2365</td>
</tr>
<tr>
<td>B.147</td>
<td>geq_cst</td>
<td>2366</td>
</tr>
<tr>
<td>B.148</td>
<td>global_cardinality</td>
<td>2367</td>
</tr>
<tr>
<td>B.149</td>
<td>global_cardinality_low_up</td>
<td>2370</td>
</tr>
<tr>
<td>B.150</td>
<td>global_cardinality_low_up_no_loop</td>
<td>2373</td>
</tr>
<tr>
<td>B.151</td>
<td>global_cardinality_no_loop</td>
<td>2377</td>
</tr>
<tr>
<td>B.152</td>
<td>global_cardinality_with_costs</td>
<td>2380</td>
</tr>
<tr>
<td>B.153</td>
<td>global_contiguity</td>
<td>2384</td>
</tr>
<tr>
<td>B.154</td>
<td>golomb</td>
<td>2387</td>
</tr>
<tr>
<td>B.155</td>
<td>graph_crossing</td>
<td>2389</td>
</tr>
<tr>
<td>B.156</td>
<td>graph_isomorphism</td>
<td>2391</td>
</tr>
<tr>
<td>Code</td>
<td>Constraint</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>B.200</td>
<td>leq</td>
<td>2511</td>
</tr>
<tr>
<td>B.201</td>
<td>leq, cst</td>
<td>2512</td>
</tr>
<tr>
<td>B.202</td>
<td>lex2</td>
<td>2513</td>
</tr>
<tr>
<td>B.203</td>
<td>lex_alldifferent</td>
<td>2514</td>
</tr>
<tr>
<td>B.204</td>
<td>lex_between</td>
<td>2516</td>
</tr>
<tr>
<td>B.205</td>
<td>lex_chain_less</td>
<td>2521</td>
</tr>
<tr>
<td>B.206</td>
<td>lex_chain_leseq</td>
<td>2523</td>
</tr>
<tr>
<td>B.207</td>
<td>lex_different</td>
<td>2525</td>
</tr>
<tr>
<td>B.208</td>
<td>lex_equal</td>
<td>2528</td>
</tr>
<tr>
<td>B.209</td>
<td>lex_greater</td>
<td>2531</td>
</tr>
<tr>
<td>B.210</td>
<td>lex_greatereq</td>
<td>2534</td>
</tr>
<tr>
<td>B.211</td>
<td>lex_less</td>
<td>2537</td>
</tr>
<tr>
<td>B.212</td>
<td>lex_leseq</td>
<td>2540</td>
</tr>
<tr>
<td>B.213</td>
<td>lex_leseq_allperm</td>
<td>2543</td>
</tr>
<tr>
<td>B.214</td>
<td>link_set_to_bools</td>
<td>2544</td>
</tr>
<tr>
<td>B.215</td>
<td>longest_change</td>
<td>2546</td>
</tr>
<tr>
<td>B.216</td>
<td>lt</td>
<td>2549</td>
</tr>
<tr>
<td>B.217</td>
<td>map</td>
<td>2550</td>
</tr>
<tr>
<td>B.218</td>
<td>max_index</td>
<td>2552</td>
</tr>
<tr>
<td>B.219</td>
<td>max_n</td>
<td>2554</td>
</tr>
<tr>
<td>B.220</td>
<td>max_nvalue</td>
<td>2556</td>
</tr>
<tr>
<td>B.221</td>
<td>max_size_set_of_consecutive_var</td>
<td>2558</td>
</tr>
<tr>
<td>B.222</td>
<td>maximum</td>
<td>2560</td>
</tr>
<tr>
<td>B.223</td>
<td>maximum_modulo</td>
<td>2562</td>
</tr>
<tr>
<td>B.224</td>
<td>meet_sboxes</td>
<td>2564</td>
</tr>
<tr>
<td>B.225</td>
<td>min_index</td>
<td>2568</td>
</tr>
<tr>
<td>B.226</td>
<td>min_n</td>
<td>2570</td>
</tr>
<tr>
<td>B.227</td>
<td>min_nvalue</td>
<td>2572</td>
</tr>
<tr>
<td>B.228</td>
<td>min_size_set_of_consecutive_var</td>
<td>2574</td>
</tr>
<tr>
<td>B.229</td>
<td>minimum</td>
<td>2576</td>
</tr>
<tr>
<td>B.230</td>
<td>minimum_except_0</td>
<td>2578</td>
</tr>
<tr>
<td>B.231</td>
<td>minimum_greater_than</td>
<td>2581</td>
</tr>
<tr>
<td>B.232</td>
<td>minimum_modulo</td>
<td>2584</td>
</tr>
<tr>
<td>B.233</td>
<td>minimum_weight_alldifferent</td>
<td>2586</td>
</tr>
<tr>
<td>B.234</td>
<td>multi_global_contiguity</td>
<td>2588</td>
</tr>
<tr>
<td>B.235</td>
<td>multi_inter_distance</td>
<td>2590</td>
</tr>
<tr>
<td>B.236</td>
<td>nand</td>
<td>2592</td>
</tr>
<tr>
<td>B.237</td>
<td>nclass</td>
<td>2594</td>
</tr>
<tr>
<td>B.238</td>
<td>neq</td>
<td>2596</td>
</tr>
<tr>
<td>B.239</td>
<td>neq_cst</td>
<td>2597</td>
</tr>
<tr>
<td>B.240</td>
<td>nequivalence</td>
<td>2598</td>
</tr>
<tr>
<td>B.241</td>
<td>next_element</td>
<td>2600</td>
</tr>
<tr>
<td>B.242</td>
<td>next_greater_element</td>
<td>2604</td>
</tr>
<tr>
<td>B.286</td>
<td>period_vectors</td>
<td>2696</td>
</tr>
<tr>
<td>B.287</td>
<td>permutation</td>
<td>2698</td>
</tr>
<tr>
<td>B.288</td>
<td>place_in_pyramid</td>
<td>2700</td>
</tr>
<tr>
<td>B.289</td>
<td>polyomino</td>
<td>2702</td>
</tr>
<tr>
<td>B.290</td>
<td>power</td>
<td>2704</td>
</tr>
<tr>
<td>B.291</td>
<td>precedence</td>
<td>2706</td>
</tr>
<tr>
<td>B.292</td>
<td>product_ctr</td>
<td>2708</td>
</tr>
<tr>
<td>B.293</td>
<td>proper_forest</td>
<td>2710</td>
</tr>
<tr>
<td>B.294</td>
<td>range_ctr</td>
<td>2712</td>
</tr>
<tr>
<td>B.295</td>
<td>relaxed_sliding_sum</td>
<td>2714</td>
</tr>
<tr>
<td>B.296</td>
<td>remainder</td>
<td>2717</td>
</tr>
<tr>
<td>B.297</td>
<td>roots</td>
<td>2718</td>
</tr>
<tr>
<td>B.298</td>
<td>same</td>
<td>2719</td>
</tr>
<tr>
<td>B.299</td>
<td>same_and_global_cardinality</td>
<td>2721</td>
</tr>
<tr>
<td>B.300</td>
<td>same_and_global_cardinality_low_up</td>
<td>2724</td>
</tr>
<tr>
<td>B.301</td>
<td>same_intersection</td>
<td>2727</td>
</tr>
<tr>
<td>B.302</td>
<td>same_interval</td>
<td>2730</td>
</tr>
<tr>
<td>B.303</td>
<td>same_modulo</td>
<td>2732</td>
</tr>
<tr>
<td>B.304</td>
<td>same_partition</td>
<td>2734</td>
</tr>
<tr>
<td>B.305</td>
<td>same_sign</td>
<td>2737</td>
</tr>
<tr>
<td>B.306</td>
<td>scalar_product</td>
<td>2738</td>
</tr>
<tr>
<td>B.307</td>
<td>sequence_folding</td>
<td>2741</td>
</tr>
<tr>
<td>B.308</td>
<td>set_value_precede</td>
<td>2744</td>
</tr>
<tr>
<td>B.309</td>
<td>shift</td>
<td>2745</td>
</tr>
<tr>
<td>B.310</td>
<td>sign_of</td>
<td>2749</td>
</tr>
<tr>
<td>B.311</td>
<td>size_max_seq_alldifferent</td>
<td>2750</td>
</tr>
<tr>
<td>B.312</td>
<td>size_max_starting_seq_alldifferent</td>
<td>2752</td>
</tr>
<tr>
<td>B.313</td>
<td>sliding_card_skip0</td>
<td>2755</td>
</tr>
<tr>
<td>B.314</td>
<td>sliding_distribution</td>
<td>2758</td>
</tr>
<tr>
<td>B.315</td>
<td>sliding_sum</td>
<td>2761</td>
</tr>
<tr>
<td>B.316</td>
<td>sliding_time_window</td>
<td>2764</td>
</tr>
<tr>
<td>B.317</td>
<td>sliding_time_window_from_start</td>
<td>2766</td>
</tr>
<tr>
<td>B.318</td>
<td>sliding_time_window_sum</td>
<td>2769</td>
</tr>
<tr>
<td>B.319</td>
<td>smooth</td>
<td>2776</td>
</tr>
<tr>
<td>B.320</td>
<td>soft_all_equal_max_var</td>
<td>2779</td>
</tr>
<tr>
<td>B.321</td>
<td>soft_all_equal_min_ctr</td>
<td>2781</td>
</tr>
<tr>
<td>B.322</td>
<td>soft_all_equal_min_var</td>
<td>2783</td>
</tr>
<tr>
<td>B.323</td>
<td>soft_alldifferent_ctr</td>
<td>2785</td>
</tr>
<tr>
<td>B.324</td>
<td>soft_alldifferent_var</td>
<td>2787</td>
</tr>
<tr>
<td>B.325</td>
<td>soft_cumulative</td>
<td>2789</td>
</tr>
<tr>
<td>B.326</td>
<td>soft_same_interval_var</td>
<td>2791</td>
</tr>
<tr>
<td>B.327</td>
<td>soft_same_modulo_var</td>
<td>2794</td>
</tr>
<tr>
<td>B.328</td>
<td>soft_same_partition_var</td>
<td>2796</td>
</tr>
</tbody>
</table>
B.329 soft_same_var ................................. 2799
B.330 soft_used_by_interval_var .................. 2801
B.331 soft_used_by_modulo_var .................... 2804
B.332 soft_used_by_partition_var .................. 2806
B.333 soft_used_by_var ............................. 2809
B.334 some_equal .................................... 2812
B.335 sort ............................................ 2814
B.336 sort_permutation .............................. 2816
B.337 stable_compatibility ......................... 2818
B.338 stage_element ................................ 2820
B.339 stretch_circuit ................................ 2823
B.340 stretch_path .................................. 2826
B.341 stretch_path_partition ....................... 2830
B.342 strict_lex2 .................................... 2836
B.343 strictly_decreasing ........................... 2837
B.344 strictly_increasing ............................ 2840
B.345 strongly_connected ............................ 2842
B.346 subgraph_isomorphism ....................... 2843
B.347 sum ............................................. 2845
B.348 sum_ctr ........................................ 2847
B.349 sum_cubes_ctr ................................ 2849
B.350 sum_free ....................................... 2851
B.351 sum_of_increments ............................. 2852
B.352 sum_of_weights_of_distinct_values .......... 2854
B.353 sum_set ........................................ 2857
B.354 sum_squares_ctr ................................ 2859
B.355 symmetric ...................................... 2861
B.356 symmetric_alldifferent ....................... 2862
B.357 symmetric_alldifferent_except_0 ............. 2864
B.358 symmetric_cardinality ....................... 2866
B.359 symmetric_gcc ................................ 2868
B.360 temporal_path .................................. 2870
B.361 tour ............................................ 2873
B.362 track ........................................... 2875
B.363 tree ............................................. 2879
B.364 tree_range ...................................... 2881
B.365 tree_resource .................................. 2885
B.366 twin ............................................. 2889
B.367 two_layer_edge_crossing ...................... 2891
B.368 two_orth_are_in_contact ...................... 2894
B.369 two_orth_column ............................... 2897
B.370 two_orth_do_not_overlap ...................... 2899
B.371 two_orth_include ............................... 2902
B.372 used_by ........................................... 2904
B.373 used_by_interval .................................... 2906
B.374 used_by_modulo .................................... 2908
B.375 used_by_partition ................................. 2910
B.376 uses ................................................... 2913
B.377 valley ............................................... 2915
B.378 vec_eq_tuple ....................................... 2917
B.379 visible ............................................... 2919
B.380 weighted_partial_alldiff ......................... 2924
B.381 xor ................................................... 2928
B.382 Utilities ............................................. 2930
B.1  abs_value

◊  Meta-Data:

ctr_predefined(abs_value).
ctr_date(abs_value,['20100821']).
ctr_origin(abs_value,'Arithmetic.',[]).
ctr_usual_name(abs_value,abs).
ctr_synonyms(abs_value,[absolute_value]).
ctr_arguments(abs_value,['Y'-dvar,'X'-dvar]).
ctr_restrictions(abs_value,['Y'>=0]).
ctr_example(abs_value,abs_value(8,-8)).
ctr_eval(abs_value,[builtin(abs_value_b)]).
ctr_pure_functional_dependency(abs_value,[]).
ctr_functional_dependency(abs_value,1,[2]).

abs_value_b(Y,X) :-
    check_type(dvar,Y),
    check_type(dvar,X),
    X#=0#
    
    Y=X#/
    X<0#
    
    X+Y#=0.
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.2 all_differ_from_at_least_k_pos

◊ Meta-Data:

ctr_date(  
    all_differ_from_at_least_k_pos,  
    ['20030820','20040530','20060803']).

ctr_origin(  
    all_differ_from_at_least_k_pos,  
    Inspired by \cite{Frutos97}.,  
    []).

ctr_types(  
    all_differ_from_at_least_k_pos,  
    ['VECTOR'-collection(var-dvar)]).

ctr_arguments(  
    all_differ_from_at_least_k_pos,  
    ['K'-int,'VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(  
    all_differ_from_at_least_k_pos,  
    [required('VECTOR',var),  
     size('VECTOR')>=1,  
     size('VECTOR')>='K',  
     'K'>=0,  
     required('VECTORS',vec),  
     same_size('VECTORS',vec)]).

ctr_example(  
    all_differ_from_at_least_k_pos,  
    all_differ_from_at_least_k_pos(  
        2,  
        [[vec-[[var-2],[var-5],[var-2],[var-0]]],  
         [vec-[[var-3],[var-6],[var-2],[var-1]]],  
         [vec-[[var-3],[var-6],[var-1],[var-0]]]]).  

ctr_typical(  
    all_differ_from_at_least_k_pos,  
    ['K']>0,size('VECTOR')<size('VECTORS'),size('VECTORS'>1]).

ctr_exchangeable(  
    all_differ_from_at_least_k_pos,  
    [items('VECTORS',all),items_sync('VECTORS`vec',all)]).
ctr_graph(
    all_differ_from_at_least_k_pos,
    ['VECTORS']
)
2,
['CLIQUE' (\=\=) >> collection(vectors1, vectors2)],
[differ_from_at_least_k_pos(
    K,
    vectors1\^vec,
    vectors2\^vec)
],
['NARC' = size('VECTORS') * size('VECTORS') - size('VECTORS')],
['NO_LOOP', 'SYMMETRIC']).

ctr_eval(
    all_differ_from_at_least_k_pos,
    [reformulation(all_differ_from_at_least_k_pos_r)]).

ctr_contractible(
    all_differ_from_at_least_k_pos,
    [],
    VECTORS,
    any).

ctr_extensible(
    all_differ_from_at_least_k_pos,
    [],
    'VECTORS'\^vec,
    any).

all_differ_from_at_least_k_pos_r(K, VECTORS) :-
    integer(K),
    K >= 0,
    all_differ_from_at_least_k_pos_rr(K, VECTORS).

all_differ_from_at_least_k_pos_rr(_, []). :- !.

all_differ_from_at_least_k_pos_rr(K, [[] \31035-VECTOR1 | R]) :-
    length(VECTOR1, N),
    N >= K,
    all_differ_from_at_least_k_pos1(R, VECTOR1, K),
    all_differ_from_at_least_k_pos_rr(K, R).

all_differ_from_at_least_k_pos1([], []). :-}

all_differ_from_at_least_k_pos1([VECTOR2 | R], VECTOR1, K) :-
    eval(differ_from_at_least_k_pos(K, VECTOR1, VECTOR2)),
}
all_differ_from_at_least_k_pos1(R,VECTOR1,K).
**B.3  all_equal**

◊ **META-DATA:**

```prolog
ctr_date(all_equal,['20081005','20100418']).
ctr_origin(
    all_equal,
    Derived from %c,
    [soft_all_equal_min_ctr]).
ctr_synonyms(all_equal,[rel]).
ctr_arguments(all_equal,['VARIABLES'-collection(var-dvar)]).
ctr_restrictions(
    all_equal,
    [required('VARIABLES',var),size('VARIABLES'>0)]).
ctr_example(
    all_equal,
    all_equal([[var-5],[var-5],[var-5],[var-5]])).
ctr_typical(all_equal,[size('VARIABLES'>2)]).
ctr_exchangeable(
    all_equal,
    [items('VARIABLES',all),
     vals(['VARIABLES'\var,int,\=\=,all,dontcare])].
ctr_graph(
    all_equal,
    ['VARIABLES'],
    2,
    ['PATH'\>collection(variables1,variables2)],
    [variables1\var=variables2\var],
    ['NARC'=size('VARIABLES')-1],
    []).
ctr_eval(
    all_equal,
    [checker(all_equal_c),reformulation(all_equal_r)]).
ctr_contractible(all_equal,[]..'VARIABLES',any).
all_equal_c(VARIABLES) :-
```

```prolog```
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

collection(VARIABLES,[int]),
get_attr1(VARIABLES,VARS),
all_equal2(VARS).

all_equal_r(VARIABLES) :-
collection(VARIABLES,[dvar]),
get_attr1(VARIABLES,VARS),
all_equal1(VARS).

all_equal1([]).

all_equal1([_26890]) :- !.

all_equal1([V1,V2|R]) :-
V1#=V2,
all_equal1([V2|R]).

all_equal2([]).

all_equal2([_26890]) :- !.

all_equal2([V,V|R]) :-
all_equal2([V|R]).
B.4 all_incomparable

◊ **META-DATA:**

```plaintext
ctr_date(all_incomparable, ['20120202']).

ctr_origin(  
    all_incomparable,  
    Inspired by incomparable rectangles.,  
    []).

ctr_synonyms(all_incomparable, [all_incomparables]).

ctr_types(all_incomparable, ['VECTOR'-collection(var-dvar)]).

ctr_arguments(  
    all_incomparable,  
    ['VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(  
    all_incomparable,  
    [required('VECTOR', var),  
    size('VECTOR')>=1,  
    required('VECTORS', vec),  
    size('VECTORS')>=1,  
    same_size('VECTORS', vec)]).

ctr_example(  
    all_incomparable,  
    all_incomparable(  
        [[vec-[[var-16],[var-2]]],  
        [vec-[[var-4],[var-11]]],  
        [vec-[[var-5],[var-10]]]]).  

ctr_typical(  
    all_incomparable,  
    [size('VECTOR')>1,  
    size('VECTORS')>1,  
    size('VECTORS')>size('VECTOR')]).

ctr_exchangeable(all_incomparable, [items('VECTORS', all)]).

ctr_graph(  
    all_incomparable,  
    ['VECTORS'],  
    2,
```
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

['CLIQUE' (=\=)>>collection(vectors1,vectors2)],
[incomparable(vectors1\vec,vectors2\vec)],
['NARC'=size('VECTORS')\times size('VECTORS')\times size('VECTORS')],
['NO_LOOP','SYMMETRIC']).

ctr_eval(all_incomparable,[reformulation(all_incomparable_r)]).

ctr_contractible(all_incomparable,[],'VECTORS',any).

all_incomparable_r(VECTORS) :-
collection(VECTORS,[col([dvar])]),
same_size(VECTORS),
get_attr11(VECTORS,VECTS),
VECTS=[VEC|_27348],
length(VEC,N),
N\geq=1,
all_incomparable(VECTS,N).

all_incomparable([_27315],_27314) :- !.

all_incomparable(_27313,1) :- !,
fail.

all_incomparable(VECTS,_27314) :-
all_incomparable1(VECTS).

all_incomparable1([]).

all_incomparable1([_27311]).

all_incomparable1([V|R]) :-
all_incomparable2(R,V),
all_incomparable1(R).

all_incomparable2([],_27311).

all_incomparable2([V|R],U) :-
all_incomparable3(U,V),
all_incomparable2(R,U).

all_incomparable3([U1,U2],[V1,V2]) :- !,
U1#<V1#\U2#>V2#\U1#>V1#\U2#<V2.
all_incomparable3(U,V) :-
    length(U,N),
    length(V,N),
    N>2,
    length(PU,N),
    length(PV,N),
    domain(PU,1,N),
    domain(PV,1,N),
    get_minimum(U,MinU),
    get_maximum(U,MaxU),
    get_minimum(V,MinV),
    get_maximum(V,MaxV),
    length(SU,N),
    length(SV,N),
    domain(SU,MinU,MaxU),
    domain(SV,MinV,MaxV),
    sorting(U,PU,SU),
    sorting(V,PV,SV),
    all_incomparable4(SU,SV,Or1),
    call(Or1),
    all_incomparable4(SV,SU,Or2),
    call(Or2).

all_incomparable4([],[],0).

all_incomparable4([U|R],[V|S],U#>V#/T) :-
    all_incomparable4(R,S,T).
B.5  all_min_dist

◊  **Meta-Data:**

```prolog
ctr_date(all_min_dist,['20050508','20060803']).
ctr_origin(all_min_dist,\cite{Regin97},[]).
ctr_synonyms(all_min_dist,[minimum_distance,inter_distance]).
ctr_arguments(all_min_dist,
    ['MINDIST'-int,'VARIABLES'-collection(var-dvar)]).
ctr_restrictions(all_min_dist,
    ['MINDIST'>0,
     size('VARIABLES')<2#
     'MINDIST'<range('VARIABLES'\^var),
     required('VARIABLES',var)]).
ctr_example(all_min_dist,
    all_min_dist(2,[[var-5],[var-1],[var-9],[var-3]])).
ctr_typical(all_min_dist,['MINDIST'>1,size('VARIABLES'>1)]).
ctr_exchangeable(all_min_dist,
    [vals(['MINDIST'],int(>=(1)),>,dontcare,dontcare),
     items('VARIABLES',all),
     vals(['VARIABLES'\^var],int,\=,all,in),
     translate(['VARIABLES'\^var])]).
ctr_graph(all_min_dist,
    ['VARIABLES'],
    2,
    ['CLIQUE'(<)>>collection(variables1,variables2)],
    [abs(variables1\^var-variables2\^var)=='MINDIST'],
    ['NARC'=size('VARIABLES')\*(size('VARIABLES')-1)/2],
    ['ACYCLIC','NO_LOOP']).
ctr_eval(all_min_dist,[reformulation(all_min_dist_r)]).
ctr_contractible(all_min_dist,[],'VARIABLES',any).
```
all_min_dist_r(MINDIST,[]) :-
  !,
  integer(MINDIST),
  MINDIST>0.

all_min_dist_r(MINDIST,VARIABLES) :-
  integer(MINDIST),
  MINDIST>0,
  collection(VARIABLES,[dvar]),
  get_attr1(VARIABLES,VARS),
  length(VARS,N),
  ( N>1 ->
    list_dvar_range(VARS,RANGE),
    MINDIST#<RANGE,
    all_min_dist1(VARIABLES,MINDIST)
  ;
   true
  ).

all_min_dist1([],_35230).

all_min_dist1([[__35239-VAR1]|R],MINDIST) :-
  all_min_dist2(R,VAR1,MINDIST),
  all_min_dist1(R,MINDIST).

all_min_dist2([],_35230,_35231).

all_min_dist2([[__35240-VAR2]|R],VAR1,MINDIST) :-
  abs(VAR1-VAR2)#>=MINDIST,
  all_min_dist2(R,VAR1,MINDIST).
B.6 alldifferent

◊ **Meta-Data:**

```
ctr_date(
    alldifferent,
    [20000128,
     20030820,
     20040530,
     20060803,
     20081227,
     20090521]).

ctr_origin(alldifferent,'\cite{Lauriere78}',[]).

ctr_synonyms(
    alldifferent,
    [alldiff,
     alldistinct,
     distinct,
     bound_alldifferent,
     bound_alldiff,
     bound_distinct,
     rel]).

ctr_arguments(alldifferent,['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(alldifferent,[required('VARIABLES',var)]).

ctr_example(
    alldifferent,
    alldifferent([[var-5],[var-1],[var-9],[var-3]])).

ctr_typical(alldifferent,[size('VARIABLES')>1]).

ctr_exchangeable(
    alldifferent,
    [items('VARIABLES',all),
     vals(['VARIABLES`var],int,\=\,all,dontcare)].

ctr_graph(
    alldifferent,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1`var=variables2`var],
```


['MAX_NSNC'=<1], ['ONE_SUC']).

ctr_eval(
  alldifferent,
  [checker(alldifferent_c),
   builtin(alldifferent_b),
   reformulation(alldifferent_r1),
   reformulation(alldifferent_r2)]).

ctr_contractible(alldifferent,[],'VARIABLES',any).

ctr_sol(alldifferent,'A000142',[1,2,6,24,120,720,5040]).

alldifferent_c(VARIABLES) :-
  collection(VARIABLES,[int]),
  get_attr1(VARIABLES,VARS),
  sort(VARS,SVARS),
  length(VARS,N),
  length(SVARS,N).

alldifferent_b(VARIABLES) :-
  collection(VARIABLES,[dvar]),
  get_attr1(VARIABLES,VARS),
  alldifferent(VARS).

alldifferent_r1(VARIABLES) :-
  collection(VARIABLES,[dvar]),
  get_attr1(VARIABLES,VARS),
  get_minimum(VARS,MIN),
  get_maximum(VARS,MAX),
  length(VARS,N),
  length(L,N),
  domain(L,MIN,MAX),
  gen_collection(L,VAR,SORTED_VARIABLES),
  eval(sort(VARIABLES,SORTED_VARIABLES)),
  eval(strictly_increasing(SORTED_VARIABLES)).

alldifferent_r2(VARIABLES) :-
  collection(VARIABLES,[dvar]),
  get_attr1(VARIABLES,VARS),
  get_minimum(VARS,MIN),
  get_maximum(VARS,MAX),
  alldifferent_r20(MIN,MAX,VARS).

alldifferent_r20(L,MAX,_67682) :-
alldifferent_r20(L, MAX, VARS) :-
    alldifferent_r21(L, MAX, VARS),
    L1 is L+1,
    alldifferent_r20(L1, MAX, VARS).

alldifferent_r21(L, U, _67682) :-
    L>U,
    !.

alldifferent_r21(L, U, VARS) :-
    alldifferent_r22(VARS, L, U, T),
    S is U-L+1,
    call(T#=<S),
    U1 is U-1,
    alldifferent_r21(L, U1, VARS).

alldifferent_r22([], _67681, _67682, 0) :-
    !.

alldifferent_r22([Vi|R], L, U, Bilu+S) :-
    Vi in L..U#=<Bilu,
    alldifferent_r22(R, L, U, S).
B.7  alldifferent_between_sets

◊  **META-DATA:**

```
ctr_date(
   alldifferent_between_sets,
   ['20030820', '20051008', '20060803']).

ctr_origin(alldifferent_between_sets, 'ILOG', []).

ctr_synonyms(
   alldifferent_between_sets,
   [all_null_intersect,
    alldiff_between_sets,
    alldistinct_between_sets,
    alldiff_on_sets,
    alldistinct_on_sets,
    alldifferent_on_sets]).

ctr_arguments(
   alldifferent_between_sets,
   ['VARIABLES'-collection(var-svar)]).

ctr_restrictions(
   alldifferent_between_sets,
   [required('VARIABLES', var)]).

ctr_example(
   alldifferent_between_sets,
   alldifferent_between_sets(
     [[var-{3,5}], [var-{}], [var-{3}], [var-(3,5,7)]]).

ctr_typical(alldifferent_between_sets, [size('VARIABLES')>2]).

ctr_exchangeable(
   alldifferent_between_sets,
   [items('VARIABLES', all)]).

ctr_graph(
   alldifferent_between_sets,
   ['VARIABLES'],
   2,
   ['CLIQUE'>>collection(variables1, variables2)],
   [eq_set(variables1^var, variables2^var)],
   ['MAX_NSNC'=1],
   ['ONE_SUCCE'].
```
ctr_contractible(alldifferent_between_sets,[],'VARIABLES',any).
B.8 alldifferent_consecutive_values

◊ META-DATA:

ctr_date(alldifferent_consecutive_values, ['20080618']).

ctr_origin(alldifferent_consecutive_values, Derived from %c., [alldifferent]).

ctr_arguments(alldifferent_consecutive_values, ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(alldifferent_consecutive_values, [required('VARIABLES', var), alldifferent('VARIABLES')]).

ctr_example(alldifferent_consecutive_values, alldifferent_consecutive_values([[var-5],[var-4],[var-3],[var-6]])).

ctr_typical(alldifferent_consecutive_values, [size('VARIABLES')>2]).

ctr_exchangeable(alldifferent_consecutive_values, [items('VARIABLES', all), vals(['VARIABLES'\var], int, =\=, all, in), translate(['VARIABLES'\var])]).

ctr_graph(alldifferent_consecutive_values, ['VARIABLES'], 1, ['SELF'>>collection(variables)], ['TRUE'], ['RANGE'('VARIABLES', var)=size('VARIABLES')-1], []).

ctr_eval(alldifferent_consecutive_values, [checker(alldifferent_consecutive_values_c),]
reformulation(alldifferent_consecutive_values_r))).

alldifferent_consecutive_values_c([]) :-
    !.

alldifferent_consecutive_values_c(VARIABLES) :-
   collection(VARIABLES,[int]),
   get_attr1(VARIABLES,VARS),
   sort(VARS,SVARS),
   length(VARS,N),
   length(SVARS,N),
   min_member(MIN,VARS),
   max_member(MAX,VARS),
   N is MAX-MIN+1.

alldifferent_consecutive_values_r([]) :-
    !.

alldifferent_consecutive_values_r(VARIABLES) :-
   collection(VARIABLES,[dvar]),
   get_attr1(VARIABLES,VARS),
   all_different(VARS),
   minimum(MIN,VARS),
   maximum(MAX,VARS),
   length(VARIABLES,N),
   N#=MAX-MIN+1.
B.9  alldifferent_cst

◊ **META-DATA:**

```plaintext
ctr_date(alldifferent_cst,['20051104','20060803']).

ctr_origin(alldifferent_cst,'\index{CHIP|indexuse} CHIP',[]).

ctr_synonyms(alldifferent_cst,[alldiff_cst,alldistinct_cst]).

ctr_arguments(
    alldifferent_cst,
    ['VARIABLES'-collection(var-dvar,cst-int)]).

ctr_restrictions(
    alldifferent_cst,
    [required('VARIABLES',[var,cst])]).

ctr_example(
    alldifferent_cst,
    alldifferent_cst(
        [[var-5,cst-0],
        [var-1,cst-1],
        [var-9,cst-0],
        [var-3,cst-4]])).

ctr_typical(
    alldifferent_cst,
    [size('VARIABLES')>2,
     range('VARIABLES'~var)>1,
     range('VARIABLES'~cst)>1]).

ctr_exchangeable(
    alldifferent_cst,
    [items('VARIABLES',all),
     attrs('VARIABLES',[var,cst]),
     translate(['VARIABLES'~var]),
     translate(['VARIABLES'~cst])].

ctr_graph(
    alldifferent_cst,
    ['VARIABLES'],
    2,
    ['CLIQUE']>>collection(variables1,variables2)),
    [variables1~var+variables1~cst=variables2~var+variables2~cst],
```
[’MAX_NSCC’=<1],
[’ONE_SUCC’]).

ctr_eval(alldifferent_cst,[reformulation(alldifferent_cst_r)]).

ctr_contractible(alldifferent_cst,[],’VARIABLES’,any).

alldifferent_cst_r(VARIABLES) :-
    collection(VARIABLES,[dvar,int]),
    get_attr1(VARIABLES,VARS),
    get_attr2(VARIABLES,CSTS),
    gen_varcst(VARS,CSTS,VARCSTS),
    all_different(VARCSTS).
B.10  alldifferent_except_0

◊ META-DATA:

ctr_date(  
alldifferent_except_0,  
[‘20000128’,‘20030820’,‘20040530’,‘20060803’]).

ctr_origin(  
alldifferent_except_0,  
Derived from %c.,  
[alldifferent]).

ctr_synonyms(  
alldifferent_except_0,  
[alldiff_except_0,alldistinct_except_0]).

ctr_arguments(  
alldifferent_except_0,  
[‘VARIABLES’-collection(var-dvar)]).

ctr_restrictions(  
alldifferent_except_0,  
[required(‘VARIABLES’,var)]).

ctr_example(  
alldifferent_except_0,  
alldifferent_except_0(  
  [[var-5],[var-0],[var-1],[var-9],[var-0],[var-3]])).

ctr_typical(  
alldifferent_except_0,  
[size(‘VARIABLES’)>2,  
atleast(2,‘VARIABLES’,0),  
range(‘VARIABLES’^var>1])).

ctr_exchangeable(  
alldifferent_except_0,  
[items(‘VARIABLES’,all),  
  vals([‘VARIABLES’^var],int(\=(0)),\=,all,dontcare)])..

ctr_graph(  
alldifferent_except_0,  
[‘VARIABLES’],  
2,  
[‘CLIQUE’>>collection(variables1,variables2)],
[variables1^var=\=0,variables1^var=variables2^var],
['MAX_NSNC'=<1],
[]).

ctr_eval(
    alldifferent_except_0,
    [checker(alldifferent_except_0_c),
     reformulation(alldifferent_except_0_r)]).

ctr_contractible(alldifferent_except_0,[],'VARIABLES',any).

alldifferent_except_0_c(VARIABLES) :-
    collection(VARIABLES,[int]),
    get_attr1(VARIABLES,VARS),
    filter_zeros(VARS,L),
    sort(L,SL),
    length(L,N),
    length(SL,N).

filter_zeros([],[]) :- !.

filter_zeros([0|R],S) :- !,
    filter_zeros(R,S).

filter_zeros([X|R],[X|S]) :-
    filter_zeros(R,S).

alldifferent_except_0_r(VARIABLES) :-
    collection(VARIABLES,[dvar]),
    get_attr1(VARIABLES,VARS),
    alldifferent_except_01(VARS).

alldifferent_except_01([],).

alldifferent_except_01([_34431]) :- !.

alldifferent_except_01([V1|R]) :-
    alldifferent_except_01(R,V1),
    alldifferent_except_01(R).

alldifferent_except_01([],_34428).

alldifferent_except_01([V2|R],V1) :-
V1\#=0\#/V2\#=0\#/V1\#\=V2,
alldifferent_except_01(R,V1).
B.11 alldifferent_interval

◊ Meta-Data:

ctr_date(alldifferent_interval,[’20030820’,’20060803’]).

ctr_origin(
    alldifferent_interval,
    Derived from %c.,
    [alldifferent]).

ctr_synonyms(
    alldifferent_interval,
    [alldiff_interval,alldistinct_interval]).

ctr_arguments(
    alldifferent_interval,
    [’VARIABLES’-collection(var-dvar),’SIZE_INTERVAL’-int]).

ctr_restrictions(
    alldifferent_interval,
    [required(’VARIABLES’,var),’SIZE_INTERVAL’>0]).

ctr_example(
    alldifferent_interval,
    alldifferent_interval(([var-2],[var-4],[var-10]),3)).

ctr_typical(
    alldifferent_interval,
    [size(’VARIABLES’)>2,
        ’SIZE_INTERVAL’>1,
        ’SIZE_INTERVAL’<range(’VARIABLES’^var)]).

ctr_exchangeable(
    alldifferent_interval,
    [items(’VARIABLES’,all),
        vals(
            [’VARIABLES’^var],
            intervals(’SIZE_INTERVAL’),
            =,
            all,
            dontcare),
        vals(
            [’VARIABLES’^var],
            intervals(’SIZE_INTERVAL’),
            =\=,
all,
in)}).

```prolog
ctr_graph(
  alldifferent_interval,
  ['VARIABLES'],
  2,
  ['CLIQUE'>>collection(variables1,variables2)],
  [variables1\var/'SIZE_INTERVAL'= variables2\var/'SIZE_INTERVAL'],
  ['MAX_NSCC'=<1],
  ['ONE_SUCC']).

ctr_eval(
  alldifferent_interval,
  [reformulation(alldifferent_interval_r)]).

ctr_contractible(alldifferent_interval, [], 'VARIABLES', any).

alldifferent_interval_r(VARIABLES,SIZE_INTERVAL) :-
  collection(VARIABLES,[dvar]),
  integer(SIZE_INTERVAL),
  SIZE_INTERVAL>0,
  get_attr1(VARIABLES,VARS),
  gen_quotient(VARS,SIZE_INTERVAL,QUOTVARS),
  all_different(QUOTVARS).
```
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.12  alldifferent_modulo

◊ **META-DATA:**

```
ctr_date(alldifferent_modulo,['20030820','20060803']).
```

```
ctr_origin(
    alldifferent_modulo,
    Derived from %c.,
    [alldifferent]).
```

```
ctr_synonyms(
    alldifferent_modulo,
    [alldiff_modulo,alldistinct_modulo]).
```

```
ctr_arguments(
    alldifferent_modulo,
    ['VARIABLES'-collection(var-dvar),'M'-int]).
```

```
ctr_restrictions(
    alldifferent_modulo,
    [required('VARIABLES',var),'M'>0,'M'>=size('VARIABLES')]).
```

```
ctr_example(
    alldifferent_modulo,
    alldifferent_modulo([[var-25],[var-1],[var-14],[var-3]],5)).
```

```
ctr_typical(alldifferent_modulo,[size('VARIABLES')>2,'M'>1]).
```

```
ctr_exchangeable(
    alldifferent_modulo,
    [items('VARIABLES',all),
     vals(['VARIABLES'-'var],mod('M'),=,all,dontcare),
     vals(['VARIABLES'-'var],mod('M'),=\=,all,in)]).
```

```
ctr_graph(
    alldifferent_modulo,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1-'var mod 'M'=variables2-'var mod 'M'],
    ['MAX_NSCC'=<1],
    ['ONE_SUCC']).
```

```
ctr_eval(
    alldifferent_modulo,
    Derived from %c.,
    [alldifferent]).
```

```
ctr_synonyms(
    alldifferent_modulo,
    [alldiff_modulo,alldistinct_modulo]).
```

```
ctr_arguments(
    alldifferent_modulo,
    ['VARIABLES'-collection(var-dvar),'M'-int]).
```

```
ctr_restrictions(
    alldifferent_modulo,
    [required('VARIABLES',var),'M'>0,'M'>=size('VARIABLES')]).
```

```
ctr_example(
    alldifferent_modulo,
    alldifferent_modulo([[var-25],[var-1],[var-14],[var-3]],5)).
```

```
ctr_typical(alldifferent_modulo,[size('VARIABLES')>2,'M'>1]).
```

```
ctr_exchangeable(
    alldifferent_modulo,
    [items('VARIABLES',all),
     vals(['VARIABLES'-'var],mod('M'),=,all,dontcare),
     vals(['VARIABLES'-'var],mod('M'),=\=,all,in)]).
```

```
ctr_graph(
    alldifferent_modulo,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1-'var mod 'M'=variables2-'var mod 'M'],
    ['MAX_NSCC'=<1],
    ['ONE_SUCC']).
```

```
ctr_eval(
    alldifferent_modulo,
    Derived from %c.,
    [alldifferent]).
```
[reformulation(alldifferent_modulo_r)].

ctr_contractible(alldifferent_modulo,[],'VARIABLES',any).

alldifferent_modulo_r(VARIABLES,M) :-
  collection(VARIABLES,[dvar]),
  integer(M),
  M>0,
  length(VARIABLES,N),
  M=N,
  get_attr1(VARIABLES,VARS),
  gen_remainder(VARS,M,REMVARS),
  all_different(REMVARS).
B.13  alldifferent_on_intersection

◊ Meta-Data:

ctr_date(alldifferent_on_intersection,['20040530','20060803']).

ctr_origin(
    alldifferent_on_intersection,
    Derived from %c and %c.,
    [common,alldifferent]).

ctr_synonyms(
    alldifferent_on_intersection,
    [alldiff_on_intersection,alldistinct_on_intersection]).

ctr_arguments(
    alldifferent_on_intersection,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    alldifferent_on_intersection,
    [required('VARIABLES1',var),required('VARIABLES2',var)]).

ctr_example(
    alldifferent_on_intersection,
    alldifferent_on_intersection(
        [[var-5],[var-9],[var-1],[var-5]],
        [[var-2],[var-1],[var-6],[var-9],[var-6],[var-2]])).

ctr_typical(
    alldifferent_on_intersection,
    [size('VARIABLES1')>1,size('VARIABLES2')>1]).

ctr_exchangeable(
    alldifferent_on_intersection,
    [args([[VARIABLES1','VARIABLES2']]),
     items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(
         ['VARIABLES1'\var,'VARIABLES2'\var],
         int,
         !=, all, don'tcare)]).
ctr_graph(
alldifferent_on_intersection,
[‘VARIABLES1’,’VARIABLES2’],
2,
[’PRODUCT’>>collection(variables1,variables2)],
[variables1`var=variables2`var],
[’MAX_NCC’=<2],
[’ACYCLIC’,’BIPARTITE’,’NO_LOOP’]).

ctr_eval(
alldifferent_on_intersection,
[reformulation(alldifferent_on_intersection_r)]).

ctr_contractible(
alldifferent_on_intersection,
[],
VARIABLES1,
any).

ctr_contractible(
alldifferent_on_intersection,
[],
VARIABLES2,
any).

alldifferent_on_intersection_r(VARIABLES1,VARIABLES2) :-
collection(VARIABLES1,[dvar]),
collection(VARIABLES2,[dvar]),
get_atrr1(VARIABLES1,VARS1),
get_atrr1(VARIABLES2,VARS2),
alldifferent_on_intersection(VARS1,1,VARS1,VARS2).

alldifferent_on_intersection([],_37660,_37661,_37662).

alldifferent_on_intersection([VAR1|R1],I,VARS1,VARS2) :-
alldifferent_on_intersection( 
 VARS2, 
 1, 
  VAR1, 
  I, 
  VARS1, 
  VARS2),
I1 is I+1,
alldifferent_on_intersection(R1,I1,VARS1,VARS2).

alldifferent_on_intersection(}
alldifferent_on_intersection([VAR2|R2], J, VAR1, I, VARS1, VARS2) :-
alldifferent_on_intersection1(VARS1, I, VAR1, I, VAR2, J),
alldifferent_on_intersection1(VARS2, I, VAR2, J, VAR1, I),
J1 is J+1,
alldifferent_on_intersection(R2, J1, VAR1, I, VARS1, VARS2).

alldifferent_on_intersection1([VAR|R], K, VAR1, I, VAR2, J) :-
K=\=I,
!,
VAR1#=VAR2#>VAR#\=VAR1,
K1 is K+1,
alldifferent_on_intersection1(R, K1, VAR1, I, VAR2, J).

alldifferent_on_intersection1([\_37668|R], K, VAR1, I, VAR2, J) :-
K=:=I,
K1 is K+1,
alldifferent_on_intersection1(R, K1, VAR1, I, VAR2, J).
B.14 alldifferent_partition

◊ META-DATA:

\[\text{ctr\_date(alldifferent\_partition,} [\text{‘20030820’, ‘20060803’}] \).\]

\[\text{ctr\_origin(}
\begin{align*}
&\text{alldifferent\_partition,} \\
&\text{Derived from %c.,} \\
&[\text{alldifferent}]\).
\end{align*}
\]

\[\text{ctr\_synonyms(}
\begin{align*}
&\text{alldifferent\_partition,} \\
&[\text{alldiff\_partition, alldistinct\_partition}]\).
\end{align*}
\]

\[\text{ctr\_types(}
\begin{align*}
&\text{alldifferent\_partition,} \\
&[\text{‘VALUES’-collection(val-int)}]\).
\end{align*}
\]

\[\text{ctr\_arguments(}
\begin{align*}
&\text{alldifferent\_partition,} \\
&[\text{‘VARIABLES’-collection(var-dvar),} \\
&‘\text{PARTITIONS’-collection(p-’VALUES’)}]\).
\end{align*}
\]

\[\text{ctr\_restrictions(}
\begin{align*}
&\text{alldifferent\_partition,} \\
&[\text{size(’VALUES’)>=1,} \\
&\text{required(’VALUES’, val),} \\
&\text{distinct(’VALUES’, val),} \\
&\text{size(’VARIABLES’)=<size(’PARTITIONS’),} \\
&\text{required(’VARIABLES’, var),} \\
&\text{size(’PARTITIONS’)>=2,} \\
&\text{required(’PARTITIONS’, p)}]\).
\end{align*}
\]

\[\text{ctr\_example(}
\begin{align*}
&\text{alldifferent\_partition,} \\
&\text{alldifferent\_partition(}
\begin{align*}
&[[\text{var-6}, \text{var-3}, \text{var-4}], \\
&[[\text{p-}[[\text{val-1}, \text{val-3}]], \\
&[\text{p-}[[\text{val-4}]], \\
&[\text{p-}[[\text{val-2}, \text{val-6}]]]]])].
\end{align*}
\end{align*}
\)

\[\text{ctr\_typical(alldifferent\_partition,} [\text{size(’VARIABLES’)}>2])\).
\]

\[\text{ctr\_exchangeable(}
\begin{align*}
&\text{alldifferent\_partition,} \\
&\text{alldifferent\_partition,} \\
&\text{alldifferent\_partition,}
\end{align*}
\]

\[\text{alldifferent\_partition,} \\
\]
2008

APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

[items('VARIABLES',all),
items('PARTITIONS',all),
items('PARTITIONS' \^ p, all),
vals(
    ['VARIABLES'\^ var],
    part('PARTITIONS'),
    =,
    all,
    dontcare),
vals(['VARIABLES'\^ var],part('PARTITIONS'),=\=,all,in])).

ctr_graph(
alldifferent_partition,
['VARIABLES'],
2,
['CLIQUE']>>collection(variables1,variables2]),
[in_same_partition(
    variables1\^ var,
    variables2\^ var,
    PARTITIONS)],
['MAX_NSCC'=<1],
['ONE_SUCCE']).

ctr_eval(
alldifferent_partition,
[reformulation(alldifferent_partition_r)])).

ctr_contractible(alldifferent_partition,[],'VARIABLES',any).

alldifferent_partition_r(VARIABLES,PARTITIONS) :-
collection(VARIABLES,[dvar]),
collection(PARTITIONS,[col_len_gteq(1,[int])])),
get_attr1(VARIABLES,VARS),
get_col_attr1(PARTITIONS,1,PVALS),
flattern(PVALS,VALS),
all_different(VALS),
length(VARIABLES,N),
length(PARTITIONS,M),
N=<M,
M>1,
length(PVALS,LPVALS),
get_partition_var(VARS,PVALS,PVARS,LPVALS,0),
allDifferent(PVARS).
B.15  alldifferent_same_value

◊ **META-DATA:**

```prolog
ctr_date(
    alldifferent_same_value,
    ['20000128','20030820','20060803']).

ctr_origin(
    alldifferent_same_value,
    Derived from %c.,
    [alldifferent]).

ctr_synonyms(
    alldifferent_same_value,
    [alldiff_same_value,alldistinct_same_value]).

ctr_arguments(
    alldifferent_same_value,
    ['NSAME'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    alldifferent_same_value,
    ['NSAME'>=0,
     'NSAME'=<size('VARIABLES1'),
     size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var)]).

ctr_example(
    alldifferent_same_value,
    alldifferent_same_value(2,
        [[var-7],[var-3],[var-1],[var-5]],
        [[var-1],[var-3],[var-1],[var-7]])).

ctr_typical(
    alldifferent_same_value,
    ['NSAME'<size('VARIABLES1'),size('VARIABLES1')>2]).

ctr_exchangeable(
    alldifferent_same_value,
    [items_sync('VARIABLES1','VARIABLES2',all),
     vals(
2010

APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

[['VARIABLES1'\'var,'VARIABLES2'\'var],
     int,
     =\!=,
     all,
     dontcare]]).

ctr_graph(
    alldifferent_same_value,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT' ('CLIQUE','LOOP','=>)
        collection(variables1,variables2),
        variables1\'var=variables2\'var],
    ['MAX_NSNC'=<1,'NARC_NO_LOOP'='NSAME'],
    [])).

ctr_eval(
    alldifferent_same_value,
    [reformulation(alldifferent_same_value_r)])

ctr_functional_dependency(alldifferent_same_value,1,[2,3])

alldifferent_same_value_r(NSAME,VARIABLES1,VARIABLES2) :-
    check_type(dvar,NSAME),
    collection(VARIABLES1,[dvar]),
    collection(VARIABLES2,[dvar]),
    length(VARIABLES1,N1),
    length(VARIABLES2,N2),
    NSAME#>=0,
    NSAME#=<N1,
    N1=N2,
    get_attr1(VARIABLES1,VARS1),
    get_attr1(VARIABLES2,VARS2),
    all_different(VARS1),
    alldifferent_same_value1(VARS1,VARS2,SUMBOOLS),
    call(NSAME#=SUMBOOLS).

alldifferent_same_value1([],[],0).

alldifferent_same_value1([V1|R1],[V2|R2],B+R) :-
    V1#=V2#<=B,
    alldifferent_same_value1(R1,R2,R).
B.16 allperm

◊ **META-DATA:**

```prolog
ctr_date(allperm,['20031008','20070916']).

ctr_origin(
  allperm,
  \cite{FlenerFrischHnichKiziltanMiguelPearsonWalsh02},
  []).

ctr_synonyms(allperm,[all_perm,all_permutations]).

ctr_types(allperm,['VECTOR'-collection(var-dvar)]).

ctr_arguments(allperm,['MATRIX'-collection(vec-'VECTOR')]).

ctr_restrictions(
  allperm,
  [size('VECTOR')>=1,
    required('VECTOR',var),
    required('MATRIX',vec),
    same_size('MATRIX',vec)]).

ctr_example(
  allperm,
  allperm( [[vec-[[var-1],[var-2],[var-3]]],
            [vec-[[var-3],[var-1],[var-2]]]]).

ctr_typical(allperm,[size('VECTOR')>1,size('MATRIX')>1]).

ctr_exchangeable(allperm,[translate([MATRIX`vec`var])]).

ctr_graph(
  allperm,
  ['MATRIX'],
  2,
  ['CLIQUE'(<)>>collection(matrix1,matrix2)],
  [matrix1`key=1,
    matrix2`key>1,
    lex_lesseq_allperm(matrix1`vec,matrix2`vec)],
  ['NARC'=size('MATRIX')-1],
  ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(allperm,[reformulation(allperm_r)]).
```
ctr_contractible(allperm,[],"MATRIX"^vec,suffix).

allperm_r(MATRIX) :-
    same_size(MATRIX),
    MATRIX=[[vec-F]|R],
    allperm_sorted(R,S),
    allperm_order(S,F).

allperm_sorted([],[]).

allperm_sorted([[vec-X]|R],[S|T]) :-
    get_attr1(X,L),
    get_minimum(L,MIN),
    get_maximum(L,MAX),
    length(X,LX),
    length(Y,LX),
    domain(Y,MIN,MAX),
    gen_collection(Y,var,S),
    eval(sort(X,S)),
    allperm_sorted(R,T).

allperm_order([],_31571).

allperm_order([X|R],F) :-
    eval(lex_lesseq(F,X)),
    allperm_order(R,F).
B.17 among

◊ Meta-Data:

```prolog
ctr_date(among, [‘20000128’, ‘20030820’, ‘20040807’, ‘20060804’]).

ctr_origin(among, ‘\cite{BeldiceanuCentejean94}’[], []).

ctr_synonyms(among, [between, count]).

ctr_arguments(among,
    [‘NVAR’-dvar,
     ‘VARIABLES’-collection(var-dvar),
     ‘VALUES’-collection(val-int)]).

ctr_restrictions(among,
    [‘NVAR’>=0,
     ‘NVAR’=size(‘VARIABLES’),
     required(‘VARIABLES’-var),
     required(‘VALUES’-val),
     distinct(‘VALUES’-val)]).

ctr_example(among,
    among,
    among(3,
        [[var-4],[var-5],[var-5],[var-4],[var-1]],
        [[val-1],[val-5],[val-8]]).

ctr_typical(among,
    [‘NVAR’>0,
     ‘NVAR’<size(‘VARIABLES’),
     size(‘VARIABLES’)>1,
     size(‘VALUES’)>1,
     size(‘VARIABLES’)>size(‘VALUES’)].

ctr_exchangeable(among,
    [items(‘VARIABLES’, all),
     items(‘VALUES’, all),
     vals(‘VARIABLES’-var],
     comp(‘VALUES’-val),
```
CTR Graph

\[
\begin{align*}
\text{ctr_graph}( & \text{among,} \\
& ['\text{VARIABLES}',] \\
& 1, \\
& ['\text{SELF}'>\text{collection}([\text{variables}]),] \\
& [\text{variables}^\text{\texttt{var}} \in ['\text{VALUES}'],] \\
& ['\text{NARC}='\text{\texttt{NVAR}}'],] \\
& []). \\
\end{align*}
\]

CTR Evaluation

\[
\begin{align*}
\text{ctr_eval}( & \text{among,} [\text{reformulation}([\text{among_r}]),\text{automaton}([\text{among_a}])]). \\
\end{align*}
\]

CTR Pure Functional Dependency

\[
\begin{align*}
\text{ctr_pure_functional_dependency}( & \text{among,} []). \\
\end{align*}
\]

CTR Functional Dependency

\[
\begin{align*}
\text{ctr_functional_dependency}( & \text{among,} 1, [2,3]). \\
\end{align*}
\]

CTR Contractible

\[
\begin{align*}
\text{ctr_contractible}( & \text{among,} ['\text{NVAR}=0'], ['\text{VARIABLES}', \text{any}]). \\
\end{align*}
\]

CTR Contractible

\[
\begin{align*}
\text{ctr_contractible}( & \text{among,} ['\text{NVAR}=\text{size}('\text{VARIABLES}'),] \\
& \text{VARIABLES},] \\
& \text{any}). \\
\end{align*}
\]

CTR Aggregate

\[
\begin{align*}
\text{ctr_aggregate}( & \text{among,} [], [+], \text{union}, \text{union}). \\
\end{align*}
\]

among_r(NVAR, VARIABLES, VALUES) :-
\[
\begin{align*}
& \text{check_type}([\text{dvar}], \text{NVAR}),] \\
& \text{collection}([\text{VARIABLES},[\text{dvar}]),] \\
& \text{collection}([\text{VALUES},[\text{int}]),] \\
& \text{get_attr1}([\text{VARIABLES}, \text{VARS}),] \\
& \text{get_attr1}([\text{VALUES}, \text{VALS}),] \\
& \text{length}([\text{VARIABLES}, \text{N}),] \\
& \text{NVAR}>0,] \\
& \text{NVAR}=<\text{N},] \\
& \text{all_different}([\text{VALS}),] \\
& \text{among1}([\text{VARS}, \text{VALS}, \text{SUM_BVARS}),] \\
& \text{call}(\text{NVAR}=<\text{SUM_BVARS}). \\
\end{align*}
\]

among1([], _46147, 0).

among1([V|R], VALS, B+S) :-
\[
\begin{align*}
& \text{build_or_var_in_values}([\text{VALS}, \text{V}, \text{OR}]),] \\
\end{align*}
\]
call(OR#<=B),
among1(R,VALS,S).

among_a(FLAG,NVAR,VARIABLES,VALUES) :-
  check_type(dvar,NVAR),
collection(VARIABLES,[dvar]),
collection(VALUES,[int]),
get_attr1(VALUES,LIST_VALUES),
length(VARIABLES,N),
NVAR#>=0,
NVAR#=<N,
all_different(LIST_VALUES),
list_to_fdset(LIST_VALUES,SET_OF_VALUES),
among_signature(VARIABLES,signature,SET_OF_VALUES),
automaton(
  signature,
  _48426,
  signature,
  [source(s),sink(s)],
  [arc(s,0,s),arc(s,1,s,[C+1])],
  [C],
  [0],
  [COUNT]),
COUNT#=NVAR#<=FLAG.

among_signature([],[],_46148).

among_signature([[var-VAR]|VARs],[S|Ss],SET_OF_VALUES) :-
  VAR in_SET SET_OF_VALUES#<=S,
among_signature(VARs,Ss,SET_OF_VALUES).
B.18  among_diff_0

◊ Meta-Data:

ctr_date(among_diff_0, ['20040807','20060804']).

ctr_origin(
    among_diff_0,
    Used in the automaton of %c., [nvalue]).

ctr_arguments(
    among_diff_0,
    ['NVAR'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    among_diff_0,
    ['NVAR'>=0,
     'NVAR'=<size('VARIABLES'),
     required('VARIABLES',var)]).

ctr_example(
    among_diff_0,
    among_diff_0(3,[[var-0],[var-5],[var-5],[var-0],[var-1]]).

ctr_typical(
    among_diff_0,
    ['NVAR'>0,
     'NVAR'<size('VARIABLES'),
     size('VARIABLES')>1,
     atleast(1,'VARIABLES',0)])

ctr_exchangeable(
    among_diff_0,
    [items('VARIABLES',all),
     vals(
       ['VARIABLES'\var],
       int(\=(0)),
       \=,
       dontcare,
       dontcare)]).

ctr_graph(
    among_diff_0,
    ['VARIABLES'],
    1,
['SELF'>>collection(variables)],
[variables^var\=0],
['NARC'='NVAR'],
[]).

ctr_eval(
   among_diff_0,
   [reformulation(among_diff_0_r),automaton(among_diff_0_a)]).

ctr_pure_functional_dependency(among_diff_0,[]).

ctr_functional_dependency(among_diff_0,1,[2]).

ctr_contractible(among_diff_0,['NVAR'=0],'VARIABLES',any).

ctr_contractible(
   among_diff_0,
   ['NVAR'=size('VARIABLES')],
   VARIABLES,
   any).

ctr_aggregate(among_diff_0,[],[+,union]).

among_diff_0_r(NVAR,VARIABLES) :-
   check_type(dvar,NVAR),
   collection(VARIABLES,[dvar]),
   get_attr1(VARIABLES,VARS),
   length(VARIABLES,N),
   NVAR#\=0,
   NVAR#\=<N,
   among_diff_01(VARS,SUM_BVARS),
   call(NVAR#=SUM_BVARS).

among_diff_01([],0).

among_diff_01([V|R],B+S) :-
   V\=0#\=<B,
   among_diff_01(R,S).

among_diff_0_a(FLAG,NVAR,VARIABLES) :-
   check_type(dvar,NVAR),
   collection(VARIABLES,[dvar]),
   length(VARIABLES,N),
   NVAR#\=0,
   NVAR#\=<N,
   among_diff_0_signature(VARIABLES,SIGNALATURE),
   call(FLAG,NVAR,VARIABLES).
automaton(
  SIGNATURE,
  _31202,
  SIGNATURE,
  [source(s),sink(s)],
  [arc(s,0,s),arc(s,1,s,[C+1])],
  [C],
  [0],
  [COUNT]),
COUNT#=NVAR#<=>FLAG.

among_diff_0_signature([],[]).

among_diff_0_signature([var-VAR]|VARs,[S|Ss]) :-
  VAR#\=0#<=>S,
  among_diff_0_signature(VARs,Ss).
B.19 among_interval

◊ **META-DATA:**

```prolog
ctr_date(among_interval, [’20030820’, ’20040530’, ’20060804’]).
ctr_origin(among_interval, ’Derived from %c.’, [among]).

ctr_arguments(
    among_interval,
    [’NVAR’—dvar,
      ’VARIABLES’—collection(var—dvar),
      ’LOW’—int,
      ’UP’—int]).

ctr_restrictions(
    among_interval,
    [’NVAR’>=0,
     ’NVAR’=<size(’VARIABLES’),
     required(’VARIABLES’, var),
     ’LOW’=<’UP’]).

ctr_example(
    among_interval,
    among_interval(3,
      [var-4], [var-5], [var-8], [var-4], [var-1],
      3,
      5)).

ctr_typical(
    among_interval,
    [’NVAR’>0,
     ’NVAR’<size(’VARIABLES’),
     size(’VARIABLES’) >1,
     ’LOW’<’UP’,
     ’LOW’=<maxval(’VARIABLES’—var),
     ’UP’>=minval(’VARIABLES’—var)]).

ctr_exchangeable(
    among_interval,
    [items(’VARIABLES’, all),
     vals(’VARIABLES’—var],
     comp(’LOW’ in ’UP’),
     =].
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

dontcare,
dontcare))).

ctr_graph(
  among_interval,
  ['VARIABLES'],
  1,
  ['SELF'>>collection(variables)],
  ['LOW'=variables\^var,variables\^var=<'UP'],
  ['NARC'='NVAR'],
  []).

ctr_eval(
  among_interval,
  [reformulation(among_interval_r),
   automaton(among_interval_a)]).

ctr_pure_functional_dependency(among_interval,[]).

ctr_functional_dependency(among_interval,1,[2,3,4]).

ctr_contractible(among_interval,['NVAR'=0,'VARIABLES',any]).

ctr_contractible(
  among_interval,
  ['NVAR'=size('VARIABLES')],
  VARIABLES,
  any).

ctr_aggregate(among_interval,[],[+,union,id,id]).

among_interval_r(NVAR,VARIABLES,LOW,UP) :-
  check_type(dvar,NVAR),
  collection(VARIABLES,[dvar]),
  get_attr1(VARIABLES,VARS),
  integer(LOW),
  integer(UP),
  length(VARIABLES,N),
  NVAR#>=0,
  NVAR#=<N,
  LOW=<UP,
  among_interval1(VARS,SUM_BVARS,LOW,UP),
  call(NVAR#=SUM_BVARS).

among_interval1([],0,_31526,_31527).
among_interval1([V|R],B+S,LOW,UP) :-
    V#>=LOW#/V#=<UP#<=>B,
    among_interval1(R,S,LOW,UP).

among_interval_a(FLAG,NVAR,VARIABLES,LOW,UP) :-
    check_type(dvar,NVAR),
    collection(VARIABLES,[dvar]),
    integer(LOW),
    integer(UP),
    length(VARIABLES,N),
    NVAR#>=0,
    NVAR#=<N,
    LOW=<UP,
    among_interval_signature(VARIABLES,SIGNATURE,LOW,UP),
    automaton(
        SIGNATURE,
        _33665,
        SIGNATURE,
        [source(s),sink(s)],
        [arc(s,0,s),arc(s,1,s,[C+1])],
        [C],
        [0],
        [COUNT]),
    COUNT#=NVAR#<=>FLAG.

among_interval_signature([],[],_31526,_31527).

among_interval_signature([[var-VAR]|VARs],[S|Ss],LOW,UP) :-
    LOW#=<VAR#/\VAR#=<UP#<=>S,
    among_interval_signature(VARs,Ss,LOW,UP).
B.20 among_low_up

Meta-Data:

ctr_date(among_low_up,[‘20030820’,‘20040530’,‘20060804’]).

ctr_origin(among_low_up,’\cite{BeldiceanuContejean94}',[]).

ctr_arguments(
among_low_up,
[‘LOW’-int,
 ‘UP’-int,
 ‘VARIABLES’-collection(var-dvar),
 ‘VALUES’-collection(val-int)]).

ctr_restrictions(
among_low_up,
[‘LOW’>=0,
 ‘LOW’<=size(‘VARIABLES’),
 ‘UP’>=0,
 ‘UP’<=size(‘VARIABLES’),
 ‘UP’=‘LOW’,
 required(‘VARIABLES’,var),
 required(‘VALUES’,val),
distinct(‘VALUES’,val)])).

ctr_example(
among_low_up,
among_low_up(1,
 2,
 [[var-9],[var-2],[var-4],[var-5]],
 [[val-0],[val-2],[val-4],[val-6],[val-8]])).

ctr_typical(
among_low_up,
[‘LOW’<size(‘VARIABLES’),
 ‘UP’>0,
 ‘LOW’<‘UP’,
 size(‘VARIABLES’)>1,
 size(‘VALUES’)>1,
 size(‘VARIABLES’)>size(‘VALUES’),
 ‘LOW’>0\’\’‘UP’<size(‘VARIABLES’)]).

ctr_exchangeable(
among_low_up,
items('VARIABLES',all),
items('VALUES',all),
vals(["LOW"],int(>=0),>,dontcare,dontcare),
vals(
  ["UP"],
  int(=<size('VARIABLES'))),<,
  dontcare,
  dontcare),
vals(["VARIABLES"^var],
  comp('VALUES"^val),
  =,
  dontcare,
  dontcare])).

ctr_graph(
  among_low_up,
  ['VARIABLES','VALUES'],
  2,
  ['PRODUCT'>>collection(variables,values)],
  [variables^var=values^val],
  ["NARC">='LOW','NARC'=<'UP'],
  ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(
  among_low_up,
  [reformulation(among_low_up_r),automaton(among_low_up_a)]).

ctr_contractible(among_low_up,['UP'=0], 'VARIABLES', any).

ctr_contractible(
  among_low_up,
  ['UP'=size('VARIABLES')],
  VARIABLES,
  any).

ctr_aggregate(among_low_up,[],[+,+,union,union]).

among_low_up_r(LOW,UP,VARIABLES,VALUES) :-
  integer(LOW),
  integer(UP),
  collection(VARIABLES,[dvar]),
  collection(VALUES,[int]),
  get_attr1(VARIABLES,VARS),
  get_attr1(VALUES,VALS),
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

length(VARIABLES,N),
LOW>=0,
LOW=<N,
UP>=0,
UP=<N,
UP>=LOW,
all_different(VALS),
among_low_up1(VARS,VALS,SUM_BVARS),
call(LOW#=SUM_BVARS),
call(UP#=SUM_BVARS).

among_low_up1([],_38981,0).

among_low_up1([V|R],VALS,B+S) :-
build_or_var_in_values(VALS,V,OR),
call(OR#<=>B),
among_low_up1(R,VALS,S).

among_low_up_a(FLAG,LOW,UP,VARIABLES,VALUES) :-
integer(LOW),
integer(UP),
collection(VARIABLES,[dvar]),
collection(VALUES,[int]),
get_attr1(VALUES,LIST_VALUES),
length(VARIABLES,N),
LOW>=0,
LOW=<N,
UP>=0,
UP=<N,
UP>=LOW,
all_different(LIST_VALUES),
list_to_fdset(LIST_VALUES,SET_OF_VALUES),
among_low_up_signature( VARIABLES, SIGNATURE, SET_OF_VALUES),
NVAR in LOW..UP,
automaton( SIGNATURE, _42425, SIGNATURE, [source(s),sink(s)], [arc(s,0,s),arc(s,1,s,[C+1])], [C], [0], [COUNT]),
COUNT# = NVAR$ <=> FLAG.

among_low_up_signature([], [], 38982).

among_low_up_signature(\[var-VAR]\|VARs|, [S|Ss], SET_OF_VALUES) :-
    VAR in_set SET_OF_VALUES$ <=> S,
    among_low_up_signature(VARs, Ss, SET_OF_VALUES).
B.21 among_modulo

\textbf{Meta-Data:}

\begin{verbatim}
ctr_date(among_modulo,['20030820','20040530','20060804']).
ctr_origin(among_modulo,'Derived from %c.',[among]).

ctrl_arguments(among_modulo,
   ['NVAR'-dvar,
   'VARIABLES'-collection(var-dvar),
   'REMAINDER'-int,
   'QUOTIENT'-int]).

ctrl_restrictions(among_modulo,
   ['NVAR']>=0,
   'NVAR'=<size('VARIABLES'),
   required('VARIABLES',var),
   'REMAINDER'>=0,
   'REMAINDER'<'QUOTIENT',
   'QUOTIENT'>0).

ctrl_example(among_modulo,
   among_modulo(3,
      [[var-4],[var-5],[var-8],[var-4],[var-1]],
      0,
      2)).

ctrl_typical(among_modulo,
   ['NVAR']>=0,
   'NVAR'=<size('VARIABLES'),
   size('VARIABLES')>1,
   'QUOTIENT'>1,
   'QUOTIENT'=<maxval('VARIABLES'\^var)).

ctrl_exchangeable(among_modulo,
   [items('VARIABLES',all),
    vals(
      ['VARIABLES'\^var],
      comp('QUOTIENT' mod 'REMAINDER'),
      'REMAINDER')].
\end{verbatim}
=,
dontcare,
dontcare))).

ctr_graph(
among_modulo,
[‘VARIABLES’],
1,
[‘SELF’>>collection(variables)],
[variables^var mod ’QUOTIENT’=’REMAINDER’],
[‘NARC’=’NVAR’],
[]).

ctr_eval(
among_modulo,
[reformulation(among_modulo_r),automaton(among_modulo_a)]).

ctr_pure_functional_dependency(among_modulo,[]).

ctr_functional_dependency(among_modulo,1,[2,3,4]).

ctr_contractible(among_modulo,[’NVAR’=0],’VARIABLES’,any).

ctr_contractible(
among_modulo,
[’NVAR’=size(’VARIABLES’)],
VARIABLES,
any).

ctr_aggregate(among_modulo,[],[+,union,id,id]).

among_modulo_r(NVAR,VARIABLES,REMAINDER,QUOTIENT) :-
check_type(dvar,NVAR),
collection(VARIABLES,[dvar]),
get_attr1(VARIABLES,VARS),
length(VARIABLES,N),
length(QUOTIENT),
nondet(NVAR#,0,NVAR#=<N,0,REMAINDER>=0,REMAINDER<QUOTIENT,QUOTIENT>0,gen_remainder(VARS,QUOTIENT,REMVARS),among_modulo1(REMVARS,REMAINDER,SUM_BVARS),call(NVAR#=SUM_BVARS).
among_modulo1([],_31939,0).

among_modulo1([V|R],REMAINDER,B+S) :-
    V#=REMAINDER#<=B,
    among_modulo1(R,REMAINDER,S).

among_modulo_a(FLAG,NVAR,VARIABLES,REMAINDER,QUOTIENT) :-
    check_type(dvar,NVAR),
    collection(VARIABLES,[dvar]),
    integer(REMAINDER),
    integer(QUOTIENT),
    length(VARIABLES,N),
    NVAR#>=0,
    NVAR#=<N,
    REMAINDER>=0,
    REMAINDER<QUOTIENT,
    QUOTIENT>0,
    among_modulo_signature(
        VARIABLES,
        SIGNATURE,
        REMAINDER,
        QUOTIENT),
    automaton(
        SIGNATURE,
        _34748,
        SIGNATURE,
        [source(s),sink(s)],
        [arc(s,0,s),arc(s,1,s,[C+1])],
        [C],
        [0],
        [COUNT]),
    COUNT#=NVAR#<=>FLAG.

among_modulo_signature([],[],_31940,_31941).

among_modulo_signature(
    [[var-VAR]|VARs],
    [S|Ss],
    REMAINDER,
    QUOTIENT) :-
    VAR mod QUOTIENT#=REMAINDER#<=S,
    among_modulo_signature(VARs,Ss,REMAINDER,QUOTIENT).
B.22 among_seq

◊ **META-DATA:**

ctr_date(among_seq,['20000128','20030820']).

ctr_origin(among_seq,'\cite{BeldiceanuContejean94}',[]).

ctr_synonyms(among_seq,[sequence]).

ctr_arguments(
  among_seq,
  ['LOW'-int,'UP'-int,'SEQ'-int,'VARIABLES'-collection(var-dvar),
   'VALUES'-collection(val-int)]).

ctr_restrictions(
  among_seq,
  ['LOW'>=0,'LOW'=<size('VARIABLES'),
   'UP'='LOW','SEQ'>0,'SEQ'>='LOW','SEQ'=<size('VARIABLES'),
   required('VARIABLES',var),
   required('VALUES',val),
   distinct('VALUES',val)]).

ctr_example(
  among_seq,
  among_seq(1,2,4,
    [[var-9],[var-2],[var-4],[var-5],[var-5],[var-7],[var-7],
     [var-2]],
    [[val-0],[val-2],[val-4],[val-6],[val-8]])).

ctr_typical(}
among_seq, ['LOW'< 'SEQ', 'UP'>0, 'SEQ'>1, 'SEQ'< size('VARIABLES'), size('VARIABLES')>1, size('VALUES')>0, size('VARIABLES')>size('VALUES'), 'LOW'>0#/'UP'< 'SEQ').

ctr_exchangeable( among_seq, [items('VARIABLES',reverse), items('VALUES',all), vals(['LOW'],int(>=(0)),>,dontcare,dontcare), vals(['UP'],int(=(<('SEQ'))),<,dontcare,dontcare), vals([VARIABLES^var], comp('VALUES^val), =, dontcare, dontcare)])).

ctr_graph( among_seq, ['VARIABLES'], SEQ, ['PATH'=>collection], [among_low_up('LOW','UP',collection,'VALUES')], ['NARC'=size('VARIABLES')-'SEQ'+1], []).

ctr_eval(among_seq, [reformulation(among_seq_r)]).

ctr_contractible(among_seq, ['UP'=0],'VARIABLES',any).
ctr_contractible(among_seq, ['SEQ'=1],'VARIABLES',any).
ctr_contractible(among_seq, [],'VARIABLES',prefix).
ctr_contractible(among_seq, [],'VARIABLES',suffix).

among_seq_r(LOW,UP,SEQ,VARIABLES,VALUES) :-
    integer(LOW), integer(UP), integer(SEQ),
collection(VARIABLES,[dvar]),
collection(VALUES,[int]),
get_attr1(VALUES,VALS),
length(VARIABLES,N),
LOW>=0,
LOW=<N,
SEQ>0,
SEQ>=N,
all_different(VALS),
among_seq1(LOW,UP,SEQ,VARIABLES,VALUES).

among_seq1(_LOW,_UP,SEQ,VARIABLES,_VALUES) :-
  length(VARIABLES,N),
  N<SEQ,
  !.

among_seq1(LOW,UP,SEQ,VARIABLES,VALUES) :-
  length(VARIABLES,N),
  N>=SEQ,
  among_seq2(VARIABLES,SEQ,SEQVARIABLES),
  eval(among_low_up(LOW,UP,SEQVARIABLES,VALUES)),
  VARIABLES=[[_28403|RVARIABLES],
  among_seq1(LOW,UP,SEQ,RVARIABLES,VALUES).

among_seq2(_28349,0,[]) :-
  !.

among_seq2([VAR|VARS],SEQ,[VAR|RVARS]) :-
  SEQ>0,
  SEQ1 is SEQ-1,
  among_seq2(VARS,SEQ1,RVARS).
B.23 among_var

◇ Meta-Data:

ctr_date(among_var,['20090418']).

ctr_origin(among_var,'Generalisation of %c',[among]).

ctr_arguments(
    among_var,
    ['NVAR'-dvar,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-dvar)]).

ctr_restrictions(
    among_var,
    ['NVAR']>=0,
    'NVAR'=<size('VARIABLES'),
    required('VARIABLES',var),
    required('VALUES',val)).

ctr_example(
    among_var,
    among_var(3,
        [[var-4],[var-5],[var-5],[var-4],[var-1]],
        [[val-1],[val-5],[val-8],[val-1]]).

ctr_typical(
    among_var,
    [size('VARIABLES')>1,
     size('VALUES')>1,
     size('VARIABLES')>size('VALUES')]).

ctr_exchangeable(
    among_var,
    [items('VARIABLES',all),
     items('VALUES',all),
     vals([‘VARIABLES’^var,’VALUES’^val],
          int,=
          all,
          dontcare),
     vals([‘VARIABLES’^var],
          int,=\=,
          all,
          dontcare)].

\[\text{\textcopyright}\text{American Society for Information Science and Technology}\]
comp(’VALUES’\^val), =, dontcare, dontcare).

ctr_graph(
    among_var,
    [’VARIABLES’, ’VALUES’],
    2,
    [’PRODUCT’>>collection(variables,values)],
    [variables\^var=values\^val],
    [’NSOURCE’=’NVAR’],
    [’ACYCLIC’, ’BIPARTITE’, ’NO_LOOP’]).

ctr_eval(among_var,[reformulation(among_var_r)]).

ctr_pure_functional_dependency(among_var,[]).

ctr_functional_dependency(among_var,1,[2,3]).

ctr_contractible(among_var,[’NVAR’=0],’VARIABLES’,any).

ctr_contractible(
    among_var,
    [’NVAR’=size(’VARIABLES’)],
    VARIABLES,
    any).

ctr_aggregate(among_var,[[,+ union union]]).

among_var_r(NVAR,VARIABLES,VALUES) :-
    check_type(dvar,NVAR),
    collection(VARIABLES,[dvar]),
    collection(VALUES,[dvar]),
    get_attr1(VARIABLES,VARS),
    get_attr1(VALUES,VALS),
    length(VARIABLES,N),
    NVAR#>=0,
    NVAR#=<N,
    among_var1(VARS,VALS,SUM_BVARS),
    call(NVAR#=SUM_BVARS).

among_var1([],_35340,0).

among_var1([V|R],VALS,B+S) :-
    build_or_var_in_values(VALS,V,OR),
    build_or_var_in_values(VALS,R,OR),
call(OR#<=B),
among_var1(R,VALS,S).
B.24 and

◊ META-DATA:

ctr_date(and,[‘20051226’]).

ctr_origin(and,’Logic’,[]).

ctr_synonyms(and,[rel]).

ctr_arguments(
    and,
    [‘VAR’-dvar,’VARIABLES’-collection(var-dvar)]).

ctr_restrictions(
    and,
    [‘VAR’>=0,
     ‘VAR’=<1,
     size(‘VARIABLES’)>=2,
     required(‘VARIABLES’,var),
     ‘VARIABLES’\var>=0,
     ‘VARIABLES’\var=<1]).

ctr_example(
    and,
    [and(0,[var-0],[var-0]),
     and(0,[var-0],[var-1]),
     and(0,[var-1],[var-0]),
     and(1,[var-1],[var-1]),
     and(0,[var-1],[var-0],[var-1])].

ctr_exchangeable(and,[items(‘VARIABLES’,all)]).

ctr_eval(and,[reformulation(and_r),automaton(and_a)]).

ctr_pure_functional_dependency(and,[]).

ctr_functional_dependency(and,1,[2]).

ctr_extensible(and,[‘VAR’=0,’VARIABLES’,any]).

ctr_aggregate(and,[],[#/\,union]).

and_r(VAR,VARIABLES) :-
    check_type(dvar(0,1),VAR),
    collection(VARIABLES,[dvar(0,1)]),
length(VARIABLES,N),
N>=2,
get_attr1(VARIABLES,VARS),
and1(VARS,ANDVARS),
call(ANDVARS#<=>VAR).

and1([VAR],VAR) :-
    !.

and1([VAR|VARS],VAR#/\S) :-
    and1(VARS,S).

and_a(FLAG,VAR,VARIABLES) :-
    check_type(dvar(0,1),VAR),
    collection(VARIABLES,[dvar(0,1)]),
    length(VARIABLES,N),
    N>=2,
    get_attr1(VARIABLES,LIST),
    append([VAR],LIST,LIST_VARIABLES),
    AUTOMATON=automaton(
        LIST_VARIABLES,
        _21570,
        LIST_VARIABLES,
        [source(s),sink(k),sink(j)],
        [arc(s,0,i),
         arc(s,1,j),
         arc(i,0,k),
         arc(i,1,i),
         arc(k,0,k),
         arc(k,1,k),
         arc(j,1,j)],
        [],
        [],
        []),
    automaton_bool(FLAG,[0,1],AUTOMATON).
B.25 arith

◊ **META-DATA:**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctr_date(arith,['20040814','20060804'])</td>
<td></td>
</tr>
<tr>
<td>ctr_origin(arith)</td>
<td>Used in the definition of several automata, [1].</td>
</tr>
<tr>
<td>ctr_synonyms(arith,[rel])</td>
<td></td>
</tr>
<tr>
<td>ctr_arguments(arith)</td>
<td>['VARIABLES'-collection(var-dvar), 'RELOP'-atom, 'VALUE'-int].</td>
</tr>
<tr>
<td>ctr_restrictions(arith)</td>
<td>[required('VARIABLES',var), in_list('RELOP',[=,=,&lt;,&gt;=,&gt;,=&lt;])]</td>
</tr>
<tr>
<td>ctr_example(arith)</td>
<td>arith([[var-4],[var-5],[var-7],[var-4],[var-5]],&lt;,9).</td>
</tr>
<tr>
<td>ctr_typical(arith)</td>
<td>[size('VARIABLES')&gt;1,in_list('RELOP',[=])].</td>
</tr>
<tr>
<td>ctr_exchangeable(arith)</td>
<td>[items('VARIABLES',all), vals(['VARIABLES'^var,int,==,dontcare,in])].</td>
</tr>
<tr>
<td>ctr_graph(arith)</td>
<td>['VARIABLES'], 1, ['SELF']&gt;&gt;collection(variables), ['RELOP'(variables^var,'VALUE')], ['NARC'=size('VARIABLES')], [1].</td>
</tr>
<tr>
<td>ctr_eval(arith)</td>
<td>[reformulation(arith_r),automaton(arith_a)]).</td>
</tr>
</tbody>
</table>
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

ctr_contractible(arith,[],’VARIABLES’,any).

arith_r(VARIABLES,RELOP,VALUE) :-
collection(VARIABLES,[dvar]),
memberchk(RELOP,[=,\=,,>,>=,>\=,\\=\=]),
integer(VALUE),
get_attr1(VARIABLES,VARS),
arith1(VARS,RELOP,VALUE).

arith1([],_35489,_35490).

arith1([VAR|RVARS],RELOP,VALUE) :-
call_term_relop_value(VAR,RELOP,VALUE),
arith1(RVARS,RELOP,VALUE).

arith_a(FLAG,VARIABLES,RELOP,VALUE) :-
collection(VARIABLES,[dvar]),
memberchk(RELOP,[=,\=,,>,>=,>\=,\\=\=]),
integer(VALUE),
arith_signature(VARIABLES,SIGNATURE,RELOP,VALUE),
AUTOMATON=automaton(
    SIGNATURE,
    _36986,
    SIGNATURE,
    [source(s),sink(s)],
    [arc(s,1,s)],
    [],
    [],
    []),
arith_signature(VARIABLES,SIGNATURE,RELOP,VALUE),
arith_signature([VAR],_35490,_35491).

arith_signature([[VAR-VAR]|VARs],[S|Ss],\=,VALUE) :-!
    VAR#=VALUE\=\=S,
arith_signature(VARs,Ss,\=,VALUE).

arith_signature([[VAR-VAR]|VARs],[S|Ss],\=\=,VALUE) :-!
    VAR#\=VALUE\=\=S,
arith_signature(VARs,Ss,\=\=,VALUE).

arith_signature([[VAR-VAR]|VARs],[S|Ss],\<,VALUE) :-!
    VAR\<\=VALUE\<\=S,
arith_signature(VARs,Ss,\<\=,VALUE).
VAR# < VALUE# <= S,
arith_signature(VARs, Ss, <, VALUE).

arith_signature([[var-VAR]|VARs],[S|Ss],[>=,VALUE] :-
!,
VAR# >= VALUE# <= S,
arith_signature(VARs, Ss, >=, VALUE).

arith_signature([[var-VAR]|VARs],[S|Ss],[>,VALUE] :-
!,
VAR# > VALUE# <= S,
arith_signature(VARs, Ss, >, VALUE).

arith_signature([[var-VAR]|VARs],[S|Ss],[=<,VALUE] :-
VAR# <= VALUE# <= S,
arith_signature(VARs, Ss, =<, VALUE).
## B.26 arith_or

**Meta-Data:**

```prolog
ctr_date(arith_or, ['20040814', '20060804']).

ctr_origin(
    arith_or,
    Used in the definition of several automata, []).

ctr_arguments(
    arith_or,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'RELOP'-atom,
     'VALUE'-int]).

ctr_restrictions(
    arith_or,
    [required('VARIABLES1', var),
     required('VARIABLES2', var),
     size('VARIABLES1') = size('VARIABLES2'),
     in_list('RELOP', [=, \

ctr_example(
    arith_or,
    arith_or(
        [[var-0], [var-1], [var-0], [var-0], [var-1]],
        [[var-0], [var-0], [var-0], [var-1], [var-0]],
        =, 0)).

ctr_typical(
    arith_or,
    [size('VARIABLES1') > 0, in_list('RELOP', [=])]).

ctr_exchangeable(
    arith_or,
    [items_sync('VARIABLES1', 'VARIABLES2', all)]).

ctr_graph(
    arith_or,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT' (=)>>collection(variables1, variables2)],
```
(['RELOP'(variables1\^var,'VALUE')\/
 'RELOP'(variables2\^var,'VALUE')],
['NARC'=size('VARIABLES1')],
['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(
    arith_or,
    [reformulation(arith_or_r),automaton(arith_or_a)]).

ctr_contractible(arith_or,[],['VARIABLES1','VARIABLES2'],any).

arith_or_r(VARIABLES1,VARIABLES2,RELOP,VALUE) :-
    collection(VARIABLES1,[dvar]),
    collection(VARIABLES2,[dvar]),
    memberchk(RELOP,[=,\=,<,\>=,>,\=<]),
    integer(VALUE),
    length(VARIABLES1,N1),
    length(VARIABLES2,N2),
    N1=N2,
    memberchk(RELOP,[=,\=,<,\>=,>,\=<]),
    get_attr1(VARIABLES1,VARS1),
    get_attr1(VARIABLES2,VARS2),
    arith_or1(VARS1,VARS2,RELOP,VALUE).

arith_or1([],[],_35046,_35047).

arith_or1([VAR1|RVAR1],[VAR2|RVAR2],=,VALUE) :-
    !,
    VAR1#=VALUE#/VAR2#=VALUE,
    arith_or1(RVAR1,RVAR2,=,VALUE).

arith_or1([VAR1|RVAR1],[VAR2|RVAR2],\=,VALUE) :-
    !,
    VAR1\=VALUE#/VAR2\=VALUE,
    arith_or1(RVAR1,RVAR2,\=,VALUE).

arith_or1([VAR1|RVAR1],[VAR2|RVAR2],<,VALUE) :-
    !,
    VAR1<VALUE#/VAR2<VALUE,
    arith_or1(RVAR1,RVAR2,[VAR2|RVAR2],<,VALUE).

arith_or1([VAR1|RVAR1],[VAR2|RVAR2],\>=,VALUE) :-
    !,
    VAR1\>=VALUE#/VAR2\>=VALUE,
    arith_or1(RVAR1,RVAR2,[VAR2|RVAR2],\>=,VALUE).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

arith_or1([VAR1|VAR1], [VAR2|VAR2], >, VALUE) :-
!,
VAR1#>VALUE#\VAR2#>VALUE,
arith_or1(VAR1, VAR2, [VAR2|VAR2], >, VALUE).

arith_or1([VAR1|VAR1], [VAR2|VAR2], =<, VALUE) :-
VAR1#=<VALUE#\VAR2#=<VALUE,
arith_or1(VAR1, VAR2, =<, VALUE).

arith_or_a(FLAG, VARIABLES1, VARIABLES2, RELOP, VALUE) :-
collection(VARIABLES1, [dvar]),
collection(VARIABLES2, [dvar]),
memberchk(RELOP, [=, =\=, <, >, >\=, =\<]),
i(NT),
length(VARIABLES1, N1),
length(VARIABLES2, N2),
N1=N2,
arith_or_signature(
    VARIABLES1,
    VARIABLES2,
    SIGNATURE,
    RELOP,
    VALUE),
AUTOMATON=
automaton(
    SIGNATURE,
    _37605,
    SIGNATURE,
    [source(s), sink(s)],
    [arc(s, 1, s)],
    [],
    [],
    []),
automaton_bool(FLAG, [0, 1], AUTOMATON).

arith_or_signature([], [], [], _35047, _35048).

arith_or_signature(
    [[var-VAR1] | VAR1s],
    [[var-VAR2] | VAR2s],
    [S | Ss]
    =,
    VALUE) :-
!,
VAR1#=VALUE#\VAR2#=#VALUE#\=>S,
arih_or_signature(VAR1s, VAR2s, Ss, =, VALUE).
arith_or_signature(  [[var-VAR1]|VAR1s],  [[var-VAR2]|VAR2s],  [S|Ss],  =\=,  VALUE) :-  !,  VAR1#\=VALUE#/VAR2#\=VALUE#<=>S,  arith_or_signature(VAR1s,VAR2s,Ss,\=,VALUE).

arith_or_signature(  [[var-VAR1]|VAR1s],  [[var-VAR2]|VAR2s],  [S|Ss],  <,  VALUE) :-  !,  VAR1#<VALUE#/VAR2#<VALUE#<=>S,  arith_or_signature(VAR1s,VAR2s,Ss,<,VALUE).

arith_or_signature(  [[var-VAR1]|VAR1s],  [[var-VAR2]|VAR2s],  [S|Ss],  >=,  VALUE) :-  !,  VAR1#>=VALUE#/VAR2#>=VALUE#<=>S,  arith_or_signature(VAR1s,VAR2s,Ss,\=,VALUE).

arith_or_signature(  [[var-VAR1]|VAR1s],  [[var-VAR2]|VAR2s],  [S|Ss],  >,  VALUE) :-  !,  VAR1#>VALUE#/VAR2#>VALUE#<=>S,  arith_or_signature(VAR1s,VAR2s,Ss,\=,VALUE).

arith_or_signature(  [[var-VAR1]|VAR1s],  [[var-VAR2]|VAR2s],  [S|Ss],  <=,  VALUE) :-  !,  VAR1#<=VALUE#/VAR2#<=VALUE#<=>S,  arith_or_signature(VAR1s,VAR2s,Ss,\=,VALUE).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

\[ \text{VALUE} \) :-
VAR1#=<VALUE# \( \& \) VAR2#=<VALUE# \( \Rightarrow \) S,
arith_or_signature(VAR1s,VAR2s,Ss,=,<,VALUE).\]
B.27  arith_sliding

◊ META-DATA:

ctr_date(arith_sliding,['20040814']).

ctr_origin(
  arith_sliding,
  Used in the definition of some automaton, []).

ctr_arguments(
  arith_sliding,
  [‘VARIABLES’-collection(var-dvar),
   ’RELOP’-atom,
   ’VALUE’-int]).

ctr_restrictions(
  arith_sliding,
  [required(‘VARIABLES’,var),
   in_list(’RELOP’,=[=,\=<,\>=,\>,\=<])]).

ctr_example(
  arith_sliding,
  arith_sliding(
    [[var-0],
     [var-0],
     [var-1],
     [var-2],
     [var-0],
     [var-0],
     [var- -3]],
    <,
    4)).

ctr_typical(
  arith_sliding,
  [size(‘VARIABLES’)\>1,in_list(’RELOP’,[<,\>=,\>,\=<])]).

ctr_graph(
  arith_sliding,
  [‘VARIABLES’],
  ’,’
  [PATH_1]>>collection],
  [arith(collection,’RELOP’,’VALUE’)],
  ['NARC'=size('VARIABLES')],
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

\[
\begin{align*}
&\text{ctr_eval(} \\
&\quad \text{arith_sliding,} \\
&\quad \text{[reformulation(arith_sliding_r),} \\
&\quad \text{automaton(arith_sliding_a)]).} \\
&\text{ctr_contractible(} \\
&\quad \text{arith_sliding,} \\
&\quad \text{[in_list('RELOP',[<,\leq]),minval('VARIABLES'\textsuperscript{\textlangle}var\rangle\geq0],} \\
&\quad \text{VARIABLES,} \\
&\quad \text{any).} \\
&\text{ctr_contractible(arith_sliding,[],'VARIABLES',suffix).} \\
&\text{arith_sliding_r(VARIABLES,RELOP,VALUE) :-} \\
&\quad \text{collection(VARIABLES,[dvar]),} \\
&\quad \text{memberchk(RELOP,[=,\neq,\leq,<,\geq,>,\leq,\geq]),} \\
&\quad \text{integer(VALUE),} \\
&\quad \text{get_attr1(VARIABLES,VARS),} \\
&\quad \text{reverse(VARS,RVARS),} \\
&\quad \text{arith_sliding1(RVARS,RELOP,VALUE).} \\
&\text{arith_sliding1([],_19027,_19028).} \\
&\text{arith_sliding1([VAR|RVARS],RELOP,VALUE) :-} \\
&\quad \text{arith_sliding2([VAR|RVARS],SUM),} \\
&\quad \text{call_term_relop_value(SUM,RELOP,VALUE),} \\
&\quad \text{arith_sliding1(RVARS,RELOP,VALUE).} \\
&\text{arith_sliding2([],0).} \\
&\text{arith_sliding2([VAR|RVARS],VAR+R) :-} \\
&\quad \text{arith_sliding2(RVARS,R).} \\
&\text{arith_sliding_a(FLAG,VARIABLES,\neq,VALUE) :-} \\
&\quad \text{!,} \\
&\quad \text{collection(VARIABLES,[dvar]),} \\
&\quad \text{integer(VALUE),} \\
&\quad \text{length(VARIABLES,N),} \\
&\quad \text{length(SIGNATURE,N),} \\
&\quad \text{domain(SIGNATURE,0,0),} \\
&\quad \text{arith_sliding_signature(VARIABLES,VARS,SIGNATURE),} \\
&\quad \text{automaton(} \\
&\quad \quad \text{VARS,} \\
&\quad \quad \text{VAR,} \\
\end{align*}
\]
SIGNATURE,  
[source(s), sink(s), sink(t)],  
[arc(s, 0, t, [T+C+VAR]),  
arc(t, 0, t, (C#=VALUE->[T+C+VAR])),  
arc(t, 0, t, (C#\=VALUE->[0+C+VAR])),  
[T, C],  
[1, 0],  
[T1, C1]),  
T1#=1#/\C1#=VALUE#<=>FLAG.

arith_sliding_a(FLAG, VARIABLES, =\=, VALUE) :-  
!,  
collection(VARIABLES, [dvar]),  
integer(VALUE),  
length(VARIABLES, N),  
length(SIGNATURE, N),  
domain(SIGNATURE, 0, 0),  
arith_sliding_signature(VARIABLES, VARS, SIGNATURE),  
automaton(  
VARS,  
VAR,  
SIGNATURE,  
[source(s), sink(s), sink(t)],  
[arc(s, 0, t, [T+C+VAR]),  
arc(t, 0, t, (C#=VALUE->[T+C+VAR])),  
arc(t, 0, t, (C#\=VALUE->[0+C+VAR])),  
[T, C],  
[1, 0],  
[T1, C1]),  
T1#=1#/\C1#=VALUE#<=>FLAG.

arith_sliding_a(FLAG, VARIABLES, <, VALUE) :-  
!,  
collection(VARIABLES, [dvar]),  
integer(VALUE),  
length(VARIABLES, N),  
length(SIGNATURE, N),  
domain(SIGNATURE, 0, 0),  
arith_sliding_signature(VARIABLES, VARS, SIGNATURE),  
automaton(  
VARS,  
VAR,  
SIGNATURE,  
[source(s), sink(s), sink(t)],  
[arc(s, 0, t, [T+C+VAR]),  
arc(t, 0, t, (C#<VALUE->[T+C+VAR])),  
arc(t, 0, t, (C#<VALUE->[T+C+VAR])),
arithmetic result: $2048$

APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

```prolog
arith_sliding_a(FLAG, VARIABLES, >=, VALUE) :- !,
    collection(VARIABLES, [dvar]),
    integer(VALUE),
    length(VARIABLES, N),
    length(SIGNATURE, N),
    domain(SIGNATURE, 0, 0),
    arith_sliding_signature(VARIABLES, VARS, SIGNATURE),
    automaton(VARS, VAR, SIGNATURE, [source(s), sink(s), sink(t)],
              [arc(s, 0, t, [T, C+VAR]),
               arc(t, 0, t, (C#>=VALUE->[T,C+VAR])),
               arc(t, 0, t, (C#<VALUE->[0,C+VAR])),
               [T, C],
               [1, 0],
               [T1, C1]),
    T1#=1#/\C1#<VALUE#<=FLAG.

arith_sliding_a(FLAG, VARIABLES, >, VALUE) :- !,
    collection(VARIABLES, [dvar]),
    integer(VALUE),
    length(VARIABLES, N),
    length(SIGNATURE, N),
    domain(SIGNATURE, 0, 0),
    arith_sliding_signature(VARIABLES, VARS, SIGNATURE),
    automaton(VARS, VAR, SIGNATURE, [source(s), sink(s), sink(t)],
              [arc(s, 0, t, [T, C+VAR]),
               arc(t, 0, t, (C#>VALUE->[T,C+VAR])),
               arc(t, 0, t, (C#=<VALUE->[0,C+VAR])),
               [T, C],
               [1, 0],
               [T1, C1]),
    T1#=1#/\C1#>=VALUE#<=FLAG.
```

arithmetic result: $2048$
T1#=1#/

2049

T1#=1#/

\C1#>VALUE#<=>FLAG.

\text{arith}\_sliding\_a(FLAG,\text{VARIABLES},=<,VALUE) :-
\text{collection(VARIABLES,}\{dvar\}),
\text{integer(VALUE)},
\text{collection(SIGNATURE,}\{0,0\}),
\text{domain(SIGNATURE,}\{0,0\}),
\text{arith}\_sliding\_signature(\text{VARIABLES},\text{VARS},\text{SIGNATURE})},
\text{automaton}(
    \text{VARS},
    \text{VAR},
    \text{SIGNATURE},
    \{\text{sink(s)},\text{sink(t)}\},
    \{\text{arc(s,0,t,}T,C+VAR\}),
    \{\text{arc(t,0,t,}C<=VALUE->[T,C+VAR])\},
    \{\text{arc(t,0,t,}C>VALUE->[0,C+VAR])\}],[T,C],
    [1,0],
    [T1,C1]),
T1#=1#/

\text{arith}\_sliding\_signature([],[],[]).

\text{arith}\_sliding\_signature([\text{V|Vs}],[\text{0|Ss}]) :-
\text{arith}\_sliding\_signature(\text{VARs},\text{Vs},\text{Ss}).
B.28 assign_and_counts

◊ Meta-Data:

ctr_date(assign_and_counts, ['20000128', '20030820', '20060804']).

ctr_origin(assign_and_counts, 'N. Beldiceanu', []).

ctr_arguments(
    assign_and_counts,
    ['COLOURS'-collection(val-int),
     'ITEMS'-collection(bin-dvar, colour-dvar),
     'RELOP'-atom,
     'LIMIT'-dvar]).

ctr_restrictions(
    assign_and_counts,
    [required('COLOURS', val),
     distinct('COLOURS', val),
     required('ITEMS', [bin, colour]),
     in_list('RELOP', [=, =\, <, =\, >, =\, <=])].

ctr_example(
    assign_and_counts,
    assign_and_counts(
        [[val-4]],
        [[bin-1, colour-4],
         [bin-3, colour-4],
         [bin-1, colour-4],
         [bin-1, colour-5]],
        =<, 2)).

ctr_typical(
    assign_and_counts,
    [size('COLOURS')>0,
     size('ITEMS')>1,
     range('ITEMS' `bin)>1,
     in_list('RELOP', [<=, =<]),
     'LIMIT'>0,
     'LIMIT'<size('ITEMS'))].

ctr_exchangeable(
    assign_and_counts,
    [items('COLOURS', all),
     items('ITEMS', all),
     items('RELOP', all),
     items('LIMIT', all)]]).
vals(['ITEMS'~bin],int,\=,all,dontcare)).

ctr_derived_collections(
    assign_and_counts,
    [col('VALUES'-collection(val-int),
        [item(val='COLOURS'~val)])]).

ctr_graph(
    assign_and_counts,
    ['ITEMS','ITEMS'],
    2,
    ['PRODUCT'=>collection(items1,items2)],
    [items1~bin=items2~bin],
    [',
     ['ACYCLIC','BIPARTITE','NO_LOOP'],
     [SUCC=>
      [source,
       variables-
       col('VARIABLES'-collection(var-dvar),
           [item(var='ITEMS'~colour)]),
       [counts('VALUES',variables,'RELOP','LIMIT')]])].

ctr_eval(
    assign_and_counts,
    [reformulation(assign_and_counts_r)])).

ctr_contractible(
    assign_and_counts,
    [in_list('RELOP',[=<,\=<])],
    ITEMS,
    any).

ctr_extensible(
    assign_and_counts,
    [in_list('RELOP',[\>=,>])],
    ITEMS,
    any).

assign_and_counts_r(COLOURS,ITEMS,RELOP,LIMIT) :-
    collection(COLOURS,[int]),
    collection(ITEMS,[dvar,dvar]),
    memberchk(RELOP,[\=,\=,=\=,\>=,\>=,\=\<=]),
    check_type(dvar,LIMIT),
    get_attr1(COLOURS,COLS),
    all_different(COLS),
    get_attr1(ITEMS,BINS),
    ...
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

```prolog
get_attr2(ITEMS,ITEMSCOLOURS),
get_minimum(BINS,MINBINS),
gen_matrix_bool(MINBINS,MAXBINS,BINS,BMATRIX),
assign_and_counts1(ITEMSCOLOURS,COLS,CLINE),
assign_and_counts2(BMATRIX,CLINE,RELOP,LIMIT).

assign_and_counts1([],_39619,[]).
assign_and_counts1([ITEMCOLOUR|RITEMCOLOURS],COLS,[B|R]) :-
  build_or_var_in_values(COLS,ITEMCOLOUR,OR),
call(OR#<=>B),
  assign_and_counts1(RITEMCOLOURS,COLS,R).

assign_and_counts2([],_39619,_39620,_39621).
assign_and_counts2([BLINE|RBMATRIX],CLINE,RELOP,LIMIT) :-
  assign_and_counts3(BLINE,CLINE,TERM),
call_term_relop_value(TERM,RELOP,LIMIT),
  assign_and_counts2(RBMATRIX,CLINE,RELOP,LIMIT).

assign_and_counts3([],[],0).
assign_and_counts3([B|RBLINE],[C|RCLINE],B*C+R) :-
  assign_and_counts3(RBLINE,RCLINE).
```
B.29 assign_and_nvalues

◊ Meta-Data:

ctr_date(
    assign_and_nvalues,
    ['20000128', '20030820', '20040530', '20050321', '20060804']).

ctr_origin(
    assign_and_nvalues,
    Derived from %c and %c.,
    [assign_and_counts, nvalues]).

ctr_arguments(
    assign_and_nvalues,
    ['ITEMS'-collection(bin-dvar, value-dvar),
     'RELOP'-atom,
     'LIMIT'-dvar]).

ctr_restrictions(
    assign_and_nvalues,
    [required('ITEMS', [bin, value]),
     in_list('RELOP', [=, \=, <, \>=, >, =<])].

ctr_example(
    assign_and_nvalues,
    assign_and_nvalues(
        [[bin-2, value-3],
         [bin-1, value-5],
         [bin-2, value-3],
         [bin-2, value-3],
         [bin-2, value-4]],
        =<, 2)).

ctr_typical(
    assign_and_nvalues,
    [size('ITEMS')>1,
     range('ITEMS'\bin)>1,
     range('ITEMS'\value)>1,
     in_list('RELOP', [\=, \=, <, \>=, >, =<]),
     'LIMIT'>1,
     'LIMIT'<size('ITEMS')]).

ctr_exchangeable(
    assign_and_nvalues,
[items('ITEMS',all),
vals(['ITEMS'ˆbin],int,\=,all,dontcare)).

ctr_graph(
   assign_and_nvalues,
   ['ITEMS','ITEMS'],
   2,
   ['PRODUCT'>>collection(items1,items2)],
   [items1ˆbin=items2ˆbin],
   [],
   ['ACYCLIC','BIPARTITE','NO_LOOP'],
   [SUCC>>
    [source,
     variables-
     col('VARIABLES'–collection(var-dvar),
     [item(var-'ITEMS'ˆvalue)]),
    [nvalues(variables,'RELOP','LIMIT')]].

ctr_eval(
   assign_and_nvalues,
   [reformulation(assign_and_nvalues_r)]).

ctr_contractible(
   assign_and_nvalues,
   [in_list('RELOP',[<,=<=])],
   ITEMS,
   any).

ctr_extensible(
   assign_and_nvalues,
   [in_list('RELOP',[>=,>])],
   ITEMS,
   any).

assign_and_nvalues_r(ITEMS,RELOP,LIMIT) :-
collection(ITEMS,[dvar,dvar]),
memberchk(RELOP,[=,\=,\<,\>=,\>,\=<]),
check_type(dvar,LIMIT),
get_attr1(ITEMS,BINS),
get_attr2(ITEMS,VALUES),
get_minimum(BINS,MINBINS),
get_maximum(BINS,MAXBINS),
gen_matrix_bool(MINBINS,MAXBINS,BINS,BMATRIX),
get_minimum(VALUES,MINVALUES),
JOKER is MINVALUES-1,
LIM is LIMIT+1,
assign_and_nvalues1(BMATRIX,VALUES,JOKER,RELOP,LIM).

assign_and_nvalues1([],_40152,_40153,_40154,_40155).

assign_and_nvalues1([BLINE|RBMATRIX],VALUES,JOKER,RELOP,LIM) :-
    assign_and_nvalues2(BLINE,VALUES,JOKER,VALS),
    length(VALS,M),
    N in 0..M,
    nvalue(N,VALS),
    call_term_relop_value(N,RELOP,LIM),
    assign_and_nvalues1(RBMATRIX,VALUES,JOKER,RELOP,LIM).

assign_and_nvalues2([],[],JOKER,[JOKER]).

assign_and_nvalues2([VAR|RVAR],[VAL|RVAL],JOKER,[V|R]) :-
    VAR#=0#/\V#=JOKER#/\VAR#=1#/\V#=VAL,
    assign_and_nvalues2(RVAR,RVAL,JOKER,R).
B.30 \textit{atleast}

\textbf{Meta-Data:}

ctr\_date(atleast,[’20030820’,’20040807’,’20060804’]).

ctr\_origin(atleast,’\\index{CHIP|indexuse}CHIP’,[]).

ctr\_synonyms(atleast,[count]).

ctr\_arguments(
  atleast,
  [’N’\-int,’VARIABLES’\-collection(var-dvar),’VALUE’\-int]).

ctr\_restrictions(
  atleast,
  [’N’\>=0,’N’\=<size(’VARIABLES’),required(’VARIABLES’,var)]).

ctr\_example(
  atleast,
  atleast(2,[[var-4],[var-2],[var-4],[var-5]],4)).

ctr\_typical(
  atleast,
  [’N’\>0,’N’\<size(’VARIABLES’),size(’VARIABLES’)>1]).

ctr\_exchangeable(
  atleast,
  [items(’VARIABLES’,all),
    vals([’N’],int (>=0)),>,dontcare,dontcare),
    vals(\n      [’VARIABLES’\^var],
      comp(’VALUE’),
      >=,
      dontcare, 
      dontcare))].

ctr\_graph(
  atleast,
  [’VARIABLES’],
  1,
  [’SELF’\>collection(variables)],
  [variables\^var=’VALUE’],
  [’NARC’=’N’],
  []).
ctr_eval(
    atleast,
    [reformulation(atleast_r), automaton(atleast_a)]).

ctr_extensible(atleast,[],'VARIABLES',any).

atleast_r(N,VARIABLES,VALUE) :-
    integer(N),
    collection(VARIABLES,[dvar]),
    integer(VALUE),
    length(VARIABLES,NVARIABLES),
    N>=0,
    N=<NVARIABLES,
    get_attr1(VARIABLES,VARS),
    atleast1(VARS,VALUE,SUM_BVARS),
    call(SUM_BVARS#>=N).

atleast1([],_33070,0).

atleast1([V|R],VALUE,B+S) :-
    V#=VALUE#<=>B,
    atleast1(R,VALUE,S).

atleast_a(FLAG,N,VARIABLES,VALUE) :-
    integer(N),
    collection(VARIABLES,[dvar]),
    integer(VALUE),
    length(VARIABLES,M),
    N>=0,
    N=<M,
    atleast_signature(VARIABLES,SIGNATURE,VALUE),
    NVAR in N..M,
    automaton(
        SIGNATURE,
        _35069,
        SIGNATURE,
        [source(s),sink(s)],
        [arc(s,0,s),arc(s,1,s,[C+1])],
        [C],
        [0],
        [COUNT]),
    COUNT#=NVAR#<=>FLAG.

atleast_signature([],[],_33071).

atleast_signature([[var-VAR]|VARs],[S|Ss],VALUE) :-
    integer(VALUE),
    length(VARIABLES,NVARIABLES),
    N>=0,
    N=<NVARIABLES,
    get_attr1(VARIABLES,VARS),
    atleast1(VARS,VALUE,SUM_BVARS),
    call(SUM_BVARS#>=N).
VAR#=VALUE#<=>S,
atleast_signature(VARs,Ss,VALUE).
B.31  atleast_nvalue

◊ **META-DATA:**

```prolog
ctr_date(atleast_nvalue, ['20050618', '20060804']).

ctr_origin(atleast_nvalue, '\cite{Regin95}', []).

ctr_synonyms(atleast_nvalue, [k_diff]).

ctr_arguments(
  atleast_nvalue,
  ['NVAL'-dvar, 'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
  atleast_nvalue,
  [required('VARIABLES', var),
   'NVAL'>=0, 
   'NVAL'=<size('VARIABLES'),
   'NVAL'=<range('VARIABLES' \textsuperscript{var})]).

ctr_example(
  atleast_nvalue,
  atleast_nvalue(2, [[var-3], [var-1], [var-7], [var-1], [var-6]])).

ctr_typical(
  atleast_nvalue,
  ['NVAL'>0, 
   'NVAL'=<size('VARIABLES'),
   'NVAL'=<range('VARIABLES' \textsuperscript{var}),
   size('VARIABLES')>1]).

ctr_exchangeable(
  atleast_nvalue,
  [vals(['NVAL'], int(>=0)), >, dontcare, dontcare), 
  items('VARIABLES', all), 
  vals(['VARIABLES' \textsuperscript{var}, int, =\textless, all, dontcare])].

ctr_graph(
  atleast_nvalue,
  ['VARIABLES'],
  2,
  ['CLIQUE'>>collection(variables1, variables2)],
  [variables1 \textsuperscript{var}=variables2 \textsuperscript{var}],
  [variables1 \textsuperscript{var}]=variables2 \textsuperscript{var}])
```
[‘NSCC’>=‘NVAL’],
[‘EQUIVALENCE’]).

ctr_eval(atleast_nvalue,[reformulation(atleast_nvalue_r)]).

ctr_extensible(atleast_nvalue,[],’VARIABLES’,any).

atleast_nvalue_r(NVAL,VARIABLES) :-
    check_type(dvar,NVAL),
    collection(VARIABLES,[dvar]),
    get_attr1(VARIABLES,VARS),
    length(VARIABLES,N),
    NVAL#>=0,
    NVAL#=<N,
    list_dvar_range(VARS,R),
    NVAL#=<R,
    V in 0..N,
    V#>=NVAL,
    nvalue(V,VARS).
B.32  atleast_nvvector

◊ **Meta-Data:**

ctr_date(atleast_nvvector,['20081226']).

ctr_origin(atleast_nvvector,'Derived from %c',[nvector]).

ctr_types(atleast_nvvector,['VECTOR'-collection(var-dvar)]).

ctr_arguments(  
atleast_nvvector,  
['NVEC'-dvar,'VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(  
atleast_nvvector,  
[size('VECTOR')>=1,  
'NVEC'>=0,  
'NVEC'=<size('VECTORS'),  
required('VECTORS',vec),  
same_size('VECTORS',vec)].

ctr_example(  
atleast_nvvector,  
atleast_nvvector(  
2,  
[[vec-[[var-5],[var-6]]],  
[[vec-[[var-5],[var-6]]],  
[[vec-[[var-9],[var-3]]],  
[[vec-[[var-5],[var-6]]],  
[[vec-[[var-9],[var-4]]]]
  )].

ctr_typical(  
atleast_nvvector,  
[size('VECTOR')>1,  
'NVEC'>1,  
'NVEC'=<size('VECTORS'),  
size('VECTORS')>1]).

ctr_exchangeable(  
atleast_nvvector,  
[vals(['NVEC'],int>(=0)),>,dontcare,dontcare),  
items('VECTORS',all),  
items_sync('VECTORS''vec',all),  
vals(['VECTORS''vec'],int,\=,all,dontcare)].
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

ctr_graph(
    atleast_nvector,
    ['VECTORS'],
    2,
    ['CLIQUE'>>collection(vectors1,vectors2)],
    [lex_equal(vectors1^vec,vectors2^vec)],
    ['NSCC'='NVEC'],
    ['EQUIVALENCE']).

ctr_eval(atleast_nvector,[reformulation(atleast_nvector_r)]).

ctr_extensible(atleast_nvector,[],'VECTORS',any).

atleast_nvector_r(NVEC,VECTORS) :-
    check_type(dvar,NVEC),
    length(VECTORS,N),
    NVEC#>=0,
    NVEC#=<N,
    NV in 0..N,
    nvector_common(NV,VECTORS),
    NV#>=NVEC.
**B.33 atmost**

◊ **META-DATA:**

```prolog
ctr_date(atmost, ['20030820', '20040807', '20060804']).

ctr_origin(atmost, '\\index{CHIP|indexuse}CHIP', []).

ctr_synonyms(atmost, [count]).

ctr_arguments(atmost,
    ['N'-int,'VARIABLES'-collection(var-dvar),'VALUE'-int]).

ctr_restrictions(atmost, ['N'>=0, required('VARIABLES', var)]).

ctr_example(atmost,
    atmost(1, [[var-4], [var-2], [var-4], [var-5]], 2)).

ctr_typical(atmost,
    ['N'>0,
     'N'<size('VARIABLES'),
     size('VARIABLES')>1,
     atleast(1, 'VARIABLES', 'VALUE')]).

ctr_exchangeable(atmost,
    items('VARIABLES', all),
    vals(['N'], int, <, dontcare, dontcare),
    vals(['VARIABLES'\var],
        comp('VALUE'),
        =<, dontcare, dontcare)).

ctr_graph(atmost,
    ['VARIABLES'],
    1,
    ['SELF'\collection(variables)],
    [variables\var='VALUE'],
    ['NARC'=<'N'],
    []).
```
ctr_eval(atmost,[reformulation(atmost_r),automaton(atmost_a)]).

ctr_contractible(atmost,[],'VARIABLES',any).

atmost_r(N,VARIABLES,VALUE) :-
  integer(N),
  collection(VARIABLES,[dvar]),
  integer(VALUE),
  N>=0,
  get_attr1(VARIABLES,VARS),
  atmost1_(VARS,VALUE,SUM_BVARS),
  call(SUM_BVARS#=<N).

atmost1_([],_30608,0).

atmost1_([V|R],VALUE,B+S) :-
  V#=VALUE#<=>B,
  atmost1_(R,VALUE,S).

atmost_a(FLAG,N,VARIABLES,VALUE) :-
  integer(N),
  collection(VARIABLES,[dvar]),
  integer(VALUE),
  N>=0,
  atmost_signature(VARIABLES,SIGNATURE,VALUE),
  length(VARIABLES,M),
  MN is min(M,N),
  NVAR in 0..MN,
  automaton(
    SIGNATURE,
    _32619,
    SIGNATURE,
    [source(s),sink(s)],
    [arc(s,O,s),arc(s,1,s,[C+1])],
    [C],
    [0],
    [COUNT]),
  COUNT#=NVAR#<=>FLAG.

atmost_signature([],[],_30609).

atmost_signature([[var-VAR]|VARs],[S|Ss],VALUE) :-
  VAR#=VALUE#<=>S,
  atmost_signature(VARs,Ss,VALUE).
B.34 atmost1

◊ META-DATA:

ctr_predefined(atmost1).

ctr_date(atmost1, ['20061003']).

ctr_origin(atmost1, '\cite{SadlerGervet01}', []).

ctr_synonyms(atmost1, [pair_atmost1]).

ctr_arguments(atmost1, ['SETS'-collection(s-svar,c-int)]).

ctr_restrictions(atmost1, [required('SETS', [s,c]), 'SETS'ˆc>=1]).

ctr_example(atmost1, atmost1(
  [[s-{5,8},c-2],
   [s-{5},c-1],
   [s-{5,6,7},c-3],
   [s-{1,4},c-2]]).

ctr_typical(atmost1, [size('SETS')>1]).

ctr_exchangeable(atmost1, [items('SETS', all), vals(['SETS'ˆs], int, =\=, all, dontcare)]).

ctr_contractible(atmost1, [], 'SETS', any).
B.35 atmost_nvalue

◊ Meta-Data:

ctr_date(atmost_nvalue,['20050618','20060804','20090926']).

ctr_origin(atmost_nvalue,\cite{BessiereHebrardHnichKiziltanWalsh05},[]).

ctr_synonyms(atmost_nvalue,[soft_alldiff_max_var,soft_alldifferent_max_var,soft_alldistinct_max_var]).

ctr_arguments(atmost_nvalue,['NVAL’-dvar,’VARIABLES’-collection(var-dvar)]).

ctr_restrictions(atmost_nvalue,['NVAL’>=min(1,size(‘VARIABLES’)),required(‘VARIABLES’,var)]).

ctr_example(atmost_nvalue,atmost_nvalue(4,[[var-3],[var-1],[var-3],[var-1],[var-6]])).

ctr_typical(atmost_nvalue,['NVAL’>1,’NVAL’<size(‘VARIABLES’),size(‘VARIABLES’)>1]).

ctr_exchangeable(atmost_nvalue,[vals(['NVAL’],int,<,dontcare,dontcare),items(‘VARIABLES’,all),vals(['VARIABLES’ˆvar],int,\=,all,dontcare),vals(['VARIABLES’ˆvar],int,\=,dontcare,in)]).

ctr_graph(atmost_nvalue,['VARIABLES’],2,['CLIQUE’>>collection(variables1,variables2)],
[variables1 \text抛}=\text抛variables2 \text抛var],
[\text'\text NSCC'=\text'<\text'\text NVAL'],
[\text'\text EQUVALENCE'])).

\text{ctr\_eval}(\text{atmost\_nvalue},[\text{reformulation}(\text{atmost\_nvalue\_r})]).

\text{ctr\_contractible}(\text{atmost\_nvalue},[],'\text{VARIABLES}',\text{any}).

\text{atmost\_nvalue\_r}(\text{NVAL},\text{VARIABLES}) :-
  \text{check\_type}(\text{dvar},\text{NVAL}),
  \text{collection}(\text{VARIABLES},[\text{dvar}]),
  \text{get\_attr}(\text{VARIABLES},\text{VARS}),
  \text{length}(\text{VARIABLES},\text{N}),
  \text{NVAL}\#<\text{N},
  \text{V}\text{ in 0..N},
  \text{V}\#<\text{NVAL},
  \text{nvalue}(\text{V},\text{VARS}).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.36  atmost_nvector

◊ Meta-Data:

ctr_date(atmost_nvector,[’20081226’]).

ctr_origin(atmost_nvector,’Derived from %c’,[nvector]).

ctr_types(atmost_nvector,[’VECTOR’-collection(var-dvar)]).

ctr_arguments(
    atmost_nvector,
    [’NVEC’-dvar,’VECTORS’-collection(vec-’VECTOR’)]).

ctr_restrictions(
    atmost_nvector,
    [size(’VECTOR’)>1,
     ’NVEC’>=min(1,size(’VECTORS’)),
     required(’VECTORS’,vec),
     same_size(’VECTORS’,vec)]).

ctr_example(
    atmost_nvector,
    atmost_nvector(3,
        [[vec-[[var-5],[var-6]]],
         [vec-[[var-5],[var-6]]],
         [vec-[[var-9],[var-3]]],
         [vec-[[var-5],[var-6]]],
         [vec-[[var-9],[var-3]]]]).

ctr_typical(
    atmost_nvector,
    [size(’VECTOR’) > 1,
     ’NVEC’ > 1,
     ’NVEC’ < size(’VECTORS’),
     size(’VECTORS’) > 1]).

ctr_exchangeable(
    atmost_nvector,
    [vals([’NVEC’],int,<,dontcare,dontcare),
     items(’VECTORS’,all),
     items_sync(’VECTORS’ˆvec,all),
     vals([’VECTORS’ˆvec],int,=\=,all,dontcare)]).

ctr_graph(
}
atmost_nvector, 
[‘VECTORS’], 
2, 
[‘CLIQUE’>>collection(vectors1,vectors2)], 
[lex_equal(vectors1^vec,vectors2^vec)], 
[‘NSCC’=<‘NVEC’], 
[‘EQUIVALENCE’]).

ctr_eval(atmost_nvector,[reformulation(atmost_nvector_r)]).

ctr_contractible(atmost_nvector,[],’VECTORS’,any).

atmost_nvector_r(NVEC,VECTORS) :- 
    length(VECTORS,N), 
    NV in 0..N, 
    nvector_common(NV,VECTORS), 
    NV#=NVEC.
B.37 balance

◊ Meta-Data:

ctr_date(balance, ['20000128', '20030820', '20060804', '20110713']).

ctr_origin(balance, 'N. Beldiceanu', []).

ctr_arguments(
    balance,
    ['BALANCE'-dvar, 'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    balance,
    ['BALANCE'>=0,
     'BALANCE'=<max(0, size('VARIABLES')-2),
     required('VARIABLES', var)]).

ctr_example(
    balance,
    balance(2, [[var-3], [var-1], [var-7], [var-1], [var-1]])).

ctr_typical(balance, [size('VARIABLES')>2]).

ctr_exchangeable(
    balance,
    [items('VARIABLES', all),
     vals(['VARIABLES'ˆvar], int, =\=, all, dontcare)]).

ctr_graph(
    balance,
    ['VARIABLES'],
    2,
    ['CLIQUE']>>collection(variables1, variables2)],
    [variables1ˆvar=variables2ˆvar],
    ['RANGE_NSCC'='BALANCE'],
    ['EQUIVALENCE']).

ctr_eval(balance, [reformulation(balance_r)]).

ctr_pure_functional_dependency(balance, []).

ctr_functional_dependency(balance, 1, [2]).

balance_r(0, []) :- !.
balance_r(BALANCE,VARIABLES) :-
  check_type(dvar,BALANCE),
  collection(VARIABLES,[dvar]),
  get_attr1(VARIABLES,VARS),
  length(VARIABLES,N),
  N2 is max(N-2,0),
  BALANCE#>=0,
  BALANCE#=<N2,
  get_minimum(VARS,MINVARS),
  get_maximum(VARS,MAXVARS),
  balance1(MINVARS,MAXVARS,N,VALS,OCCS,OCCS1),
  eval(global_cardinality(VARIABLES,VALS)),
  MIN in 1..N,
  MAX in 1..N,
  eval(minimum(MIN,OCCS1)),
  eval(maximum(MAX,OCCS)),
  BALANCE+MIN#=MAX.

balance1(I,S,_37046,[],[],[]) :-
  I>S,
  !.

balance1(I,S,N,[[val-I|0|0|0|0]],T,U) :-
  I=<S,
  O in 0..N,
  O#0#=O1#=N,
  O#0#=O1#=O,
  I1 is I+1,
  balance1(I1,S,N,R,T,U).
B.38 balance_cycle

◊ Meta-Data:

ctr_date(balance_cycle, ['20111218']).

ctr_origin(
    balance_cycle,
    derived from %c and %c,
    [balance,cycle]).

ctr_arguments(
    balance_cycle,
    ['BALANCE'-dvar,'NODES'-collection(index-int,succ-dvar)]).

ctr_restrictions(
    balance_cycle,
    ['BALANCE']=0,
    'BALANCE'=<max(0,size('NODES')-2),
    required('NODES',[index,succ]),
    'NODES'~index>=1,
    'NODES'~index=<size('NODES'),
    distinct('NODES',index),
    'NODES'~succ=1,
    'NODES'~succ=<size('NODES')]).

ctr_example(
    balance_cycle,
    balance_cycle(
        1,
        [[index-1,succ-2],
        [index-2,succ-1],
        [index-3,succ-5],
        [index-4,succ-3],
        [index-5,succ-4]])).

ctr_typical(balance_cycle,[size('NODES')>2]).

ctr_exchangeable(balance_cycle,[items('NODES',all)]).

ctr_graph(
    balance_cycle,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1~succ=nodes2~index],
[‘\$NTREE\$’=0,’\$RANGE\_NCC\$’=‘BALANCE’],
[‘\$ONE\_SUCC\$’]).

ctr_eval(balance_cycle,[checker(balance_cycle_c)]).

ctr_functional_dependency(balance_cycle,1,[2]).

balance_cycle_c(BALANCE,NODES) :-
  length(NODES,N),
  N2 is max(N-2,0),
  check_type(dvar(0,N2),BALANCE),
  collection(NODES,[int(1,N),dvar(1,N)]),
  get_attr1(NODES,INDEXES),
  get_attr2(NODES,SUCCS),
  sort(INDEXES,Js),
  sort(SUCCS,Js),
  length(Js,N),
  (for(J,1,N),
   foreach(X,SUCCS),
   foreach(Free,Term),
   foreach(Free-1,KeyTerm),foreach(J,Js),param(Term,N) do
     nth1(X,Term,Free)),
  keysort(KeyTerm,KeySorted),
  keyclumped(KeySorted,KeyClumped),
  (foreach(_33260-Ones,KeyClumped),
   foreach(Count,Counts)do
     length(Ones,Count)),
  min_member(Min,Counts),
  max_member(Max,Counts),
  BALANCE is Max-Min.
B.39 balance_interval

◊ Meta-Data:

ctr_date(balance_interval, ['20030820', '20060804']).

ctr_origin(balance_interval, 'Derived from %c.', [balance]).

ctr_arguments(
    balance_interval,
    ['BALANCE' - dvar, 
     'VARIABLES' - collection(var - dvar), 
     'SIZE_INTERVAL' - int]).

ctr_restrictions(
    balance_interval, 
    ['BALANCE' >= 0, 
     'BALANCE' <= <max(0, size('VARIABLES') - 2), 
     required('VARIABLES', var), 
     'SIZE_INTERVAL' > 0]).

ctr_example(
    balance_interval,
    balance_interval(3,
        [[var-6], [var-4], [var-3], [var-3], [var-4]],
        3)).

ctr_typical(
    balance_interval, 
    [size('VARIABLES') > 2, 
     'SIZE_INTERVAL' > 1, 
     'SIZE_INTERVAL' < range('VARIABLES' var)]).

ctr_exchangeable(
    balance_interval, 
    [items('VARIABLES', all),
     vals(
       ['VARIABLES' var], 
       intervals('SIZE_INTERVAL'),
       =, 
       dontcare, 
       dontcare)]).

ctr_graph(
    balance_interval,
[‘VARIABLES’],
2,
[‘CLIQUE’>>collection(variables1,variables2)],
[variables1\var/’SIZE_INTERVAL’=
 variables2\var/’SIZE_INTERVAL’],
[‘RANGE_NSCC’=‘BALANCE’],
[‘EQUIVALENCE’]).

ctr_pure_functional_dependency(balance_interval,[]).
ctr_functional_dependency(balance_interval,1,[2,3]).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.40 balance_modulo

♦ Meta-Data:

ctr_date(balance_modulo,['20030820','20060804']).

ctr_origin(balance_modulo,'Derived from %c.',[balance]).

ctr_arguments(
    balance_modulo,
    ['BALANCE'-dvar,'VARIABLES'-collection(var-dvar),'M'-int]).

ctr_restrictions(
    balance_modulo,
    ['BALANCE']>=0,
    'BALANCE'=<max(0,size('VARIABLES')-2),
    required('VARIABLES',var),
    'M'>0)).

ctr_example(
    balance_modulo,
    balance_modulo(2,
        [[var-6],[var-1],[var-7],[var-1],[var-5]],
        3)).

ctr_typical(
    balance_modulo,
    [size('VARIABLES')>2,'M'>1,'M'<maxval('VARIABLES'-'var')].

ctr_exchangeable(
    balance_modulo,
    [items('VARIABLES',all),
    vals(['VARIABLES'-'var'],mod('M'),=,dontcare,dontcare)]).

ctr_graph(
    balance_modulo,
    ['VARIABLES'],
    2,
    ['CLIQUE']>>collection(variables1,variables2)],
    [variables1-'var mod 'M'=variables2-'var mod 'M',
    ['RANGE_NSCC'='BALANCE'],
    ['EQUIVALENCE']]).

ctr_pure_functional_dependency(balance_modulo,[]).
ctr_functional_dependency(balance_modulo, 1, [2, 3]).
B.41 balance_partition

**Meta-Data:**

```prolog
ctr_date(balance_partition, [’20030820’, ’20060804’]).

ctr_origin(balance_partition, ’Derived from %c.’, [balance]).

ctr_types(balance_partition, [’VALUES’-collection(val-int)]).

ctr_arguments(
    balance_partition,
    [’BALANCE’-dvar,
    ’VARIABLES’-collection(var-dvar),
    ’PARTITIONS’-collection(p-’VALUES’)]).

ctr_restrictions(
    balance_partition,
    [size(’VALUES’) >= 1,
     required(’VALUES’, val),
     distinct(’VALUES’, val),
     ’BALANCE’ >= 0,
     ’BALANCE’ <= max(0, size(’VARIABLES’)-2),
     required(’VARIABLES’, var),
     required(’PARTITIONS’, p),
     size(’PARTITIONS’) >= 2]).

ctr_example(
    balance_partition,
    balance_partition(
        1,
        [[var-6], [var-2], [var-6], [var-4], [var-4]],
        [[p-[[val-1], [val-3]]],
         [p-[[val-4]]],
         [p-[[val-2], [val-6]]])).

ctr_typical(
    balance_partition,
    [size(’VARIABLES’) > 2, size(’VARIABLES’) > size(’PARTITIONS’)]).

ctr_exchangeable(
    balance_partition,
    [items(’VARIABLES’, all),
    items(’PARTITIONS’, all),
    items(’PARTITIONS’ ^ p, all),
    vals(}
['VARIABLES'\textasciitilde var],
part('PARTITIONS'),
=,
dontcare,
dontcare)\}).

\begin{verbatim}
ctr_graph(
    balance_partition,
    ['VARIABLES'],
    2,
    ['CLIQUE'\textasciitilde >> collection(variables1,variables2)],
    [in_same_partition(
        variables1\textasciitilde var,
        variables2\textasciitilde var,
        PARTITIONS)],
    ['RANGE_NS\textsc{cc}'='BALANCE'],
    ['\textsc{equivALENCE}']).

ctr\_pure_functional\_dependency(balance\_partition,[]).

ctr\_functional\_dependency(balance\_partition,i,[2,3]).
\end{verbatim}
B.42 balance_path

◊ Meta-Data:

ctr_date(balance_path, ['20111226']).

ctr_origin(
    balance_path,
    derived from %c and %c,
    [balance, path]).

ctr_arguments(
    balance_path,
    ['BALANCE'-dvar, 'NODES'-collection(index-int, succ-dvar)]).

ctr_restrictions(
    balance_path,
    ['BALANCE' >= 0,
     'BALANCE' <= max(0, size('NODES') - 2),
     required('NODES', [index, succ]),
     'NODES' index >= 1,
     'NODES' index <= size('NODES'),
     distinct('NODES', index),
     'NODES' succ >= 1,
     'NODES' succ <= size('NODES')].)

ctr_example(
    balance_path,
    balance_path(3,
        [[index-1, succ-1],
         [index-2, succ-3],
         [index-3, succ-5],
         [index-4, succ-4],
         [index-5, succ-1],
         [index-6, succ-6],
         [index-7, succ-7],
         [index-8, succ-6]])).

ctr_typical(balance_path, [size('NODES') > 2]).

ctr_exchangeable(balance_path, [items('NODES', all)]).

ctr_graph(
    balance_path,
    ['NODES'],
    ...)
2,
['CLIQUE'>>collection(nodes1,nodes2)],
[nodes1^succ=nodes2^index],
['MAX_NSCC'=<1,'MAX_ID'=<1,'RANGE_NCC'='BALANCE'],
['ONE_SUCC']).

ctr_eval(balance_path,[checker(balance_path_c)]).

ctr_functional_dependency(balance_path,1,[2]).

balance_path_c(BALANCE,NODES) :-
  length(NODES,N),
  N2 is max(N-2,0),
  check_type(dvar(0,N2),BALANCE),
  collection(NODES,[int(1,N),dvar(1,N)]),
  get_attr1(NODES,INDEXES),
  get_attr2(NODES,SUCCS),
  sort(INDEXES,SIND),
  length(SIND,N),
  length(RANKS,N),
  domain(RANKS,1,N),
  balance_path1(INDEXES,SUCCS,RANKS,SUCC_WITHOUT_LOOPS),
  sort(SUCCsWithoutloops,SSUCCsWithoutloops),
  length(SUCCsWithoutloops,NSL),
  length(SSUCCsWithoutloops,NSL),
  (foreach(X,SUCCS),
   foreach(Free,Term),
   foreach(Free-1,KeyTerm),param(Term,N)do
   nth1(X,Term,Free)),
  keysort(KeyTerm,KeySorted),
  keyclumped(KeySorted,KeyClumped),
  (foreach(_40798-Ones,KeyClumped),
   foreach(Count,Counts)do
   length(Ones,Count)),
  min_member(Min,Counts),
  max_member(Max,Counts),
  BALANCE is Max-Min.

balance_path1([],[],_40599,[]) :- !.

balance_path1([I|RI],[I|RS],RANKS,SUCC) :- !,
  balance_path1(RI,RS,RANKS,SUCC).

balance_path1([I|RI],[S|RS],RANKS,[S|SUCC]) :-
nthl(I,RANKS,Ri),
nthl(S,RANKS,Rs),
Ri#<Rs,
balance_path1(RI,RS,RANKS,SUCC).
B.43 balance_tree

◊ **Meta-Data:**

```prolog
ctr_date(balance_tree, ['20111226']).
ctr_origin( balance_tree, derived from %c and %c, [balance,tree]).
ctr_arguments( balance_tree, ['BALANCE'-dvar,'NODES'-collection(index-int,succ-dvar)]).
ctr_restrictions( balance_tree, ['BALANCE'>=0, 'BALANCE'=<max(0,size('NODES')-2), required('NODES',[index,succ]), 'NODES'\^index>=1, 'NODES'\^index=<size('NODES'), distinct('NODES',index), 'NODES'\^succ>=1, 'NODES'\^succ=<size('NODES'))].
ctr_example( balance_tree, balance_tree( 4, [[index-1,succ-1], [index-2,succ-5], [index-3,succ-5], [index-4,succ-7], [index-5,succ-1], [index-6,succ-1], [index-7,succ-7], [index-8,succ-5]]).
ctr_typical(balance_tree, [size('NODES')>2]).
ctr_exchangeable(balance_tree, [items('NODES',all)]).
ctr_graph( balance_tree, ['NODES'],
```
2,
['CLIQUE'>>collection(nodes1,nodes2)],
[nodes1\^\text{succ}=nodes2\^\text{index}],
['MAX_NSNC'=<1,'RANGE_NCC'='BALANCE'], [[]).

\texttt{ctr\_eval(balance\_tree,[checker(balance\_tree\_c)]).}

\texttt{ctr\_functional\_dependency(balance\_tree,1,[2]).}

\texttt{balance\_tree\_c(BALANCE,NODES) :-
  length(NODES,N),
  N2 is max(N-2,0),
  check\_type(dvar(0,N2),BALANCE),
  collection(NODES,[\text{int}(1,N),dvar(1,N)]),
  get\_attr1(NODES,INDEXES),
  get\_attr2(NODES,SUCCS),
  true.}

\texttt{balance\_tree\_c(BALANCE,NODES) :-
  length(NODES,N),
  N2 is max(N-2,0),
  check\_type(dvar(0,N2),BALANCE),
  collection(NODES,[\text{int}(1,N),dvar(1,N)]),
  get\_attr1(NODES,INDEXES),
  get\_attr2(NODES,SUCCS),
  sort(INDEXES,SIND),
  length(SIND,N),
  length(RANKS,N),
  domain(RANKS,1,N),
  balance\_tree1(INDEXES,SUCCS,RANKS),
  (foreach(X,SUCCS),
   foreach(Free,Term),
   foreach(Free-1,KeyTerm),param(Term,N)do
    nth1(X,Term,Free)),
  keysort(KeyTerm,KeySorted),
  keyclumped(KeySorted,KeyClumped),
  (foreach(_38822-Ones,KeyClumped),
   foreach(Count,Counts)do
    length(Ones,Count)),
  min\_member(Min,Counts),
  max\_member(Max,Counts),
  BALANCE is Max-Min.

\texttt{balance\_tree1([],[],_38648) :- !.}
balance_tree1([I|RI],[I|RS],RANKS) :-
  !,
  balance_tree1(RI,RS,RANKS).

balance_tree1([I|RI],[S|RS],RANKS) :-
  nth1(I,RANKS,Ri),
  nth1(S,RANKS,Rs),
  Ri#<Rs,
  balance_tree1(RI,RS,RANKS).
**B.44 between_min_max**

◊ **Meta-Data:**

```prolog
ctr_date(between_min_max,['20050824','20060804']).
```

```prolog
ctr_origin(
  between_min_max,
  Used for defining %c.,
  [cumulative_convex]).
```

```prolog
ctr_arguments(
  between_min_max,
  ['VAR'-dvar,'VARIABLES'-collection(var-dvar)]).
```

```prolog
ctr_restrictions(
  between_min_max,
  [required('VARIABLES',var),size('VARIABLES')>0]).
```

```prolog
ctr_example(
  between_min_max,
  between_min_max(3,[[var-1],[var-1],[var-4],[var-8]])).
```

```prolog
ctr_typical(
  between_min_max,
  [size('VARIABLES')>1,range('VARIABLES' ^ var)>1]).
```

```prolog
ctr_exchangeable(
  between_min_max,
  [items('VARIABLES',all),
   vals(
     ['VAR'],
     int(['VAR','VARIABLES' ^ var]),
     =\=,
     all,
     dontcare)]).
```

```prolog
ctr_derived_collections(
  between_min_max,
  [col('ITEM'-collection(var-dvar),[item(var-'VAR')])]).
```

```prolog
ctr_graph(
  between_min_max,
  ['ITEM','VARIABLES'],
  2,
  ['PRODUCT'>>collection(item,variables)],
```
[item £ var>=variables £ var],
['NARC' £ =1],
['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_graph(
  between_min_max,
  ['ITEM','VARIABLES'],
  2,
  ['PRODUCT'>collection(item,variables)],
  [item £ var=<variables £ var],
  ['NARC' £ =1],
  ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(
  between_min_max,
  [reformulation(between_min_max_r),
   automaton(between_min_max_a)]).

ctr_extensible(between_min_max,[],'VARIABLES',any).

between_min_max_r(VAR,VARIABLES) :-
  check_type(dvar,VAR),
  collection(VARIABLES,[dvar]),
  length(VARIABLES,N),
  N>0,
  get_attr1(VARIABLES,VARS),
  get_minimum(VARS,MINIMUM),
  get_maximum(VARS,MAXIMUM),
  MIN in MINIMUM..MAXIMUM,
  MAX in MINIMUM..MAXIMUM,
  minimum(MIN,VARS),
  maximum(MAX,VARS),
  VAR#>=MIN,
  VAR#=<MAX.

between_min_max_a(FLAG,VAR,VARIABLES) :-
  check_type(dvar,VAR),
  collection(VARIABLES,[dvar]),
  length(VARIABLES,N),
  N>0,
  between_min_max_signature(VARIABLES,VAR,SIGNAL),
  AUTOMATON=
  automaton(
    SIGNAL,
    _34871,
    SIGNAL,
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

[\text{source}(s), \text{sink}(t)],
[\text{arc}(s,0,i),
 \text{arc}(s,1,t),
 \text{arc}(s,2,j),
 \text{arc}(i,0,i),
 \text{arc}(i,1,t),
 \text{arc}(i,2,t),
 \text{arc}(j,0,t),
 \text{arc}(j,1,t),
 \text{arc}(j,2,j),
 \text{arc}(t,0,t),
 \text{arc}(t,1,t),
 \text{arc}(t,2,t)],
[
[],
[
[],
[
]]],
\text{automaton\_bool}(\text{FLAG}, [0,1,2], \text{AUTOMATON}).

\text{between\_min\_max\_signature}([], _{32936}, []).

\text{between\_min\_max\_signature}([[\text{var-VARi}] | \text{VARs}], \text{VAR}, [S | Ss]) :-
\text{S in 0..2},
\text{VAR} < \text{VARi} \iff S = 0,
\text{VAR} = \text{VARi} \iff S = 1,
\text{VAR} > \text{VARi} \iff S = 2,
\text{between\_min\_max\_signature}(\text{VARs}, \text{VAR}, \text{Ss}).
B.45  bin_packing

◇ META-DATA:

ctr_date(
    bin_packing,
    ['20000128', '20030820', '20040530', '20060804']).

ctr_origin(bin_packing,'Derived from %c.',[cumulative]).

ctr_arguments(
    bin_packing,
    ['CAPACITY'-int,'ITEMS'-collection(bin-dvar,weight-int)]).

ctr_restrictions(
    bin_packing,
    ['CAPACITY'>=0,
     required('ITEMS',[bin,weight]),
     'ITEMS'\weight>=0,
     'ITEMS'\weight=<'CAPACITY']).

ctr_example(
    bin_packing,
    bin_packing(5,
        [[bin-3,weight-4],[bin-1,weight-3],[bin-3,weight-1]]).

ctr_typical(
    bin_packing,
    ['CAPACITY'>maxval('ITEMS'\weight),
     'CAPACITY'=<sum('ITEMS'\weight),
     size('ITEMS')>1,
     range('ITEMS'\bin)>1,
     range('ITEMS'\weight)>1,
     'ITEMS'\bin>=0,
     'ITEMS'\weight>0]).

ctr_exchangeable(
    bin_packing,
    [vals(['CAPACITY'],int,<,dontcare,dontcare),
     items('ITEMS',all),
     vals(['ITEMS'\weight],int(>=0)),>,dontcare,dontcare),
     vals(['ITEMS'\bin],int,=\,),all,dontcare)]).

ctr_graph(
    bin_packing,
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

[ITEM,ITEM],
2,
[PRODUCT>>collection(items1,items2)],
[items1^bin=items2^bin],
[],
[ACYCLIC,'BIPARTITE','NO_LOOP'],
[SUCC>>
[source,
variables-
   col('VARIABLES'-collection(var-dvar),
      [item(var-'ITEMS'^weight)]),
   [sum_ctr(variables,=<,'CAPACITY')].

ctr_eval(bin_packing,[reformulation(bin_packing_r)]).

ctr_contractible(bin_packing,[],ITEMS',any).

bin_packing_r(CAPACITY,ITEMS) :-
   integer(CAPACITY),
   CAPACITY>=0,
   collection(ITEMS,[dvar,int(0,CAPACITY)]),
   bin_packing1(ITEMS,1,TASKS),
   cumulative(TASKS,[limit(CAPACITY)]).
B.46  bin_packing_capa

◊ Meta-Data:

ctr_predefined(bin_packing_capa).

ctr_date(bin_packing_capa, ['20091404']).

ctr_origin(bin_packing_capa, 'Derived from %c.', [bin_packing]).

ctr_arguments(
  bin_packing_capa,
  ['BINS'-collection(id-int,capa-int),
   'ITEMS'-collection(bin-dvar,weight-int)].

ctr_restrictions(
  bin_packing_capa,
  [size('BINS')>0,
   required('BINS',[id,capa]),
   distinct('BINS',id),
   'BINS'\'id>=1,
   'BINS'\'id=<size('BINS'),
   'BINS'\'capa>=0,
   required('ITEMS', [bin,weight]),
   in_attr('ITEMS', bin, 'BINS', id),
   'ITEMS'\'weight>=0]).

ctr_example(
  bin_packing_capa,
  bin_packing_capa(
    [[id-1,capa-4],
     [id-2,capa-3],
     [id-3,capa-5],
     [id-4,capa-3],
     [id-5,capa-3],
     [[bin-3,weight-4],[bin-1,weight-3],[bin-3,weight-1]])).

ctr_typical(
  bin_packing_capa,
  [size('BINS')>1,
   range('BINS'\'capa)>1,
   'BINS'\'capa>maxval('ITEMS'\'weight),
   'BINS'\'capa=<sum('ITEMS'\'weight),
   size('ITEMS')>1,
   range('ITEMS'\'bin)>1,
   range('ITEMS'\'weight)>1,
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

'ITEMS' weight>0).

ctr_exchangeable(
    bin_packing_capa,
    [items('BINS', all),
     items('ITEMS', all),
     vals(['BINS' capa], int, <, dontcare, dontcare),
     vals(['ITEMS' weight], int (>= 0), >, dontcare, dontcare),
     vals(['BINS' id, 'ITEMS' bin], int, =\=, all, dontcare)]).

ctr_eval(bin_packing_capa, [reformulation(bin_packing_capa_r)]).

ctr_contractible(bin_packing_capa, [], 'ITEMS', any).

bin_packing_capa_r(BINS, ITEMS) :-
    length(BINS, N),
    collection(BINS, [int(1,N), int_gteq(0)]),
    collection(ITEMS, [dvar, int_gteq(0)]),
    get_attr1(BINS, IDS),
    get_attr2(BINS, CAPAS),
    get_maximum(CAPAS, MAX),
    MAX1 is MAX+1,
    all_different(IDS),
    bin_packing1(ITEMS, 1, TASKS),
    length(ITEMS, M),
    M1 is M+1,
    bin_packing1(BINS, M1, MAX, COMPLEMENTS),
    append(COMPLEMENTS, TASKS, COMPLEMENTS_TASKS),
    cumulative(COMPLEMENTS_TASKS, [limit(MAX1)]).

bin_packing_capa1([], 20256, 20257, []).

bin_packing_capa1(
    [[20267-I, 20274-W] | R],
    ID,
    MAX,
    [task(I,1,I1,H,ID) | S]) :-
    I1 is I+1,
    H is MAX-W+1,
    bin_packing_capa1(R, ID, MAX, S).
B.47 binary_tree

◊ Meta-Data:

ctr_date(binary_tree,['20000128','20030820','20060804']).

ctr_origin(binary_tree,'Derived from %c.',[tree]).

ctr_arguments(
    binary_tree,
    ['NTREES'-dvar,'NODES'-collection(index-int,succ-dvar)]).

ctr_restrictions(
    binary_tree,
    ['NTREES'>=0,
    'NTREES'=<size('NODES'),
    required('NODES',[index,succ]),
    'NODES'\index>=1,
    'NODES'\index=<size('NODES'),
    distinct('NODES',index),
    'NODES'\succ>=1,
    'NODES'\succ=<size('NODES')]).

ctr_example(
    binary_tree,
    binary_tree(2,
        [[index-1,succ-1],
        [index-2,succ-3],
        [index-3,succ-5],
        [index-4,succ-7],
        [index-5,succ-1],
        [index-6,succ-1],
        [index-7,succ-7],
        [index-8,succ-5]])).

ctr_typical(
    binary_tree,
    ['NTREES'>0,'NTREES'<size('NODES'),size('NODES')>2]).

ctr_exchangeable(binary_tree,[items('NODES',all)]).

ctr_graph(
    binary_tree,
    ['NODES'],
    2,
['CLIQUE'>>collection(nodes1,nodes2)],
[nodes1\textasciitilde succ=nodes2\textasciitilde index],
['MAX_NSNC'=<1,'NCC'='NTREES','MAX_ID'=<2],
['ONE_SUCC']).

ctr_eval(binary_tree,[reformulation(binary_tree_r)]).

ctr_functional_dependency(binary_tree,1,[2]).

\texttt{binary\_tree\_r(NTREES,NODES) :-
  eval(tree(NTREES,NODES)),
  get_attr1(NODES,INDEXES),
  get_attr2(NODES,SUCCS),
  k_ary_tree(INDEXES,INDEXES,SUCCS,2).}
B.48 bipartite

◊ META-DATA:

ctr_date(bipartite,['20061001']).

ctr_origin(bipartite,'\cite{Dooms06}',[]).

ctr_arguments(  
bipartite,  
['NODES'-collection(index-int,succ-svar)]).

ctr_restrictions(  
bipartite,  
[required('NODES',[index,succ]),
 'NODES'~index>=1,
 'NODES'~index=<size('NODES'),
 distinct('NODES',index),
 'NODES'~succ>=1,
 'NODES'~succ=<size('NODES'))].

ctr_example(  
bipartite,  
bipartite(  
[[index-1,succ-{2,3}],
 [index-2,succ-{1,4}],
 [index-3,succ-{1,4,5}],
 [index-4,succ-{2,3,6}],
 [index-5,succ-{3,6}],
 [index-6,succ-{4,5}]]).

ctr_typical(bipartite,[size('NODES')>2]).

ctr_exchangeable(bipartite,[items('NODES',all)]).

ctr_graph(  
bipartite,  
['NODES'],  
2,  
['CLIQUE'>>collection(nodes1,nodes2)],
 [nodes2~index in_set nodes1~succ],
 [],  
['SYMMETRIC','BIPARTITE']).
B.49 calendar

◊ **Meta-Data:**

```prolog
ctr_predefined(calendar).

ctr_date(calendar, ['20061014']).

ctr_origin(calendar, '\cite{BeldiceanuR00}', []).

ctr_types(
    calendar, 
    ['UNAVAILABILITIES'-collection(low-int,up-int)]).

ctr_arguments(
    calendar, 
    [INSTANTS-
        collection(
            machine-dvar, 
            virtual-dvar, 
            ireal-dvar, 
            flagend-int), 
        'MACHINES'-collection(id-int,cal='UNAVAILABILITIES'')].

ctr_restrictions(
    calendar, 
    [required('UNAVAILABILITIES',[low,up]), 
     'UNAVAILABILITIES'ˆlow=<'UNAVAILABILITIES'ˆup, 
     required('INSTANTS',[machine,virtual,ireal,flagend]), 
     in_attr('INSTANTS',machine,'MACHINES',id), 
     'INSTANTS'ˆflagend>=0, 
     'INSTANTS'ˆflagend=<1, 
     size('MACHINES')>0, 
     required('MACHINES',[id,cal]), 
     distinct('MACHINES',id)].

ctr_example(
    calendar, 
    calendar(
        [[machine-1,virtual-2,ireal-3,flagend-0], 
        [machine-1,virtual-5,ireal-6,flagend-1], 
        [machine-2,virtual-4,ireal-5,flagend-0], 
        [machine-2,virtual-6,ireal-9,flagend-1], 
        [machine-3,virtual-2,ireal-2,flagend-0], 
        [machine-3,virtual-5,ireal-5,flagend-1], 
        [machine-4,virtual-2,ireal-2,flagend-0], 
        [machine-4,virtual-6,ireal-9,flagend-1], 
        [machine-5,virtual-4,ireal-5,flagend-0], 
        [machine-5,virtual-6,ireal-9,flagend-1]]).
```
[machine-4, virtual-7, ireal-9, flagend-1],
[[id-1, cal-[[low-2, up-2], [low-6, up-7]]],
[id-2, cal-[[low-2, up-2], [low-6, up-7]]],
[id-3, cal-[]],
[id-4, cal-[[low-3, up-4]]]]).

ctr_typical(calendar, [size('INSTANTS') > 1, size('MACHINES') > 1]).

ctr_exchangeable(
calendar,
[items('INSTANTS', all), items('MACHINES', all)]).

ctr_eval(calendar, [reformulation(calendar_r)]).

ctr_contractible(calendar, [], 'INSTANTS', any).

calendar_r(INSTANTS, MACHINES) :-
collection(INSTANTS, [dvar, dvar, dvar, int(0, 1)]),
collection(MACHINES, [int, col([int, int])]),
length(MACHINES, M),
M > 0,
get_attr1(MACHINES, IDS),
all_different(IDS),
calendar.low_up(MACHINES),
( INSTANTS = [] ->
true
; calendar_in_attr(INSTANTS, IDS),
calendar_normalize(MACHINES, MACHINESN),
calendar_gen(INSTANTS, MACHINESN)
).

calendar_in_attr([], _24263).

calendar_in_attr([[_24272-M|_24270]|R], IDS) :-
build_or_var_in_values(IDS, M, TERM),
call(TERM),
calendar_in_attr(R, IDS).

calendar_low_up([]).

calendar_low_up([[_24268,_24273-CAL]|R]) :-
calendar_low_up1(CAL),
calendar_low_up(R).

calendar_low_up1([]).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

```
calendar_low_up1([[\_24271-L,\_24278-U]|R]) :-
    L=<U,
    calendar_low_up1(R).

calendar_normalize([],[]).

calendar_normalize([[[id-ID,cal-CAL]|R],
    [[[id-ID,cal-MERGED_CAL]|S]]) :-
    calendar_merge_intervals(CAL,MERGED_CAL),
    calendar_normalize(R,S).

calendar_merge_intervals(List,NewList) :-
    (foreach([low-L,up-U],List),fromto([],S1,S3,Set)do
        fdset_interval(S2,L,U),fdset_union(S1,S2,S3)),
    (foreach([A|B],Set),foreach([low-A,up-B],NewList)do
        true).

calendar_gen([],_24263).

calendar_gen([[[machine-M,virtual-V,ireal-R,flagend-F]|T],
    CALENDARS) :-
    calendar_gen(CALENDARS,M,V,R,F),
    calendar_gen(T,CALENDARS).

calendar_gen([],_24263,_24264,_24265,_24266).

calendar_gen([[[id-I,cal-C]|S],M,V,R,F) :-
    calendar_gen(C,1,0,I,M,V,R,F),
    calendar_gen(S,M,V,R,F).

calendar_gen([],1,0,I,M,V,R,_F) :-
    M#=I#<=>M#=I#/\R#==V.

calendar_gen([[[low-L,up-U]|S],1,0,I,M,V,R,F) :-
    LF is L+F,
    M#=I#/\R#<LF#<=M#=I#/\R#==V,
    calendar_gen([[[low-L,up-U]|S],0,0,I,M,V,R,F).

calendar_gen([[[low-K,up-U],[low-L,up-W]|S],0,Sum,I,M,V,R,F) :-
    NSum is Sum+U-K+1,
    KF is K+F,
    UF is U+F,
    LF is L+F,
    R in KF..UF#=>M\=I,
```


calendar_gen([[low-L, up-U]], 0, Sum, I, M, V, R, F) :-
  NSum is Sum+U-L+1,
  LF is L+F,
  UF is U+F,
  R in LF..UF --> M\#=I,
  M\#=I\#/R\#>UF\#<=>M\#=I\#/R\#=V+NSum.
B.50  cardinality_atleast

◊ **META-DATA:**

```prolog
ctr_date(  
    cardinality_atleast,  
    ['20030820','20040530','20060805']).

ctr_origin(  
    cardinality_atleast,  
    Derived from %c.,  
    [global_cardinality]).

ctr_arguments(  
    cardinality_atleast,  
    ['ATLEAST'-dvar,  
     'VARIABLES'-collection(var-dvar),  
     'VALUES'-collection(val-int)]).

ctr_restrictions(  
    cardinality_atleast,  
    ['ATLEAST'>=0,  
     'ATLEAST'=<size('VARIABLES'),  
     required('VARIABLES',var),  
     required('VALUES',val),  
     distinct('VALUES',val))].

ctr_example(  
    cardinality_atleast,  
    cardinality_atleast(  
        1,  
        [[var-3],[var-3],[var-8]],  
        [[val-3],[val-8]]]).

ctr_typical(  
    cardinality_atleast,  
    ['ATLEAST'>0,  
     'ATLEAST'<size('VARIABLES'),  
     size('VARIABLES')>1,  
     size('VALUES')>0,  
     size('VARIABLES')>size('VALUES')].

ctr_exchangeable(  
    cardinality_atleast,  
    [items('VARIABLES',all),  
     items('VALUES',all),  
     
```
vals(['VARIABLES'\textasciitilde var],
    all(notin('VALUES'\textasciitilde val)),
    =,
    dontcare,
    dontcare),
vals(['VARIABLES'\textasciitilde var,'VALUES'\textasciitilde val],
    int,
    =\equiv,
    all,
    dontcare)).

ctr\_graph(
    cardinality\_atleast,
    ['VARIABLES','VALUES'],
    2,
    ['PRODUCT'\textasciitilde>\textasciitilde>\textasciitilde collection(variables,values)],
    [variables\textasciitilde var=\equiv values\textasciitilde val],
    ['MAX_ID'=\textasciitilde size('VARIABLES')-'ATLEAST'],
    ['ACYCLIC','BIPARTITE','NO\_LOOP']).

ctr\_eval(
    cardinality\_atleast,
    [reformulation(cardinality\_atleast\_r)]).

ctr\_pure\_functional\_dependency(cardinality\_atleast,[]).

ctr\_functional\_dependency(cardinality\_atleast,1,[2,3]).

cardinality\_atleast\_r(ATLEAST,VARIABLES,VALUES) :-
    check\_type(dvar,ATLEAST),
    ATLEAST\#\geq=0,
    collection(VARIABLES,[dvar]),
    length(VARIABLES,N),
    ATLEAST\#\leq<N,
    ( VALUES=[] ->
        true
    ;
        collection(VALUES,[int]),
        length(VALUES,M),
        get\_attr1(VARIABLES,VARS),
        get\_attr1(VALUES,VALS),
        all\_different(VALS),
        length(NOCCS,M),
        fd\_min(ATLEAST,MIN\_ATLEAST),
        domain(NOCCS,MIN\_ATLEAST,N),
        ... }
get_minimum(VARS,MINVARS),
get_maximum(VARS,MAXVARS),
get_minimum(VALS,MINVALS),
get_maximum(VALS,MAXVALS),
MIN is \text{min}(MINVARS,MINVALS),
MAX is \text{max}(MAXVARS,MAXVALS),
\text{complete\_card}(MIN,MAX,N,VALS,NOCCS,VN),
\text{global\_cardinality}(VARS,VN)
}.
B.51 cardinality_atmost

◊ **META-DATA:**

\[
\text{ctr\_date}(\text{cardinality\_atmost}, [\text{\texttt{20030820}}, \text{\texttt{20040530}}, \text{\texttt{20060805}}]).
\]

\[
\text{ctr\_origin}(
    \text{cardinality\_atmost},
    \text{Derived from \%c.},
    [\text{global\_cardinality}]).
\]

\[
\text{ctr\_arguments}(
    \text{cardinality\_atmost},
    ['\texttt{ATMOST}'-dvar, 
    '\texttt{VARIABLES}'-collection(var-dvar),
    '\texttt{VALUES}'-collection(val-int)]).
\]

\[
\text{ctr\_restrictions}(
    \text{cardinality\_atmost},
    '\texttt{ATMOST}'\geq 0,
    '\texttt{ATMOST}'\leq \text{size('VARIABLES')},
    \text{required('VARIABLES', var)},
    \text{required('VALUES', val)},
    \text{distinct('VALUES', val)}).
\]

\[
\text{ctr\_example}(
    \text{cardinality\_atmost},
    \text{cardinality\_atmost}(2,
    [[\text{var-2}], [\text{var-1}], [\text{var-7}], [\text{var-1}], [\text{var-2}]],
    [[\text{val-5}], [\text{val-7}], [\text{val-2}], [\text{val-9}]]).
\]

\[
\text{ctr\_typical}(
    \text{cardinality\_atmost},
    '\texttt{ATMOST}'\geq 0,
    '\texttt{ATMOST}'\leq \text{size('VARIABLES')},
    \text{size('VARIABLES')}\geq 1,
    \text{size('VALUES')}\geq 0,
    \text{size('VARIABLES')}\leq \text{size('VALUES')}).
\]

\[
\text{ctr\_exchangeable}(
    \text{cardinality\_atmost},
    \text{items('VARIABLES', all)},
    \text{items('VALUES', all)},
    \text{vals}(
        ['\texttt{VARIABLES}'^\texttt{\textbackslash var}],
    )).
\]
all(notin('VALUES'ˆval)),
=,
dontcare,
dontcare),
vals(
   ['VARIABLES'ˆvar,'VALUES'ˆval],
   int,
   =\=,
   all,
   dontcare)).

ctr_graph(
   cardinality_atmost,
   ['VARIABLES','VALUES'],
   2,
   ['PRODUCT'>>collection(variables,values)],
   [variablesˆvar=valuesˆval],
   ['MAX_ID'='ATMOST'],
   ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(
   cardinality_atmost,
   [reformulation(cardinality_atmost_r)]).

ctr_pure_functional_dependency(cardinality_atmost,[]).

ctr_functional_dependency(cardinality_atmost,1,[2,3]).

cardinality_atmost_r(ATMOST,VARIABLES,VALUES) :-
   check_type(dvar,ATMOST),
   ATMOST#>=0,
   collection(VARIABLES,[dvar]),
   length(VARIABLES,N),
   ATMOST#=<N,
   ( VALUES=[] ->
     true
   ;
     collection(VARIABLES,[int]),
     length(VARIABLES,M),
     get_attr1(VARIABLES,VARS),
     get_attr1(VARIABLES,VALS),
     all_different(VALS),
     length(NOCCS,M),
     fd_max(ATMOST,MAX_ATMOST),
     domain(NOCCS,0,MAX_ATMOST),
     get_minimum(VARS,MINVARS),
     get_maximum(VARS,MAXVARS),
     true
   ).
get_minimum(VALS,MINVALS),
get_maximum(VALS,MAXVALS),
MIN is min(MINVARS,MINVALS),
MAX is max(MAXVARS,MAXVALS),
complete_card(MIN,MAX,N,VALS,NOCCS,VN),
global_cardinality(VARS,VN).
B.52 cardinality_atmost_partition

Meta-Data:

\texttt{ctr\_date(cardinality\_atmost\_partition,\text{[}'20030820','20060805'\text{]}).}

\texttt{ctr\_origin(}
\hspace{1em}\texttt{cardinality\_atmost\_partition,}
\hspace{1em}\texttt{Derived from \%c.,}
\hspace{1em}\texttt{[global\_cardinality]).}

\texttt{ctr\_types(}
\hspace{1em}\texttt{cardinality\_atmost\_partition,}
\hspace{1em}\texttt{[\textquoteleft VALUES\textquoteright\text{-collection(val-int)])}.}

\texttt{ctr\_arguments(}
\hspace{1em}\texttt{cardinality\_atmost\_partition,}
\hspace{1em}\texttt{[\textquoteright ATMOST\textquoteright\text{-dvar,}}
\hspace{1em}\texttt{\textquoteright VARIABLES\textquoteright\text{-collection(var-dvar),}}
\hspace{1em}\texttt{\textquoteright PARTITIONS\textquoteright\text{-collection(p\text{-VALUES})])}.}

\texttt{ctr\_restrictions(}
\hspace{1em}\texttt{cardinality\_atmost\_partition,}
\hspace{1em}\texttt{[size('VALUES')\geq 1,}
\hspace{1em}\texttt{required('VALUES',val),}
\hspace{1em}\texttt{distinct('VALUES',val),}
\hspace{1em}\texttt{\textquoteright ATMOST\textquoteright\geq 0,}
\hspace{1em}\texttt{\textquoteright ATMOST\textquoteright\leq size('VARIABLES'),}
\hspace{1em}\texttt{required('VARIABLES',var),}
\hspace{1em}\texttt{required('PARTITIONS',p),}
\hspace{1em}\texttt{size('PARTITIONS')\geq 2]).}

\texttt{ctr\_example(}
\hspace{1em}\texttt{cardinality\_atmost\_partition,}
\hspace{1em}\texttt{cardinality\_atmost\_partition(}
\hspace{2em}2,
\hspace{2em}[[var-2],[var-3],[var-7],[var-1],[var-6],[var-0]],
\hspace{2em}[[p-[[val-1],[val-3]]],
\hspace{3em}p-[[val-4]]],
\hspace{3em}p-[[val-2],[val-6]]])}.}

\texttt{ctr\_typical(}
\hspace{1em}\texttt{cardinality\_atmost\_partition,}
\hspace{1em}\texttt{\textquoteright ATMOST\textquoteright\geq 0,}
\hspace{1em}\texttt{\textquoteright ATMOST\textquoteright\leq size('VARIABLES'),}
\hspace{1em}\texttt{size('VARIABLES')\geq 1,}
size('VARIABLES') > size('PARTITIONS')).

ctr_exchangeable(
    cardinality_atmost_partition,
    [items('VARIABLES', all),
     items('PARTITIONS', all),
     items('PARTITIONS' ^ p, all)]).

ctr_graph(
    cardinality_atmost_partition,
    ['VARIABLES', 'PARTITIONS'],
    2,
    ['PRODUCT' >> collection(variables, partitions)],
    [variables ^ var in partitions ^ p],
    ['MAX_ID' = 'ATMOST'],
    ['ACYCLIC', 'BIPARTITE', 'NO_LOOP']).

ctr_eval(
    cardinality_atmost_partition,
    [reformulation(cardinality_atmost_partition_r)]).

ctr_pure_functional_dependency(cardinality_atmost_partition, []).

ctr_functional_dependency(cardinality_atmost_partition, 1, [2, 3]).

cardinality_atmost_partition_r(ATMOST,VARIABLES,PARTITIONS) :-
    collection(VARIABLES, [dvar]),
    length(VARIABLES, N),
    check_type(dvar(0, N), ATMOST),
    collection(PARTITIONS, [col_len_gteq(1, [int])]),
    length(PARTITIONS, P),
    P > 1,
    get_attr1(VARIABLES, VARS),
    get_col_attr1(PARTITIONS, 1, PVALS),
    flattern(PVALS, VALS),
    all_different(VALS),
    length(PVALS, LPVALS),
    LPVALS1 is LPVALS + 1,
    get_partition_var(VARS, PVALS, PVARS, LPVALS1, 0),
    complete_card_consec(1, LPVALS1, ATMOST, N, VALUES),
    global_cardinality(PVARS, VALUES).
B.53 change

◊ **Meta-Data:**

```prolog
ctr_date(change, ['20000128', '20030820', '20040530', '20060805']).

ctr_origin(change, 'CHIP\index{CHIP|indexuse}|CHIP', []).

ctr_synonyms(change, [nbchanges, similarity]).

ctr_arguments(change, ['NCHANGE'-dvar, 'VARIABLES'-collection(var-dvar), 'CTR'-atom]).

ctr_restrictions(change, ['NCHANGE'>=0, 'NCHANGE'<size('VARIABLES'), required('VARIABLES', var), in_list('CTR', [=,\=,<,>=,>,=<]))].

ctr_example(change, [change(3, [[var-4], [var-4], [var-3], [var-4], [var-1]], \=), change(1, [[var-1], [var-2], [var-4], [var-3], [var-7]], >)].

ctr_typical(change, ['NCHANGE'>0, size('VARIABLES')>1, range('VARIABLES'\^var)>1, in_list('CTR', [\=])].

ctr_exchangeable(change, [translate(['VARIABLES'\^var])].

ctr_graph(change, ['VARIABLES'], 2, ['PATH'>collection(variables1, variables2)], ['CTR'(variables1\^var, variables2\^var), 'NARC'='NCHANGE'], ['ACYCLIC', 'BIPARTITE', 'NO_LOOP']).
```
ctr_eval(change, [automaton(change_a)]).

ctr_pure_functional_dependency(change, []).

ctr_functional_dependency(change, 1, [2, 3]).

ctr_contractible(change,
    [in_list('CTR', [=\=, <, >, >\=, >=, =<]), 'NCHANGE'=0],
    VARIABLES,
    any).

ctr_contractible(change,
    [in_list('CTR', [=\=, <, >, >\=, >=, =<]), 'NCHANGE'=size('VARIABLES')-1],
    VARIABLES,
    any).

change_a(FLAG, NCHANGE, VARIABLES, CTR) :-
    collection(VARIABLES, [dvar]),
    length(VARIABLES, N),
    N_1 is N-1,
    check_type(dvar(0, N_1), NCHANGE),
    memberchk(CTR, [=\=, <, >\=, >=, =<]),
    change_signature(VARIABLES, SIGNATURE, CTR),
    automaton(
        SIGNATURE, _41957,
        SIGNATURE,
        [source(s), sink(s)],
        [arc(s, 0, s), arc(s, 1, s, [C+1])],
        [C],
        [0],
        [COUNT]),
    NCHANGE#=COUNT#=FLAG.

change_signature([], [], _40279).

change_signature([], [], _40282) :- !.

change_signature([_40283], [], _40282) :- !.
change_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss],\(=\)) :-
!,
VAR1\(\neq\)VAR2\(\leq\)S,
change_signature([[var-VAR2]|VARs],Ss,\(=\)).

change_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss],\(<\)) :-
!,
VAR1\(\leq\)VAR2\(\leq\)S,
change_signature([[var-VAR2]|VARs],Ss,<).

change_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss],\(\geq\)) :-
!,
VAR1\(\geq\)VAR2\(\leq\)S,
change_signature([[var-VAR2]|VARs],Ss,\(\geq\)).

change_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss],\(>\)) :-
!,
VAR1\(>\)VAR2\(\leq\)S,
change_signature([[var-VAR2]|VARs],Ss,\(>\)).

change_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss],\(\leq\)) :-
!,
VAR1\(\leq\)VAR2\(\leq\)S,
change_signature([[var-VAR2]|VARs],Ss,\(\leq\)).
B.54  change_continuity

◊ META-DATA:

```prolog
ctr_date(
    change_continuity,
    ['20000128','20030820','20040530','20060805']).
```

```prolog
ctr_origin(change_continuity,'N.Beldiceanu',[]).
```

```prolog
ctr_arguments(
    change_continuity,
    ['NB_PERIOD_CHANGE'-dvar,  
     'NB_PERIOD_CONTINUITY'-dvar,  
     'MIN_SIZE_CHANGE'-dvar,        
     'MAX_SIZE_CHANGE'-dvar,        
     'MIN_SIZE_CONTINUITY'-dvar,    
     'MAX_SIZE_CONTINUITY'-dvar,    
     'NB_CHANGE'-dvar,              
     'NB_CONTINUITY'-dvar,          
     'VARIABLES'-collection(var-dvar),  
     'CTR'-atom]).
```

```prolog
ctr_restrictions(
    change_continuity,
    ['NB_PERIOD_CHANGE'>=0,     
     'NB_PERIOD_CONTINUITY'>=0,  
     'MIN_SIZE_CHANGE'>=0,       
     'MAX_SIZE_CHANGE'>='MIN_SIZE_CHANGE',  
     'MIN_SIZE_CONTINUITY'>=0,    
     'MAX_SIZE_CONTINUITY'>='MIN_SIZE_CONTINUITY',  
     'NB_CHANGE'>=0,              
     'NB_CONTINUITY'>=0,          
     required('VARIABLES',var),   
     in_list('CTR',[=,\=,<,>=,>,=<])).
```

```prolog
ctr_example(
    change_continuity,
    change_continuity(3, 
                      2, 
                      2, 
                      4, 
                      2, 
                      4, 
                      6,
```
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

\[
\begin{align*}
4, \\
&[[\text{var}-1], \\
&[\text{var}-3], \\
&[\text{var}-1], \\
&[\text{var}-8], \\
&[\text{var}-8], \\
&[\text{var}-4], \\
&[\text{var}-7], \\
&[\text{var}-7], \\
&[\text{var}-7], \\
&[\text{var}-4], \\
&[\text{var}-2]], \\
&(+\equiv)).
\end{align*}
\]

\text{ctr\_typical}(
  \text{change\_continuity},
  ['\text{NB\_PERIOD\_CHANGE}'>0,
   '\text{NB\_PERIOD\_CONTINUITY}'>0,
   '\text{MIN\_SIZE\_CHANGE}'>0,
   '\text{MIN\_SIZE\_CONTINUITY}'>0,
   '\text{NB\_CHANGE}'>0,
   '\text{NB\_CONTINUITY}'>0,
   \text{size('VARIABLES')}>1,
   \text{range('VARIABLES'\text{\_var})}>1,
   \text{in\_list('CTR',[\equiv\equiv])}).
\)

\text{ctr\_exchangeable}(
  \text{change\_continuity},
  \text{translate}(['\text{VARIABLES'}\text{\_var}])).
\)

\text{ctr\_graph}(
  \text{change\_continuity},
  ['\text{VARIABLES}'],
  2,
  ['\text{PATH}'>\text{collection(variables1,variables2)}],
  ['\text{CTR}\text{\_variables1}\text{\_var,variables2}\text{\_var}'],
  ['\text{NCC}'='\text{NB\_PERIOD\_CHANGE}',
   '\text{MIN\_NCC}'='\text{MIN\_SIZE\_CHANGE}',
   '\text{MAX\_NCC}'='\text{MAX\_SIZE\_CHANGE}',
   '\text{NARC}'='\text{NB\_CHANGE}'],
  ['\text{ACYCLIC}',\text{\_BIPARTITE}',\text{\_NO\_LOOP}]).
\)

\text{ctr\_graph}(
  \text{change\_continuity},
  ['\text{VARIABLES}'],
  2,
['PATH'>>collection(variables1,variables2)],
[#'CTR'(variables1^var,variables2^var)],
['NCC'='NB_PERIOD_CONTINUITY',
'MIN_NCC'='MIN_SIZE_CONTINUITY',
'MAX_NCC'='MAX_SIZE_CONTINUITY',
'NARC'='NB_CONTINUITY'],
['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(change_continuity,[automata(change_continuity_a)]).

ctr_functional_dependency(change_continuity,1,[9,10]).

ctr_functional_dependency(change_continuity,2,[9,10]).

ctr_functional_dependency(change_continuity,3,[9,10]).

ctr_functional_dependency(change_continuity,4,[9,10]).

ctr_functional_dependency(change_continuity,5,[9,10]).

ctr_functional_dependency(change_continuity,6,[9,10]).

ctr_functional_dependency(change_continuity,7,[9,10]).

ctr_functional_dependency(change_continuity,8,[9,10]).

change_continuity_a(
    NB_PERIOD_CHANGE,
    NB_PERIOD_CONTINUITY,
    MIN_SIZE_CHANGE,
    MAX_SIZE_CHANGE,
    MIN_SIZE_CONTINUITY,
    MAX_SIZE_CONTINUITY,
    NB_CHANGE,
    NB_CONTINUITY,
    VARIABLES,
    CTR) :-
    check_type(dvar,NB_PERIOD_CHANGE),
    check_type(dvar,NB_PERIOD_CONTINUITY),
    check_type(dvar,MIN_SIZE_CHANGE),
    check_type(dvar,MAX_SIZE_CHANGE),
    check_type(dvar,MIN_SIZE_CONTINUITY),
    check_type(dvar,MAX_SIZE_CONTINUITY),
    check_type(dvar,NB_CHANGE),
    check_type(dvar,NB_CONTINUITY),
    collection(VARIABLES,[dvar]),
    ...
memberchk(CTR, [=, =!, <, =>, >, =<]),
NB_PERIODCHANGE#=0,
NB_PERIODCONTINUITY#=0,
MIN_SIZE_CHANGE#=0,
MAX_SIZE_CHANGE#=MIN_SIZE_CHANGE,
MIN_SIZE_CONTINUITY#=0,
MAX_SIZE_CONTINUITY#=MIN_SIZE_CONTINUITY,
NB_CHANGE#=0,
NB_CONTINUITY#=0,
change_continuity_signature(  
  VARIABLES,  
  SIGNATURE_CTR,  
  1,  
  CTR),
change_continuity_signature(  
  VARIABLES,  
  SIGNATURE_NOT_CTR,  
  0,  
  CTR),
change_continuity_nb_period(  
  NB_PERIODCHANGE,  
  SIGNATURE_CTR),
change_continuity_nb_period(  
  NB_PERIODCONTINUITY,  
  SIGNATURE_NOT_CTR),
change_continuity_min_size(  
  MIN_SIZECHANGE,  
  SIGNATURE_CTR),
change_continuity_min_size(  
  MIN_SIZECONTINUITY,  
  SIGNATURE_NOT_CTR),
change_continuity_max_size(  
  MAX_SIZECHANGE,  
  SIGNATURE_CTR),
change_continuity_max_size(  
  MAX_SIZECONTINUITY,  
  SIGNATURE_NOT_CTR),
change_continuity_nb(NB_CHANGE, SIGNATURE_CTR),
change_continuity_nb(NB_CONTINUITY, SIGNATURE_NOT_CTR).

change_continuity_nb_period(NB_PERIOD, SIGNATURE) :-
  automaton(  
    SIGNATURE,  
    _52543,  
    SIGNATURE,  
    [source(s), sink(s), sink(i)],
change_continuity_min_size(MIN_SIZE,SIGNATURE) :-
MIN_SIZE#=min(C1,D1),
automaton(SIGNATURE,
_source(s),sink(i),sink(j),sink(k),sink(s),
[arc(s,0,s),
 arc(s,1,i,[C+1]),
arci,1,i),
arci,0,s],
[C],
[0],
[NB_PERIOD]).

change_continuity_max_size(MAX_SIZE,SIGNATURE) :-
MAX_SIZE#=max(C1,D1),
automaton(SIGNATURE,
_source(s),sink(i),sink(s),
[arc(s,0,s,[C,D]),
arcs,1,i,[C,D+1]),
arci,0,i,[max(C,D),1]),
arci,1,i,[C,D+1]),
[C,D],
[0,1],
[C1,D1]).

change_continuity_nb(NB,SIGNATURE) :-
automaton(SIGNATURE,
change_continuity_signature([],[],_51904,_51905).

change_continuity_signature([_51909],[],_51907,_51908) :- !.

change_continuity_signature(
  [[var-VAR1],[var-VAR2]|VARs],
  [S|Ss],
  1,
  =) :- !,
      VAR1#=VAR2#<=>S,
      change_continuity_signature([var-VAR2]|VARs),Ss,1,=).

change_continuity_signature(
  [[var-VAR1],[var-VAR2]|VARs],
  [S|Ss],
  1,
  \=) :- !,
      VAR1\!=VAR2\!<=>S,
      change_continuity_signature([var-VAR2]|VARs),Ss,1,\=).

change_continuity_signature(
  [[var-VAR1],[var-VAR2]|VARs],
  [S|Ss],
  1,
  <) :- !,
      VAR1<VAR2<=>S,
      change_continuity_signature([var-VAR2]|VARs),Ss,1,<).

change_continuity_signature(
  [[var-VAR1],[var-VAR2]|VARs],
  [S|Ss],
  1,
  >=) :- !,
VAR1#>=VAR2#<=>S,  
change_continuity_signature([[var-VAR2]|VARs],Ss,1,>=).

change_continuity_signature(  
[[var-VAR1],[var-VAR2]|VARs],  
[S|Ss],  
1,  
>) :-  
!,  
VAR1#VAR2#<=>S,  
change_continuity_signature([[var-VAR2]|VARs],Ss,1,>).

change_continuity_signature(  
[[var-VAR1],[var-VAR2]|VARs],  
[S|Ss],  
1,  
=<) :-  
!,  
VAR1#VAR2#<=>S,  
change_continuity_signature([[var-VAR2]|VARs],Ss,1,=<).

change_continuity_signature(  
[[var-VAR1],[var-VAR2]|VARs],  
[S|Ss],  
0,  
=) :-  
!,  
VAR1#=VAR2#<=>S,  
change_continuity_signature([[var-VAR2]|VARs],Ss,0,=).

change_continuity_signature(  
[[var-VAR1],[var-VAR2]|VARs],  
[S|Ss],  
0,  
=\=) :-  
!,  
VAR1#=VAR2#<=>S,  
change_continuity_signature([[var-VAR2]|VARs],Ss,0,=\=).

change_continuity_signature(  
[[var-VAR1],[var-VAR2]|VARs],  
[S|Ss],  
0,  
<) :-  
!,  
VAR1#<=VAR2#<=>S,
change_continuity_signature([[var-VAR2]|VARs],Ss,0,<).

change_continuity_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    0,
    >=) :-
    !,
    VAR1#<VAR2#<=>S,
    change_continuity_signature([[var-VAR2]|VARs],Ss,0,>=).

change_continuity_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    0,
    >) :-
    !,
    VAR1#=<VAR2#<=>S,
    change_continuity_signature([[var-VAR2]|VARs],Ss,0,>).

change_continuity_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    0,
    <=) :-
    !,
    VAR1#>VAR2#<=>S,
    change_continuity_signature([[var-VAR2]|VARs],Ss,0,=<=).
B.55 change_pair

◊ **META-DATA:**

```prolog
ctr_date(change_pair,['20030820','20040530','20060805']).

ctr_origin(change_pair,'Derived from %c.',[change]).

ctr_arguments(change_pair,
              ['NCHANGE'-dvar,
               'PAIRS'-collection(x-dvar,y-dvar),
               'CTRX'-atom,
               'CTRY'-atom]).

ctr_restrictions(change_pair,
                  ['NCHANGE'>=0,
                   'NCHANGE'<size('PAIRS'),
                   required('PAIRS',[x,y]),
                   in_list('CTRX',[=,\=,<,\>=,>,\=<]),
                   in_list('CTRY',[=,\=,<,\>=,>,\=<])].

ctr_example(change_pair,
            change_pair(3,
                         [[x-3,y-5],
                          [x-3,y-7],
                          [x-3,y-7],
                          [x-3,y-8],
                          [x-3,y-4],
                          [x-3,y-7],
                          [x-1,y-3],
                          [x-1,y-6],
                          [x-1,y-6],
                          [x-3,y-7]],
                         =\=,<\>=>,\<=)).

ctr_typical(change_pair,
            ['NCHANGE'>0,
             size('PAIRS')>1,
             range('PAIRS'\^x)>1,
             range('PAIRS'\^y)>1]).
```
ctr_exchangeable(
    change_pair,
    ['PAIRS'\^x], 'PAIRS'\^y]))

ctr_graph(
    change_pair,
    ['PAIRS'],
    2,
    ['PATH'->collection(pairs1,pairs2)],
    ['CTRX' (pairs1\^x,pairs2\^x)\+'/'+CTRX' (pairs1\^y,pairs2\^y)],
    ['NARC'='NCHANGE'],
    ['ACYCLIC','BIPARTITE','NO_LOOP'])

ctr_eval(change_pair,[automaton(change_pair_a)])

ctr_pure_functional_dependency(change_pair,[])

ctr_functional_dependency(change_pair,1,[2,3,4])

change_pair_a(FLAG,NCHANGE,PAIRS,CTRX,CTRY) :-
    collection(PAIRS,[dvar,dvar]),
    length(PAIRS,N),
    N_1 is N-1,
    check_type(dvar(0,N_1),NCHANGE),
    memberchk(CTRX,[=,\<,\>=,>,\=<]),
    memberchk(CTRY,[=,\<,\>=,>,\=<]),
    change_pair_signature(PAIRS,SIGNATURE,CTRX,CTRY),
    automaton(
        SIGNATURE,
        _37471,
        SIGNATURE,
        [source(s),sink(s)],
        [arc(s,0,s),arc(s,1,s,[C+1])],
        [C],
        [0],
        [COUNT]),
    COUNT#=NCHANGE#<=>FLAG.

change_pair_signature([],[],_35598,_35599).

change_pair_signature([-35603],[],_35601,-35602) :- !.

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    _35606,-35607,-35608,-35609) :- !.

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    _35610,-35611,-35612,-35613) :- !.

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    _35614,-35615,-35616,-35617) :- !.

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    _35618,-35619,-35620,-35621) :- !.

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    _35622,-35623,-35624,-35625) :- !.

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    _35626,-35627,-35628,-35629) :- !.

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    _35631,-35632,-35633,-35634) :- !.

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    _35636,-35637,-35638,-35639) :- !.

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    _35642,-35643,-35644,-35645) :- !.
change_pair_signature([x-X1,y-Y1],[x-X2,y-Y2]|PAIRs), [S|Ss], =, =) :- !,
        X1#=X2#
        \Y1#\Y2#<=>S,
        change_pair_signature([x-X2,y-Y2]|PAIRs], Ss, =, =).

change_pair_signature([x-X1,y-Y1],[x-X2,y-Y2]|PAIRs], [S|Ss], =, =\=) :- !,
        X1#=X2#
        \Y1#\Y2#<=>S,
        change_pair_signature([x-X2,y-Y2]|PAIRs], Ss, =, =\=).

change_pair_signature([x-X1,y-Y1],[x-X2,y-Y2]|PAIRs], [S|Ss], =, <) :- !,
        X1#=X2#
        \Y1#\Y2#<=>S,
        change_pair_signature([x-X2,y-Y2]|PAIRs], Ss, =, <).

change_pair_signature([x-X1,y-Y1],[x-X2,y-Y2]|PAIRs], [S|Ss], =, >=) :- !,
        X1#=X2#
        \Y1#\Y2#<=>S,
        change_pair_signature([x-X2,y-Y2]|PAIRs], Ss, =, >=).

change_pair_signature([x-X1,y-Y1],[x-X2,y-Y2]|PAIRs], [S|Ss], =, >) :- !,
        X1#=X2#
        \Y1#\Y2#<=>S,
        change_pair_signature([x-X2,y-Y2]|PAIRs], Ss, =, >).
change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =\=,
    
    =) :-
    !,
    X1\=X2\=/Y1\=Y2\=\=S,
    change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,\=\=,\=\=).
}

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =\=,
    <) :-
    !,
    X1\=X2\=/Y1\=<Y2\=\=S,
    change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,\=\=,\=\=).
}

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =\=,
    >) :-
    !,
    X1\=X2\=/Y1\>=Y2\=\=S,
    change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,\=\=,\=\=).
}

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =\=,

    <=) :-
    !,
    X1\=X2\/=Y1\=<Y2\=\=S,
    change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,\=\=,\=\=).
}

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =\=,
    >=) :-
    !,
    X1\=X2\/=Y1\>=Y2\=\=S,
    change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,\=\=,\=\=).
}
change_pair_signature([x-X1,y-Y1],[x-X2,y-Y2]|PAIRs),[S|Ss],\=\=,>) :-
!
X1\=X2\/Y1\>Y2\<=>S,
change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,\=\=,>).

change_pair_signature([x-X1,y-Y1],[x-X2,y-Y2]|PAIRs),
[S|Ss],=<,=) :-
!
X1\=X2\/Y1\=<Y2\<=>S,
change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=\=,=).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

%!X1#<X2#/Y1#/Y2#<=S,
change_pair_signature([[[x-X1,y-Y1],x-X2,y-Y2]|PAIRs],SS,<,>=).

change_pair_signature([[[x-X1,y-Y1],x-X2,y-Y2]|PAIRs],SS,<,>) :-
%!X1#<X2#/Y1#/Y2#<=S,
change_pair_signature([[[x-X1,y-Y1],x-X2,y-Y2]|PAIRs],SS,<,<=).

change_pair_signature([[[x-X1,y-Y1],x-X2,y-Y2]|PAIRs],SS,<=,<=) :-
%!X1#<X2#/Y1#/Y2#<=S,
change_pair_signature([[[x-X1,y-Y1],x-X2,y-Y2]|PAIRs],SS,<=,=).

change_pair_signature([[[x-X1,y-Y1],x-X2,y-Y2]|PAIRs],SS,>=,=) :-
%!X1#>=X2#/Y1#/Y2#<=S,
change_pair_signature([[[x-X1,y-Y1],x-X2,y-Y2]|PAIRs],SS,>=,\=).

change_pair_signature([[[x-X1,y-Y1],x-X2,y-Y2]|PAIRs],SS,>=,\=) :-
%!X1#>=X2#/Y1#/Y2#<=S,
change_pair_signature([[[x-X1,y-Y1],x-X2,y-Y2]|PAIRs],SS,>=,\=).

change_pair_signature([[[x-X1,y-Y1],x-X2,y-Y2]|PAIRs],SS,>=,\=) :-
%!X1#>=X2#/Y1#/Y2#<=S,
change_pair_signature([[[x-X1,y-Y1],x-X2,y-Y2]|PAIRs],SS,>=,\=).
\[X_1 \geq X_2 \lor Y_1 < Y_2 \iff S,\]
\[
\text{change_pair_signature}(\[[x-X1, y-Y1], [x-X2, y-Y2] | \text{PAIRs}\], S, \geq, <).
\]

\[
\text{change_pair_signature}(\[[x-X1, y-Y1], [x-X2, y-Y2] | \text{PAIRs}\], S, \geq, \geq)
\]
\[
\text{change_pair_signature}(\[[x-X1, y-Y1], [x-X2, y-Y2] | \text{PAIRs}\], S, \geq, >).
\]

\[
\text{change_pair_signature}(\[[x-X1, y-Y1], [x-X2, y-Y2] | \text{PAIRs}\], S, >, <).
\]

\[
\text{change_pair_signature}(\[[x-X1, y-Y1], [x-X2, y-Y2] | \text{PAIRs}\], S, >, \geq).
\]

\[
\text{change_pair_signature}(\[[x-X1, y-Y1], [x-X2, y-Y2] | \text{PAIRs}\], S, \geq, \leq).
\]

\[
\text{change_pair_signature}(\[[x-X1, y-Y1], [x-X2, y-Y2] | \text{PAIRs}\], S, >, \leq).
\]

\[
\text{change_pair_signature}(\[[x-X1, y-Y1], [x-X2, y-Y2] | \text{PAIRs}\], S, >, =).
\]
change_pair_signature([[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],\[S|Ss\],>) :- !,
X1#>X2#\/_Y1#<Y2#<=S,
change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,>)

change_pair_signature([[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],\[S|Ss\],)>=) :- !,
X1#>X2#\/_Y1#>=Y2#<=S,
change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,)>=)

change_pair_signature([[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],\[S|Ss\],>) :- !,
X1#>X2#\/_Y1#=Y2#<=S,
change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,>)

change_pair_signature([[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],\[S|Ss\],)=<) :- !,
X1#>X2#\/_Y1#=<Y2#<=S,
change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,)=<)

change_pair_signature([[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],\[S|Ss\],)=) :- !,
X1#=X2#\/_Y1#=Y2#<=S,
change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,)=)
change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =<,
    =\=) :-
    !,
    X1#$\leq$X2#$\lor$ Y1#$\leq$ Y2#$\Rightarrow$ S,
    change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=,<,\=).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =<,
    <) :-
    !,
    X1#$\leq$X2#$\lor$ Y1#$<Y2#$\Rightarrow$ S,
    change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=,<,<).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =<,
    >=) :-
    !,
    X1#$\leq$X2#$\lor$ Y1#$\geq$ Y2#$\Rightarrow$ S,
    change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=,<,\>=).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =<,
    >=) :-
    !,
    X1#$\leq$X2#$\lor$ Y1#$\geq$ Y2#$\Rightarrow$ S,
    change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=,<,\>=).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =<,
    <=) :-
    !,
    X1#$\leq$X2#$\lor$ Y1#$\leq$ Y2#$\Rightarrow$ S,
    change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=,<,\=).
B.56  change_partition

◊ Meta-Data:

ctr_date(
    change_partition,
    ['20000128','20030820','20040530','20060805']).

ctr_origin(change_partition,'Derived from %c.',[change]).

ctr_types(change_partition,['VALUES'-collection(val-int)]).

ctr_arguments(
    change_partition,
    ['NCHANGE'-dvar,
    'VARIABLES'-collection(var-dvar),
    'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    change_partition,
    [size('VALUES')>=1,
    required('VALUES',val),
    distinct('VALUES',val),
    'NCHANGE'>=0,
    'NCHANGE'<size('VARIABLES'),
    required('VARIABLES',var),
    required('PARTITIONS',p),
    size('PARTITIONS')>=2]).

ctr_example(
    change_partition,
    change_partition(2,
    [ [var-6],
    [var-6],
    [var-2],
    [var-1],
    [var-3],
    [var-3],
    [var-1],
    [var-1],
    [var-2],
    [var-2],
    [var-2]],
    [[p-[[val-1],[val-3]]],
    [p-[[val-4]]],
    [...])
(p-[[val-2],[val-6]])).

ctr_typical(
    change_partition,
    ['NCHANGE'>0,
     size('VARIABLES')>1,
     range('VARIABLES'\var)>1,
     size('VARIABLES')>size('PARTITIONS')]).

ctr_exchangeable(
    change_partition,
    [items('VARIABLES',reverse),
     items('PARTITIONS',all),
     items('PARTITIONS'\p,all),
     vals(
      ['VARIABLES'\var],
      part('PARTITIONS'),
      =,
      dontcare,
      dontcare)]).

ctr_graph(
    change_partition,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2)],
    [in_same_partition(
      variables1\var,
      variables2\var,
      PARTITIONS)],
    ['NARC='NCHANGE'],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_pure_functional_dependency(change_partition,[]).

ctr_functional_dependency(change_partition,1,[2,3]).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.57 change_vectors

◊ METADATA:

\[
\text{ctr\_date(change\_vectors,('20110616'))}.
\]

\[
\text{ctr\_origin(change\_vectors,'Derived from %c',[change])}.
\]

\[
\text{ctr\_types(change\_vectors, [['VECTOR'-collection(var-dvar),'CTR'-atom]).}
\]

\[
\text{ctr\_arguments(change\_vectors, ['NCHANGE'-dvar, 'VECTORS'-collection(vec-'VECTOR'), 'CTRS'-collection(ctr-'CTR'))].}
\]

\[
\text{ctr\_restrictions(change\_vectors, [size('VECTOR')>=1, required('VECTOR',var), in_list('CTR',[=,\=,<,\=>,\=<]), 'NCHANGE'>=0, 'NCHANGE'<size('VECTORS'), required('VECTORS',vec), same_size('VECTORS',vec), required('CTRS',ctr), size('CTRS')=size('VECTOR')).}
\]

\[
\text{ctr\_example(change\_vectors, change\_vectors(3, [[vec-[[var-4],[var-0]]], [vec-[[var-4],[var-0]]], [vec-[[var-4],[var-5]]], [vec-[[var-3],[var-4]]], [vec-[[var-3],[var-4]]], [vec-[[var-3],[var-4]]], [vec-[[var-4],[var-0]]], [[ctr- =\=],[ctr- =\=]])).}
\]

\[
\text{ctr\_typical(change\_vectors, [in_list('CTR',[=\=])].}
\]
size('VECTOR') > 1,
'NCHANGE' > 0,
size('VECTORS') > 1).

ctr_eval(change_vectors, [automaton(change_vectors_a)]).

ctr_pure_functional_dependency(change_vectors, []). 

ctr_functional_dependency(change_vectors, 1, [2, 3]).

change_vectors_a(FLAG, NCHANGE, VECTORS, CTRS) :-
collection(VECTORS, [col([dvar])]),
length(VECTORS, N),
N_1 is N-1,
check_type(dvar(0, N_1), NCHANGE),
collection(CTRS, [atom([=, =, =, =, <=, >, >, <=])]),
same_size(VECTORS),
length(CTRS, M),
VECTORS = [[VECTOR1]|VECTOR1],
length(VECTOR1, M),
M >= 1,
get_attr1(VECTORS, VECTS),
get_attr1(CTRS, LCTRS),
change_vectors_signature(VECTS, SIGNATURE, LCTRS),
AUTOMATON = automaton(
  SIGNATURE,
  _20326,
  SIGNATURE,
  [source(s), sink(s)],
  [arc(s, 0, s), arc(s, 1, s, [C+1])],
  [C],
  [0],
  [NCHANGE]),
automaton_bool(FLAG, [0, 1], AUTOMATON).

change_vectors_signature([], [], _17109) :- !.

change_vectors_signature([VECTOR1], [], _17109) :- !.

change_vectors_signature([VECTOR1, VECTOR2|VECs], [S|Ss], CTRS) :- !,
build_vectors_compare_change(VECTOR1, VECTOR2, CTRS, Term),
call(Term#<=>S),
change_vectors_signature([VEC2|VECs],Ss,CTRS).
B.58 circuit

◊ **META-DATA:**

```prolog
ctr_date(circuit, ['20030820', '20040530', '20060805']).
ctr_origin(circuit, '\cite{Lauriere78}', []).
ctr_synonyms(circuit, [atour, cycle]).
ctr_arguments(
    circuit,
    ['NODES'-collection(index-int, succ-dvar)]).
ctr_restrictions(
    circuit,
    [required('NODES', [index, succ]),
     'NODES'\index>=1,
     'NODES'\index=<size('NODES'),
     distinct('NODES', index),
     'NODES'\succ>=1,
     'NODES'\succ=<size('NODES')]).
ctr_example(
    circuit,
    circuit(
        [[index-1, succ-2],
         [index-2, succ-3],
         [index-3, succ-4],
         [index-4, succ-1]])).
ctr_typical(circuit, [size('NODES')>2]).
ctr_exchangeable(circuit, [items('NODES', all)]).
ctr_graph(
    circuit,
    ['NODES'],
    2,
    ['CLIQUE'\collection(nodes1, nodes2)],
    [nodes1\succ=nodes2\index],
    ['MIN_NSCC'=size('NODES'), 'MAX_ID'=<1],
    ['ONE_SUCC']).
ctr_eval(circuit, [builtin(circuit_b)]).
```
circuit_b(NODES) :-
    length(NODES,N),
    collection(NODES,[int(1,N),dvar(1,N)]),
    get_attr1(NODES,INDEX),
    all_different(INDEX),
    get_attr2(NODES,SUCC),
    circuit(SUCC).
B.59 circuit_cluster

◊ **META-DATA:**

ctr_date(circuit_cluster,['20000128', '20030820', '20060805']).

ctr_origin(circuit_cluster,
Inspired by \cite{LaporteAsefVaziriSriskandarajah96}.).

ctr_arguments(circuit_cluster,
['NCIRCUIT'-dvar,
'NODES'-collection(index-int,cluster-int,succ-dvar)]).

ctr_restrictions(circuit_cluster,
['NCIRCUIT'>=1,
'NCIRCUIT'=<size('NODES'),
required('NODES',[index,cluster,succ]),
'NODES'\index>1,
'NODES'\index=<size('NODES'),
distinct('NODES',index),
'NODES'\succ>1,
'NODES'\succ=<size('NODES')]).

ctr_example(circuit_cluster,
[circuit_cluster(1,
  [[index-1,cluster-1,succ-1],
   [index-2,cluster-1,succ-4],
   [index-3,cluster-2,succ-3],
   [index-4,cluster-2,succ-5],
   [index-5,cluster-3,succ-8],
   [index-6,cluster-3,succ-6],
   [index-7,cluster-3,succ-7],
   [index-8,cluster-4,succ-2],
   [index-9,cluster-4,succ-9]]),
 circuit_cluster(2,
  [[index-1,cluster-1,succ-1],
   [index-2,cluster-1,succ-4],
   [index-3,cluster-2,succ-3],
   [index-4,cluster-2,succ-2],
   [index-5,cluster-3,succ-8],
   [index-6,cluster-3,succ-6],
   [index-7,cluster-3,succ-7],
   [index-8,cluster-4,succ-2],
   [index-9,cluster-4,succ-9]]).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

[index-5,cluster-3,succ-5],
[index-6,cluster-3,succ-9],
[index-7,cluster-3,succ-7],
[index-8,cluster-4,succ-8],
[index-9,cluster-4,succ-6]]).

ctr_typical(
circuit_cluster,
[‘NCIRCUIT’<size(‘NODES’),
 size(‘NODES’)>2,
 range(‘NODES’ˆcluster)>1]).

ctr_exchangeable(circuit_cluster,[items(‘NODES’,all)]).

ctr_graph(
circuit_cluster,
[‘NODES’],
2,
[‘CLIQUE’>>collection(nodes1,nodes2)],
[nodes1^succ=\=nodes1^index,nodes1^succ=nodes2^index],
[‘NTREE’=0,’NSCC’=‘NCIRCUIT’],
[‘ONE_SUCC’],
[ALL_VERTICES>>
 [variables-
 col(‘VARIABLES’-collection(var-dvar),
 [item(var-‘NODES’ˆcluster)])],
 [alldifferent(variables),
 nvalues(variables,=,size(‘NODES’,cluster))]).
B.60  circular_change

◊ Meta-Data:

ctr_date(circular_change, [‘20030820’, ‘20040530’, ‘20060805’]).

ctr_origin(circular_change, ‘Derived from %c.’, [change]).

ctr_arguments(circular_change, [‘NCHANGE’-dvar, ‘VARIABLES’-collection(var-dvar), ‘CTR’-atom]).

ctr_restrictions(circular_change, [‘NCHANGE’>=0, ‘NCHANGE’=<size(‘VARIABLES’), required(‘VARIABLES’, var), in_list(‘CTR’, [=, <=, <, >, >=, =<])].)

ctr_example(circular_change, circular_change(4, [[var-4],[var-4],[var-3],[var-4],[var-1]], =\=)).

ctr_typical(circular_change, [‘NCHANGE’>0, size(‘VARIABLES’) >1, range(‘VARIABLES’~var) >1, in_list(‘CTR’, [=\=])].)

ctr_exchangeable(circular_change, [items(‘VARIABLES’, shift), translate([‘VARIABLES’~var])].)

ctr_graph(circular_change, [‘VARIABLES’], 2, [‘CIRCUIT’>>collection(variables1,variables2)], [‘CTR’(variables1~var,variables2~var)], [‘NARC’=‘NCHANGE’].)
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

\[
\]

\text{ctr\_eval(circular\_change,[automaton(circular\_change\_a)])}.

\text{ctr\_pure\_functional\_dependency(circular\_change,[[]])}.

\text{ctr\_functional\_dependency(circular\_change,1,[2,3])}.

circular\_change\_a(\text{FLAG},NCHANGE,VARIABLES,CTR) :-
\text{collection(VARIABLES,[dvar])},
\text{length(VARIABLES,N)},
\text{check\_type(dvar(0,N),NCHANGE)},
\text{memberchk(CTR,=[,=\=,<,\ge,\gt,\le])},
\text{VARIABLES=[V1\_28445]},
\text{append(VARIABLES,[V1],CVARIABLES)},
circular\_change\_signature(CVARIABLES,SIGNATURE,CTR),
\text{automaton(}
\text{SIGNATURE},
_\text{30210},
\text{SIGNATURE},
[source(s),sink(s)],
[arc(s,0,s),arc(s,1,s,[C+1])],
[C],
[0],
[COUNT]),
COUNT#=NCHANGE#<=>\text{FLAG}.

circular\_change\_signature([],[],28382).

circular\_change\_signature([_28386],[],28385) :-
!.

circular\_change\_signature(
[[var-VAR1],[var-VAR2]|VARs],
[S\|Ss],
=) :-
!,
VAR1#=VAR2#<=>S,
circular\_change\_signature([[var-VAR2]|VARs],Ss,=).

\text{circular\_change\_signature(}
[[var-VAR1],[var-VAR2]|VARs],
[S\|Ss],
=\\text{\textbar}=:)
:-
!,
VAR1\\\textbar\text{\textbackslash}VAR2#<=>S,
circular_change_signature([[var-VAR2]|VARs],Ss,\=\=).

circular_change_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    <) :-
    !,
    VAR1#<VAR2#\=\=S,
    circular_change_signature([[var-VAR2]|VARs],Ss,\<).

circular_change_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    >=) :-
    !,
    VAR1#\=VAR2#\<\=S,
    circular_change_signature([[var-VAR2]|VARs],Ss,\>=).

circular_change_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    >) :-
    !,
    VAR1#\>VAR2#\<\=S,
    circular_change_signature([[var-VAR2]|VARs],Ss,\>).

circular_change_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    =\<) :-
    !,
    VAR1#\=<VAR2#\<\=S,
    circular_change_signature([[var-VAR2]|VARs],Ss,\=<).
B.61 clause_and

◊ Meta-Data:

ctr_date(clause_and, [’20090416’]).

ctr_origin(clause_and, ’Logic’, []).

ctr_synonyms(clause_and, [clause]).

ctr_arguments(
  clause_and,
  ['POSVARS'-collection(var-dvar),
   'NEGVARS'-collection(var-dvar),
   'VAR'-dvar]).

ctr_restrictions(
  clause_and,
  [size('POSVARS')+size('NEGVARS')>0,
   required('POSVARS', var),
   'POSVARS'\var>=0,
   'POSVARS'\var=<1,
   required('NEGVARS', var),
   'NEGVARS'\var>=0,
   'NEGVARS'\var=<1,
   'VAR'=>0,
   'VAR'=<1]).

ctr_example(
  clause_and,
  clause_and([[var-1],[var-0]],[[var-0]],0)).

ctr_typical(clause_and, [size('POSVARS')+size('NEGVARS')>1]).

ctr_exchangeable(
  clause_and,
  [items('POSVARS', all), items('NEGVARS', all)]).

ctr_eval(clause_and, [automaton(clause_and_a)]).

ctr_extensible(clause_and, ['VAR'=0, 'POSVARS', any]).

ctr_extensible(clause_and, ['VAR'=0, 'NEGVARS', any]).

clause_and_a(FLAG, POSVARS, NEGVARS, VAR) :-
  collection(POSVARS, [dvar(0,1)]),

collection(NEGVAR,[dvar(0,1)]),
check_type(dvar(0,1),VAR),
length(POSVAR,LP),
length(NEGVAR,LN),
L is LP+LN,
L>0,
geat_1(POSVAR,LISTP),
geat_1(NEGVAR,LISTN),
clause_and_negate(LISTN,LISTNN),
append([VAR],LISTP,LIST),
append(LIST,LISTNN,LIST_VARIABLES),
AUTOMATON=
automaton(
    LIST_VARIABLES,
    _21590,
    LIST_VARIABLES,
    [source(s),sink(k),sink(j)],
    [arc(s,0,i),
     arc(s,1,j),
     arc(i,0,k),
     arc(i,1,i),
     arc(k,0,k),
     arc(k,1,k),
     arc(j,1,j)],
    [],
    [],
    []),
automaton_bool(FLAG,[0,1],AUTOMATON).

clause_and_negate([],[]).

clause_and_negate([V|R],[U|S]) :-
V#<=> #\U,
clause_and_negate(R,S).
## B.62 clause_or

◊ **Meta-Data:**

```prolog
ctr_date(clause_or,['20090415']).
ctr_origin(clause_or,'Logic',[]).
ctr_synonyms(clause_or,[clause]).
ctr_arguments(
  clause_or,
  ['POSVARS'-collection(var-dvar),
   'NEGVARS'-collection(var-dvar),
   'VAR'-dvar]).
ctr_restrictions(
  clause_or,
  [size('POSVARS')+size('NEGVARS')>0,
   required('POSVARS',var),
   'POSVARS'\^var>=0,
   'POSVARS'\^var=<1,
   required('NEGVARS',var),
   'NEGVARS'\^var>=0,
   'NEGVARS'\^var=<1,
   'VAR'=0,
   'VAR'=<1]).
ctr_example(clause_or,clause_or([[var-0],[var-0]],[[var-0]],1)).
ctr_typical(clause_or,[size('POSVARS')+size('NEGVARS')>1]).
ctr_exchangeable(
  clause_or,
  [items('POSVARS',all),items('NEGVARS',all)]).
ctr_eval(clause_or,[automaton(clause_or_a)]).
ctr_extensible(clause_or,['VAR'=1],'POSVARS',any).
ctr_extensible(clause_or,['VAR'=1],'NEGVARS',any).

clause_or_a(FLAG,POSVARS,NEGVARS,VAR) :-
  collection(POSVARS,[dvar(0,1)]),
  collection(NEGVARS,[dvar(0,1)]),
  check_type(dvar(0,1),VAR),
```

```prolog```
length(POSVARS,LP),
length(NEGVARSLN),
L is LP+LN,
L>0,
get_attr1(POSVARS,LISP),
get_attr1(NEGVARSLISTN),
clause_or_negate(LISTN,LISTNN),
append([VAR],LISP,LIST),
append(LIST,LISTNN,LIST_VARIABLES),
AUTOMATON=
automaton(
    LIST_VARIABLES,
    _22056,
    LIST_VARIABLES,
    [source(s),sink(i),sink(k)],
    [arc(s,0,i),
    arc(s,1,j),
    arc(i,0,i),
    arc(j,0,j),
    arc(j,1,k),
    arc(k,0,k),
    arc(k,1,k)],
    [],
    [],
    []),
automaton_bool(FLAG,[0,1],AUTOMATON).

clause_or_negate([],[]).

clause_or_negate([V|R],[U|S]) :-
V#<=> #\U,
clause_or_negate(R,S).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.63 clique

◊ **META-DATA:**

```prolog
ctr_date(clique, ['20030820', '20040530', '20060805']).

ctr_origin(clique, '\cite{Fahle02}', []).

ctr_arguments(clique,
  ['SIZE_CLIQUE'-dvar,
   'NODES'-collection(index-int, succ-svar)]).

ctr_restrictions(clique,
  ['SIZE_CLIQUE'>=0,
   'SIZE_CLIQUE'<size('NODES'),
   required('NODES', [index, succ]),
   'NODES'\index>=1,
   'NODES'\index=<size('NODES'),
   distinct('NODES', index),
   'NODES'\succ>=1,
   'NODES'\succ=<size('NODES')]).

ctr_example(clique,
  clique(3,
    [[index-1, succ-{}],
     [index-2, succ-{3,5}],
     [index-3, succ-{2,5}],
     [index-4, succ-{}],
     [index-5, succ-{2,3}]]).

ctr_typical(clique,
  ['SIZE_CLIQUE'>=2,
   'SIZE_CLIQUE'<size('NODES'),
   size('NODES')>2]).

ctr_exchangeable(clique, [items('NODES', all)]).

ctr_graph(clique,
  ['NODES'],
  2,
  ..., 2).
```
['CLIQUE' (=\=) >> collection(nodes1, nodes2)],
[nodes2~index in set nodes1^succ],
['NARC'='SIZE_CLIQUE'*'SIZE_CLIQUE'-'SIZE_CLIQUE',
 'NVERTEX'='SIZE_CLIQUE'],
['SYMMETRIC']).

ctr_functional_dependency(clique,1,[2]).
B.64  colored_matrix

◊ Meta-Data:

ctr_predefined(colored_matrix).

ctr_date(colored_matrix,['20031017','20040530']).

ctr_origin(colored_matrix,'KOALOG',[]).

ctr_synonyms(
    colored_matrix,
    [coloured_matrix,cardinality_matrix,card_matrix]).

ctr_arguments(
    colored_matrix,
    ['C'-int,
      'L'-int,
      'K'-int,
      'MATRIX'-collection(column-int,line-int,var-dvar),
      'CPROJ'-collection(column-int,val-int,nocc-dvar),
      'LPROJ'-collection(line-int,val-int,nocc-dvar)]).

ctr_restrictions(
    colored_matrix,
    ['C'>=0,
      'L'>=0,
      'K'>=0,
      required('MATRIX',[column,line,var]),
      increasing_seq('MATRIX',[column,line]),
      size('MATRIX')='C'*'L'+C'+L'+1,
      'MATRIX'\'column>=0,
      'MATRIX'\'column='C',
      'MATRIX'\'line>=0,
      'MATRIX'\'line='L',
      'MATRIX'\'var>=0,
      'MATRIX'\'var='K',
      required('CPROJ',[column,val,nocc]),
      increasing_seq('CPROJ',[column,val]),
      size('CPROJ')='C'*'K'+C'+K'+1,
      'CPROJ'\'column>=0,
      'CPROJ'\'column='C',
      'CPROJ'\'val>=0,
      'CPROJ'\'val='K',
      required('LPROJ',[line,val,nocc]),
      increasing_seq('LPROJ',[line,val]),


size('LPROJ')='L'+'K'+1,
'LPROJ' line>=0,
'LPROJ' line=<L,
'LPROJ' val>=0,
'LPROJ' val=<K']).

ctr_example(
  colored_matrix,
  colored_matrix(
    1,
    2,
    4,
    [[column-0,line-0,var-3],
     [column-0,line-1,var-1],
     [column-0,line-2,var-3],
     [column-1,line-0,var-4],
     [column-1,line-1,var-4],
     [column-1,line-2,var-3]],
    [[column-0,val-0,nocc-0],
     [column-0,val-1,nocc-1],
     [column-0,val-2,nocc-0],
     [column-0,val-3,nocc-2],
     [column-0,val-4,nocc-0],
     [column-1,val-0,nocc-0],
     [column-1,val-1,nocc-0],
     [column-1,val-2,nocc-0],
     [column-1,val-3,nocc-1],
     [column-1,val-4,nocc-2]],
    [[line-0,val-0,nocc-0],
     [line-0,val-1,nocc-0],
     [line-0,val-2,nocc-0],
     [line-0,val-3,nocc-1],
     [line-0,val-4,nocc-1],
     [line-1,val-0,nocc-0],
     [line-1,val-1,nocc-1],
     [line-1,val-2,nocc-0],
     [line-1,val-3,nocc-0],
     [line-1,val-4,nocc-1],
     [line-2,val-0,nocc-0],
     [line-2,val-1,nocc-0],
     [line-2,val-2,nocc-0],
     [line-2,val-3,nocc-2],
     [line-2,val-4,nocc-0]))).

ctr_typical(
  colored_matrix,
appendix b. electronic constraint catalogue

['C' >= 1, 'L' >= 1, 'K' >= 1, range('MATRIX' ^ var) > 1]).

ctr_pure_functional_dependency(colored_matrix, []).  
ctr_functional_dependency(colored_matrix, 5-3, [1, 2, 3]).  
ctr_functional_dependency(colored_matrix, 6-3, [1, 2, 3]).
B.65 coloured_cumulative

◊ META-DATA:

ctr_date(
   coloured_cumulative,
   ['20000128','20030820','20060805']).

ctr_origin(
   coloured_cumulative,
   Derived from %c and %c.,
   [cumulative,nvalues]).

ctr_synonyms(coloured_cumulative,[colored_cumulative]).

ctr_arguments(
   coloured_cumulative,
   [TASKS-
      collection(
         origin-dvar,
         duration-dvar,
         end-dvar,
         colour-dvar),
         'LIMIT'-int]).

ctr_restrictions(
   coloured_cumulative,
   [require_at_least(2,'TASKS',[origin,duration,end]),
    required('TASKS',colour),
    'TASKS' duration>=0,
    'TASKS' origin='TASKS' end,
    'LIMIT'=0]).

ctr_example(
   coloured_cumulative,
   coloured_cumulative(
      [[origin-1,duration-2,end-3,colour-1],
       [origin-2,duration-9,end-11,colour-2],
       [origin-3,duration-10,end-13,colour-3],
       [origin-6,duration-6,end-12,colour-2],
       [origin-7,duration-2,end-9,colour-3]],
      2)).

ctr_typical(
   coloured_cumulative,
   [size('TASKS')>1,
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

range('TASKS'\^origin)>1,
range('TASKS'\^duration)>1,
range('TASKS'\^end)>1,
range('TASKS'\^colour)>1,
'LIMIT'<nval('TASKS'\^colour])).

ctr_exchangeable(
    coloured_cumulative,
    [items('TASKS',all),
    translate([['TASKS'\^origin,'TASKS'\^end]],
    vals(['TASKS'\^colour],int,\=,all,dontcare),
    vals(['LIMIT'],int,<,dontcare,dontcare)]).

ctr_graph(
    coloured_cumulative,
    ['TASKS'],
    1,
    ['SELF'>>collection(tasks)],
    [tasks\^origin+tasks\^duration=tasks\^end],
    ['NARC'=size('TASKS')],
    []).

ctr_graph(
    coloured_cumulative,
    ['TASKS','TASKS'],
    2,
    ['PRODUCT'>>collection(tasks1,tasks2)],
    [tasks1\^duration>0,
    tasks2\^origin=<tasks1\^origin,
    tasks1\^origin<tasks2\^end],
    [],
    ['ACYCLIC','BIPARTITE','NO_LOOP'],
    [Succ>>
    [source,\n    variables-
    col('VARIABLES'\-collection(var-dvar),
    [item(var-'TASKS'\^colour)])],
    [nvalues(variables,=,<,'LIMIT')]].

ctr_eval(
    coloured_cumulative,
    [reformulation(coloured_cumulative_r)]).

ctr_contractible(coloured_cumulative,[],'TASKS',any).

coloured_cumulative_r(TASKS,LIMIT) :-
collection(TASKS, [dvar, dvar_gteq(0), dvar, dvar]),
integer(LIMIT),
LIMIT>=0,
get_attr1(TASKS, ORIGINS),
get_attr2(TASKS, DURATIONS),
get_attr3(TASKS, ENDS),
get_attr4(TASKS, COLOURS),
ori_dur_end(ORIGINS, DURATIONS, ENDS),
coloured_cumulative1(
  ORIGINS,
  ENDS,
  COLOURS,
  1,
  ORIGINS,
  ENDS,
  COLOURS,
  LIMIT).

coloured_cumulative1(
  [],
  [],
  [],
  _49691,
  _49737,
  _49783,
  _49829,
  _49875).

coloured_cumulative1(
  [Oj|RO],
  [Ei|RE],
  [Ci|RC],
  I,
  ORIGINS,
  ENDS,
  COLOURS,
  LIMIT) :-
  coloured_cumulative2(
    ORIGINS,
    ENDS,
    COLOURS,
    1,
    I,
    O1,
    Ei,
    Ci,
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

\[\text{coloured_cumulative1(RO, RE, RC, I1, ORIGINS, ENDS, COLOURS, LIMIT).}\]

\[\text{coloured_cumulative2([], [], [], [], _49694, _49740, _49786, _49832, _49878, []).}\]

\[\text{coloured_cumulative2([_49330|RO], [_49334|RE], [_49338|RC], J, I, Oi, Ei, Ci, [Ci|R]) :- I=J, !, J1 is J+1, coloured_cumulative2(RO,RE,RC,J1,I,Oi,Ei,Ci,R).}\]

\[\text{coloured_cumulative2([Oj|RO], [Ej|RE], [Cj|RC], J, I,}\]

\[\text{Ni in 1..LIMIT, nvalue(Ni,COLi), I1 is I+1,}\]

\[\text{coloured_cumulative1(RO, RE, RC, I1, ORIGINS, ENDS, COLOURS, LIMIT).}\]
Oi,
Ei,
Ci,
[Cij|R]) :-
    I \=\= J,
    K in 1..2,
    fd_min(Ci,Ci_min),
    fd_max(Ci,Ci_max),
    fd_min(Cj,Cj_min),
    fd_max(Cj,Cj_max),
    Min is min(Ci_min,Cj_min),
    Max is max(Ci_max,Cj_max),
    Cij in Min..Max,
    element(K,[Ci,Cj],Cij),
    Oj#=\<Oi#/
Ej#>Oi#/
Cij#=Cj#/
(Oj#>Oi#/
Ej#<Oi#/
Cij#=Ci,
J1 is J+1,
coloured_cumulative2(RO,RE,RC,J1,I,Oi,Ei,Ci,R).
B.66 coloured_cumulatives

◊ Meta-Data:

ctr_date(
    coloured_cumulatives,
    ['20000128','20030820','20060805']).

ctr_origin(
    coloured_cumulatives,
    Derived from %c and %c.,
    [cumulatives,nvalues]).

ctr_synonyms(coloured_cumulatives,[colored_cumulatives]).

ctr_arguments(
    coloured_cumulatives,
    [TASKS-
      collection(
        machine-dvar,
        origin-dvar,
        duration-dvar,
        end-dvar,
        colour-dvar),
        'MACHINES'-collection(id-int,capacity-int)]).

ctr_restrictions(
    coloured_cumulatives,
    [required('TASKS',[machine,colour]),
     require_at_least(2,'TASKS',[origin,duration,end]),
     'TASKS'~duration>=0,
     'TASKS'~origin='TASKS'~end,
     required('MACHINES',[id,capacity]),
     distinct('MACHINES',id),
     'MACHINES'~capacity>=0]).

ctr_example(
    coloured_cumulatives,
    coloured_cumulatives(
      [[machine-1,origin-6,duration-6,end-12,colour-1],
      [machine-1,origin-2,duration-9,end-11,colour-2],
      [machine-2,origin-7,duration-3,end-10,colour-2],
      [machine-1,origin-1,duration-2,end-3,colour-1],
      [machine-2,origin-4,duration-5,end-9,colour-2],
      [machine-1,origin-3,duration-10,end-13,colour-2],
      [[id-1,capacity-2],[id-2,capacity-1]]))).
ctr_typical(
coloured_cumulatives,
[size('TASKS')>1,
range('TASKS'~machine)>1,
range('TASKS'~origin)>1,
range('TASKS'~duration)>1,
range('TASKS'~end)>1,
range('TASKS'~colour)>1,
'TASKS'~duration>0,
size('MACHINES')>1,
'MACHINES'~capacity>0,
'MACHINES'~capacity<nval('TASKS'~colour),
size('TASKS')>size('MACHINES'))).

ctr_exchangeable(
coloured_cumulatives,
[items('TASKS',all),
items('MACHINES',all),
vals(['MACHINES'~capacity],int,<,dontcare,dontcare),
vals(
['TASKS'~machine,'MACHINES'~id],
int,
=\=,
all,
dontcare))).

ctr_graph(
coloured_cumulatives,
['TASKS'],
1,
['SELF'~collection(tasks)],
[tasks~origin+tasks~duration=tasks~end],
['NARC'=size('TASKS')],
[]).

ctr_graph(
coloured_cumulatives,
['TASKS','TASKS'],
2,
foreach('MACHINES',['PRODUCT'~collection(tasks1,tasks2)]),
[tasks1~machine='MACHINES'~id,
tasks1~machine=tasks2~machine,
tasks1~duration>0,
tasks2~origin=<tasks1~origin,
tasks1~origin<tasks2~end],
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

[],
['ACYCLIC','BIPARTITE','NO_LOOP'],
[SUCC>>
[source,
variables-
col('VARIABLES'-collection(var-dvar),
[item(var-'TASKS'\^colour)]),
[nvalues(variables,=<,'MACHINES'\^capacity)]).

ctr_eval(
coloured_cumulatives,
[reformulation(coloured_cumulatives_r)]).

ctr_contractible(coloured_cumulatives,[],'TASKS',any).

coloured_cumulatives_r(TASKS,MACHINES) :-
collection(TASKS,[dvar,dvar,dvar_gteq(0),dvar,dvar]),
get_attr1(TASKS,VMACHINES),
get_attr2(TASKS,ORIGINS),
get_attr3(TASKS,DURATIONS),
get_attr4(TASKS,ENDS),
get_attr5(TASKS,COLOURS),
ori_dur_end(ORIGINS,DURATIONS,ENDS),
collection(MACHINES,[int,int_gteq(0)]),
get_attr1(MACHINES,IDS),
get_attr2(MACHINES,CAPACITIES),
all_different(IDS),
ge maximum(CAPACITIES,CAPA_MAX),
coloured_cumulatives1(
 VMACHINES,
 ORIGINS,
 ENDS,
 COLOURS,
 1,
 VMACHINES,
 ORIGINS,
 ENDS,
 COLOURS,
 IDS,
 CAPACITIES,
 CAPA_MAX).

coloured_cumulatives1(
 [],
 [],
 []),
coloured_cumulatives1([\],_55777,_55823,_55869,_55915,_55961,_56007,_56053,_56099).

coloured_cumulatives1([Mi|RM],[Oi|RO],[Ei|RE],[Ci|RC],I,VMACHINES,ORIGINS,ENDS,COLOURS,IDS,CAPACITIES,CAPA_MAX):- coloured_cumulatives2(VMACHINES,ORIGINS,ENDS,COLOURS,1,I,Mi,Oi,Ei,Ci,COLi),LIMIT in 0..CAPA_MAX, link_index_to_attribute(IDS,CAPACITIES,Mi,LIMIT), Ni in 0..CAPA_MAX, Ni=<LIMIT, nvalue(Ni,COLi), I1 is I+1, coloured_cumulatives1(RM,RO,RE,RC,
coloured_cumulatives2(
    [],
    [],
    [],
    [],
    _55774,
    _55820,
    _55866,
    _55912,
    _55958,
    _56004,
    []).

coloured_cumulatives2(
    [_55362|RM],
    [_55366|RO],
    [_55370|RE],
    [_55374|RC],
    J,
    I,
    Mi,
    Oi,
    Ei,
    Ci,
    [Ci|R]) :-
    I=J,
    !,
    J1 is J+1,
    coloured_cumulatives2(RM,RO,RE,RC,J1,I,Mi,Oi,Ei,Ci,R).

coloured_cumulatives2(
    [Mj|RM],
    [Oj|RO],
    [Ej|RE],
    [Cj|RC],
    J,
    I,
M_i, O_i, E_i, C_i, [C_{ij}|R]) :-
  I \neq J,
  K in 1..2,
  fd_min(C_i, C_{i\_min}),
  fd_max(C_i, C_{i\_max}),
  fd_min(C_j, C_{j\_min}),
  fd_max(C_j, C_{j\_max}),
  Min is min(C_{i\_min}, C_{j\_min}),
  Max is max(C_{i\_max}, C_{j\_max}),
  C_{ij} in Min..Max,
  element(K, [C_i, C_j], C_{ij}),
  M_{j\#} = M_i\# / O_{j\#} < O_i\#/ E_{j\#} > O_i\#/ C_{ij\#} = C_j\# / (M_{j\#} = M_i\# / O_{j\#} < O_i\#/ E_{j\#} = O_i\#) / C_{ij\#} = C_i,
  J_1 is J+1,
  coloured_cumulatives2(RM, RO, RE, RC, J_1, I, M_i, O_i, E_i, C_i, R).
B.67 common

◊ Meta-Data:

ctr_date(common,[‘20000128’,’20030820’,’20060805’]).

ctr_origin(common,’N.˘Beldiceanu’,[]).

ctr_arguments(
    common,
    [‘NCOMMON1’-dvar,
    ‘NCOMMON2’-dvar,
    ‘VARIABLES1’-collection(var-dvar),
    ‘VARIABLES2’-collection(var-dvar)]).

ctr_restrictions(
    common,
    [‘NCOMMON1’>=0,
    ‘NCOMMON1’=<size(‘VARIABLES1’),
    ‘NCOMMON2’>=0,
    ‘NCOMMON2’=<size(‘VARIABLES2’),
    required(‘VARIABLES1’,var),
    required(‘VARIABLES2’,var)]).

ctr_example(
    common,
    common(3,
    4,
    [[var-1],[var-9],[var-1],[var-5]],
    [[var-2],[var-1],[var-9],[var-9],[var-6],[var-9]]).

ctr_typical(
    common,
    [size(‘VARIABLES1’)>1,
    range(‘VARIABLES1’\‘var)>1,
    size(‘VARIABLES2’)>1,
    range(‘VARIABLES2’\‘var)>1]).

ctr_exchangeable(
    common,
    [args(
        [[‘NCOMMON1’,’NCOMMON2’],
        [‘VARIABLES1’,‘VARIABLES2’]],
        items(‘VARIABLES1’,all),
        items(‘VARIABLES2’,all),
        items(‘VARIABLES2’,all),
        items(‘VARIABLES2’,all),
        items(‘VARIABLES2’,all)])}
vals(
    ['VARIABLES1' \^ var,'VARIABLES2' \^ var],
    int,
    =\=, all,
    dontcare))).

ctr_graph(
    common,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT' \=> collection(variables1,variables2)],
    [variables1 \^ var=variables2 \^ var],
    ['NSOURCE'='NCOMMON1','NSINK'='NCOMMON2'],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(common,[reformulation(common_r)]).

ctr_pure_functional_dependency(common,[]).

ctr_functional_dependency(common,1,[3,4]).

ctr_functional_dependency(common,2,[3,4]).

common_r(NCOMMON1,NCOMMON2,VARIABLES1,VARIABLES2) :-
    collection(VARIABLES1,[dvar]),
    collection(VARIABLES2,[dvar]),
    length(VARIABLES1,N1),
    length(VARIABLES2,N2),
    check_type(dvar(0,N1),NCOMMON1),
    check_type(dvar(0,N2),NCOMMON2),
    get_attr1(VARIABLES1,VARS1),
    get_attr1(VARIABLES2,VARS2),
    common1(VARS1,VARS2,_MAT12,SUM1),
    call(NCOMMON1#=SUM1),
    common1(VARS2,VARS1,_MAT21,SUM2),
    call(NCOMMON2#=SUM2).
B.68  common_interval

◊ **Meta-Data:**

ctr_date(common_interval,['20030820','20060805']).

ctr_origin(common_interval,'Derived from %c.',[common]).

ctr_arguments(
  common_interval,
  ['NCOMMON1'-dvar,
   'NCOMMON2'-dvar,
   'VARIABLES1'-collection(var-dvar),
   'VARIABLES2'-collection(var-dvar),
   'SIZE_INTERVAL'-int]).

ctr_restrictions(
  common_interval,
  ['NCOMMON1'>=0,
   'NCOMMON1'=<size('VARIABLES1'),
   'NCOMMON2'>=0,
   'NCOMMON2'=<size('VARIABLES2'),
   required('VARIABLES1',var),
   required('VARIABLES2',var),
   'SIZE_INTERVAL'>0]).

ctr_example(
  common_interval,
  common_interval(3,
    2,
    [[var-8],[var-6],[var-6],[var-0]],
    [[var-7],[var-3],[var-3],[var-3],[var-3],[var-7]],
    3)).

ctr_typical(
  common_interval,
  [size('VARIABLES1')>1,
   range('VARIABLES1'`var)>1,
   size('VARIABLES2')>1,
   range('VARIABLES2'`var)>1,
   'SIZE_INTERVAL'>1,
   'SIZE_INTERVAL'<range('VARIABLES1'`var),
   'SIZE_INTERVAL'<range('VARIABLES2'`var)]).
common_interval,
[\text{args}(
  \text{[}'NCOMMON1','NCOMMON2'\text{]},
  \text{[}'VARIABLES1','VARIABLES2'\text{]},
  \text{[}'SIZE\_INTERVAL'\text{]})],
\text{items('VARIABLES1','all')},
\text{items('VARIABLES2','all')},
\text{vals}(
  \text{[}'VARIABLES1'\text{\textasciicircum}var],
  \text{intervals('SIZE\_INTERVAL')},
  =,\text{dontcare},\text{dontcare}),
\text{vals}(
  \text{[}'VARIABLES2'\text{\textasciicircum}var],
  \text{intervals('SIZE\_INTERVAL')},
  =,\text{dontcare},\text{dontcare})).

\text{ctr}\_\text{graph}(\text{common\_interval},
\text{[}'VARIABLES1','VARIABLES2'\text{]},
2,\text{[}'PRODUCT'\text{\textasciicircum}collection(\text{variables1},\text{variables2})],
\text{[}'VARIABLES1'\text{\textasciicircum}var/'SIZE\_INTERVAL'=
\text{variables2}\_\text{\textasciicircum}var/'SIZE\_INTERVAL'],
\text{[}'NSOURCE'='NCOMMON1','NSINK'='NCOMMON2'],
\text{[}'ACYCLIC','BIPARTITE','NO\_LOOP'\text{]}).

\text{ctr}\_\text{eval}(\text{common\_interval},\text{[reformulation(\text{common\_interval}\_r)\text{]}}).

\text{ctr}\_\text{pure}\_\text{functional}\_\text{dependency}(\text{common\_interval},\text{[}]).

\text{ctr}\_\text{functional}\_\text{dependency}(\text{common\_interval},1,\text{[}3,4,5\text{]}).

\text{ctr}\_\text{functional}\_\text{dependency}(\text{common\_interval},2,\text{[}3,4,5\text{]}).

\text{common\_interval}\_r(\text{NCOMMON1},\text{NCOMMON2},\text{VARIABLES1},\text{VARIABLES2},\text{SIZE\_INTERVAL}) : -
collection(\text{VARIABLES1},\text{[dvar]}),
collection(\text{VARIABLES2},\text{[dvar]}),
length(VARIABLES1,N1),
length(VARIABLES2,N2),
check_type(dvar(0,N1),NCOMMON1),
check_type(dvar(0,N2),NCOMMON2),
integer(SIZE_INTERVAL),
SIZE_INTERVAL>0,
get_attr1(VARIABLES1,VARS1),
get_attr1(VARIABLES2,VARS2),
gen_quotient(VARS1,SIZE_INTERVAL,QUOTVARS1),
gen_quotient(VARS2,SIZE_INTERVAL,QUOTVARS2),
common1(QUOTVARS1,QUOTVARS2,_MAT12,SUM1),
call(NCOMMON1#=SUM1),
color1(QUOTVARS2,QUOTVARS1,_MAT21,SUM2),
call(NCOMMON2#=SUM2).
B.69  common_modulo

◊ **META-DATA:**

```prolog
ctr_date(common_modulo,['20030820','20060806']).

ctr_origin(common_modulo,'Derived from %c.',[common]).

ctr_arguments(
  common_modulo,
  ['NCOMMON1'-dvar,
   'NCOMMON2'-dvar,
   'VARIABLES1'-collection(var-dvar),
   'VARIABLES2'-collection(var-dvar),
   'M'-int]).

ctr_restrictions(
  common_modulo,
  ['NCOMMON1'>=0,
   'NCOMMON1'=<size('VARIABLES1'),
   'NCOMMON2'>=0,
   'NCOMMON2'=<size('VARIABLES2'),
   required('VARIABLES1',var),
   required('VARIABLES2',var),
   'M'>0]).

ctr_example(
  common_modulo,
  common_modulo(3,
    4,
    [[var-0],[var-4],[var-0],[var-8]],
    [[var-7],[var-5],[var-4],[var-9],[var-2],[var-4]],
    5)).

ctr_typical(
  common_modulo,
  [size('VARIABLES1')>1,
   range('VARIABLES1'\^var)>1,
   size('VARIABLES2')>1,
   range('VARIABLES2'\^var)>1,
   'M'>1,
   'M'<maxval('VARIABLES1'\^var),
   'M'<maxval('VARIABLES2'\^var)]).

ctr_exchangeable(
```
common_modulo,
  [args(
    [['NCOMMON1','NCOMMON2'],
    ['VARIABLES1','VARIABLES2'],
    ['M']]),
  items('VARIABLES1',all),
  items('VARIABLES2',all),
  vals(['VARIABLES1\^\text{var}',\text{mod}('M'),=,\text{dontcare},\text{dontcare}],
  vals(['VARIABLES2\^\text{var}',\text{mod}('M'),=,\text{dontcare},\text{dontcare}])].

ctr_graph(
  common_modulo,
  ['VARIABLES1','VARIABLES2'],
  2,
  ['PRODUCT'\Rightarrow\text{collection}(variables1,variables2)],
  [variables1\^\text{var}\text{mod} 'M'=variables2\^\text{var}\text{mod} 'M'],
  ['NSOURCE'='NCOMMON1','NSINK'='NCOMMON2'],
  ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(common_modulo,[\text{reformulation}(common_modulo_r)]).

ctr_pure_functional_dependency(common_modulo,[]).

ctr_functional_dependency(common_modulo,1,[3,4,5]).

ctr_functional_dependency(common_modulo,2,[3,4,5]).

common_modulo_r(NCOMMON1,NCOMMON2,VARIABLES1,VARIABLES2,M) :-
  collection(VARIABLES1,[dvar]),
  collection(VARIABLES2,[dvar]),
  length(VARIABLES1,N1),
  length(VARIABLES2,N2),
  check_type(dvar(0,N1),NCOMMON1),
  check_type(dvar(0,N2),NCOMMON2),
  integer(M),
  M>0,
  get_attr1(VARIABLES1,VARS1),
  get_attr1(VARIABLES2,VARS2),
  gen_remainder(VARS1,M,REMVAR1),
  gen_remainder(VARS2,M,REMVAR2),
  common1(REMVAR1,REMVAR2,_MAT12,SUM1),
  call(NCOMMON1#=SUM1),
  common1(REMVAR2,REMVAR1,_MAT21,SUM2),
  call(NCOMMON2#=SUM2).
B.70 common_partition

◊ META-DATA:

ctr_date(common_partition,['20030820','20060806']).

ctr_origin(common_partition,'Derived from %c.',[common]).

ctr_types(common_partition,['VALUES'-collection(val-int)]).

ctr_arguments(
    common_partition,
    ['NCOMMON1'-dvar,
    'NCOMMON2'-dvar,
    'VARIABLES1'-collection(var-dvar),
    'VARIABLES2'-collection(var-dvar),
    'PARTITIONS'-collection(p-'VALUES'))).

ctr_restrictions(
    common_partition,
    [size('VALUES')=1,
    required('VALUES',val),
    distinct('VALUES',val),
    'NCOMMON1'=0,
    'NCOMMON1'=<size('VARIABLES1'),
    'NCOMMON2'=0,
    'NCOMMON2'=<size('VARIABLES2'),
    required('VARIABLES1',var),
    required('VARIABLES2',var),
    required('PARTITIONS',p),
    size('PARTITIONS')>=2]).

ctr_example(
    common_partition,
    common_partition(
        3,
        4,
        [[var-2],[var-3],[var-6],[var-0]],
        [[var-0],[var-6],[var-3],[var-7],[var-1]],
        [p-[[val-1],[val-3]]],
        [p-[[val-4]]],
        [p-[[val-2],[val-6]]])).

ctr_typical(
    common_partition,
    [size('VARIABLES1')=1,
range('VARIABLES1' \ var)>1,
size('VARIABLES2')>1,
range('VARIABLES2'^var)>1,
size('VARIABLES1')>size('PARTITIONS'),
size('VARIABLES2')>size('PARTITIONS')).

ctr_exchangeable(  
common_partition,
[ args(  
    ['NCOMMON1','NCOMMON2'],
    ['VARIABLES1','VARIABLES2'],
    ['PARTITIONS']),
items('VARIABLES1',all),
items('VARIABLES2',all),
items('PARTITIONS',all),
items('PARTITIONS' p,all),
vals(  
    ['VARIABLES1'^var],
    part('PARTITIONS'),
    =,  
dontcare,  
dontcare),
vals(  
    ['VARIABLES2'^var],
    part('PARTITIONS'),
    =,  
dontcare,  
dontcare)).

ctr_graph(  
common_partition,
['VARIABLES1','VARIABLES2'],
2,
['PRODUCT'=>collection(variables1,variables2)],
[ in_same_partition(  
    variables1^var,  
    variables2^var,  
    PARTITIONS)],
['NSOURCE'='NCOMMON1','NSINK'='NCOMMON2'],
['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(common_partition,[reformulation(common_partition_r)]).

ctr_pure_functional_dependency(common_partition,[]).

ctr_functional_dependency(common_partition,1,[3,4,5]).


ctr_functional_dependency(common_partition,2,[3,4,5]).

common_partition_r(  
    NCOMMON1,  
    NCOMMON2,  
    VARIABLES1,  
    VARIABLES2,  
    PARTITIONS) :-
    collection(VARIABLES1,[dvar]),  
    collection(VARIABLES2,[dvar]),  
    length(VARIABLES1,N1),  
    length(VARIABLES2,N2),  
    check_type(dvar(0,N1),NCOMMON1),  
    check_type(dvar(0,N2),NCOMMON2),  
    collection(PARTITIONS,[col_len_gteq(1,[int])]),  
    length(PARTITIONS,P),  
    P>1,  
    get_attr1(VARIABLES1,VARS1),  
    get_attr1(VARIABLES2,VARS2),  
    get_col_attr1(PARTITIONS,1,PVALS),  
    flattern(PVALS,VALS),  
    all_different(VALS),  
    length(VALS,LPVALS),  
    LPVALS1 is LPVALS+1,  
    get_partition_var(VARS1,VALS,PVARS1,LPVALS1,0),  
    LPVALS2 is LPVALS1+1,  
    get_partition_var(VARS2,VALS,PVARS2,LPVALS2,LPVALS1),  
    common1(PVARS1,PVARS2,_MAT12,SUM1),  
    call(NCOMMON1#=SUM1),  
    common1(PVARS2,PVARS1,_MAT21,SUM2),  
    call(NCOMMON2#=SUM2).
B.71 compare_and_count

♦ Meta-Data:

ctr_predefined(compare_and_count).

ctr_date(compare_and_count,['20110628']).

ctr_origin(compare_and_count,'Generalise %c',[discrepancy]).

ctr_arguments(
  compare_and_count,
  ['VARIABLES1'-collection(var-dvar),
   'VARIABLES2'-collection(var-dvar),
   'COMPARE'-atom,
   'COUNT'-atom,
   'LIMIT'-dvar]).

ctr_restrictions(
  compare_and_count,
  [size('VARIABLES1')=size('VARIABLES2'),
   required('VARIABLES1',var),
   required('VARIABLES2',var),
   in_list('COMPARE',=[=,\=,<,\>=,>,=<]),
   in_list('COUNT',=[=,\=,<,\>=,>,=<]),
   'LIMIT'>=0]).

ctr_example(
  compare_and_count,
  compare_and_count(
    [[var-4],[var-5],[var-5],[var-4],[var-5]],
    [[var-4],[var-2],[var-5],[var-1],[var-5]],
    =,
    =<,
    3)).

ctr_typical(
  compare_and_count,
  [size('VARIABLES1')>1,
   range('VARIABLES1'\^var)>1,
   range('VARIABLES2'\^var)>1,
   in_list('COMPARE',[=]),
   in_list('COUNT',=[=,\=,<,\>=,>,=<]),
   'LIMIT'>0,
   'LIMIT'<size('VARIABLES1')]).
ctr_eval(
    compare_and_count,
    [reformulation(compare_and_count_r)].
)

ctr_pure_functional_dependency(
    compare_and_count,
    [in_list('COUNT',[=])].
)

ctr_contractible(
    compare_and_count,
    [in_list('COUNT',[<,=<])],
    ['VARIABLES1','VARIABLES2'],
    any).

ctr_extensible(
    compare_and_count,
    [in_list('COUNT',[>=,>])],
    ['VARIABLES1','VARIABLES2'],
    any).

compare_and_count_r(VARIABLES1,VARIABLES2,COMPARE,COUNT,LIMIT) :-
    collection(VARIABLES1,[dvar]),
    collection(VARIABLES2,[dvar]),
    length(VARIABLES1,N1),
    length(VARIABLES2,N2),
    N1=N2,
    memberchk(COMPARE,[=,\=,<,\>=,\>=,=<]),
    memberchk(COUNT,[=,\=,<,\>=,\>=,=<]),
    check_type(dvar,LIMIT),
    LIMIT#>=0,
    get_attr1(VARIABLES1,VARS1),
    get_attr1(VARIABLES2,VARS2),
    compare_and_count_r1(VARS1,VARS2,COMPARE,TERM),
    compare_and_count_r2(COUNT,TERM,LIMIT).

compare_and_count_r1([],[],_13826,0).

compare_and_count_r1([[V1|R1],[V2|R2],=,B+T] :-
    V1#=V2#<=>B,
    compare_and_count_r1(R1,R2,=,T).

compare_and_count_r1([[V1|R1],[V2|R2],\=,B+T] :-
    V1\=V2#<=>B,
    compare_and_count_r1(R1,R2,\=,T).

compare_and_count_r1([[V1|R1],[V2|R2],<,B+T] :-

V1#<V2#<=>B,
compare_and_count_r1(R1,R2,<,T).

compare_and_count_r1([V1|R1],[V2|R2],>=,B+T) :-
V1#=V2#<=>B,
compare_and_count_r1(R1,R2,>=,T).

compare_and_count_r1([V1|R1],[V2|R2],>,B+T) :-
V1#>V2#<=>B,
compare_and_count_r1(R1,R2,>,T).

compare_and_count_r1([V1|R1],[V2|R2],=<,B+T) :-
V1#=<V2#<=>B,
compare_and_count_r1(R1,R2,=<,T).

compare_and_count_r2(=,TERM,LIMIT) :-
call(TERM#=LIMIT).

compare_and_count_r2(\=,TERM,LIMIT) :-
call(TERM\=LIMIT).

compare_and_count_r2(<,TERM,LIMIT) :-
call(TERM<LIMIT).

compare_and_count_r2(\=,TERM,LIMIT) :-
call(TERM\=LIMIT).

compare_and_count_r2(\>,TERM,LIMIT) :-
call(TERM\>LIMIT).

compare_and_count_r2(\=,TERM,LIMIT) :-
call(TERM\=LIMIT).
B.72 cond_lex_cost

◊ META-DATA:

ctr_date(cond_lex_cost, [’20060416’]).

ctr_origin(
    cond_lex_cost,
    Inspired by \cite{WallaceWilson06}.,
    []).

ctr_types(cond_lex_cost, [’TUPLE_OF_VALS’-collection(val-int)]).

ctr_arguments(
    cond_lex_cost,
    [’VECTOR’-collection(var-dvar),
     ’PREFERENCE_TABLE’-collection(tuple-’TUPLE_OF_VALS’),
     ’COST’-dvar]).

ctr_restrictions(
    cond_lex_cost,
    [size(’TUPLE_OF_VALS’) >= 1,
     required(’TUPLE_OF_VALS’, val),
     required(’VECTOR’, var),
     size(’VECTOR’) = size(’TUPLE_OF_VALS’),
     required(’PREFERENCE_TABLE’, tuple),
     same_size(’PREFERENCE_TABLE’, tuple),
     distinct(’PREFERENCE_TABLE’, []),
     in_relation(’VECTOR’, ’PREFERENCE_TABLE’),
     ’COST’ >= 1,
     ’COST’ =< size(’PREFERENCE_TABLE’)]).

ctr_example(
    cond_lex_cost,
    cond_lex_cost(
        [[var-0],[var-1]],
        [[tuple-[[val-1],[val-0]]],
         [tuple-[[val-0],[val-1]]],
         [tuple-[[val-0],[val-0]]],
         [tuple-[[val-1],[val-1]]]],
        2)).

ctr_typical(
    cond_lex_cost,
    [size(’TUPLE_OF_VALS’) > 1,
     size(’VECTOR’) > 1,
size('PREFERENCE_TABLE'>1)).

ctr_exchangeable(
  cond_lex_cost,
  [items_sync('VECTOR','PREFERENCE_TABLE'\ tuple,all),
   vals(
     ['VECTOR','PREFERENCE_TABLE'\ tuple],
     int,
     =$=,
     all,
     dontcare))).

ctr_eval(cond_lex_cost,[automata(cond_lex_cost_a)]).

cond_lex_cost_a(VECTOR,PREFERENCE_TABLE,COST) :-
  collection(VECTOR,[dvar]),
  collection(PREFERENCE_TABLE,[col([dvar])]),
  same_size(PREFERENCE_TABLE),
  check_type(dvar,COST),
  length(PREFERENCE_TABLE,LP),
  COST#>=1,
  COST#=<LP,
  PREFERENCE_TABLE=\ [[22270-L]|22266],
  length(VECTOR,LV),
  length(L,N),
  N>=1,
  LV=N,
  create_collection(PREFERENCE_TABLE,vec,var,PREF),
  eval(lex_alldifferent(PREF)),
  eval(in_relation(VECTOR,PREFERENCE_TABLE)),
  cond_lex(VECTOR,PREFERENCE_TABLE,COST).
\textbf{B.73  \texttt{cond\_lex\_greater}}

\textbf{\texttt{\diamond  \textsc{Meta-Data:}}}

\texttt{ctr\_date(\texttt{cond\_lex\_greater,[‘20060430’]}).}

\texttt{ctr\_origin(}
\texttt{cond\_lex\_greater,}
\texttt{Inspired by \cite{WallaceWilson06}.,}
\texttt{[]).}

\texttt{ctr\_types(}
\texttt{cond\_lex\_greater, }
\texttt{[‘TUPLE\_OF\_VALS’-collection(val\-int)])}.  

\texttt{ctr\_arguments(}
\texttt{cond\_lex\_greater,}
\texttt{[‘VECTOR1’-collection(var\-dvar),}
\texttt{’VECTOR2’-collection(var\-dvar),}
\texttt{’PREFERENCE\_TABLE’-collection(tuple-’TUPLE\_OF\_VALS’)]).}

\texttt{ctr\_restrictions(}
\texttt{cond\_lex\_greater,}
\texttt{[size(‘TUPLE\_OF\_VALS’)>=1,}
\texttt{required(‘TUPLE\_OF\_VALS’,val),}
\texttt{required(‘VECTOR1’,var),}
\texttt{required(‘VECTOR2’,var),}
\texttt{size(‘VECTOR1’)=size(‘VECTOR2’),}
\texttt{size(‘VECTOR1’)=size(‘TUPLE\_OF\_VALS’),}
\texttt{required(‘PREFERENCE\_TABLE’,tuple),}
\texttt{same\_size(‘PREFERENCE\_TABLE’,tuple),}
\texttt{distinct(‘PREFERENCE\_TABLE’,[]),}
\texttt{in\_relation(‘VECTOR1’,’PREFERENCE\_TABLE’),}
\texttt{in\_relation(‘VECTOR2’,’PREFERENCE\_TABLE’)]}.}

\texttt{ctr\_example(}
\texttt{cond\_lex\_greater,}
\texttt{cond\_lex\_greater(}
\texttt{[[\texttt{var-0}],[\texttt{var-0}]],}
\texttt{[[\texttt{var-1}],[\texttt{var-0}]],}
\texttt{[[\texttt{tuple-[[\texttt{val-1}},[\texttt{val-0}])],}
\texttt{[\texttt{tuple-[[\texttt{val-0}},[\texttt{val-1}])],}
\texttt{[\texttt{tuple-[[\texttt{val-0}],[\texttt{val-0}])],}
\texttt{[\texttt{tuple-[[\texttt{val-1}},[\texttt{val-1}])]]]))).}

\texttt{ctr\_typical(}
cond_lex_greater,
[size('TUPLE_OF_VALS')>1,
 size('VECTOR1')>1,
 size('VECTOR2')>1,
 size('PREFERENCE_TABLE')>1]).

ctr_exchangeable(
  cond_lex_greater,
  [items_sync(
      VECTOR1,
      VECTOR2,
      'PREFERENCE_TABLE'^tuple,
      all),
   vals(
      ['VECTOR1','VECTOR2','PREFERENCE_TABLE'^tuple],
      int,
      =\=,
      all,
      dontcare)]).

ctr_eval(cond_lex_greater, [automata(cond_lex_greater_a)]).

cond_lex_greater_a(VECTOR1,VECTOR2,PREFERENCE_TABLE) :-
  collection(VECTOR1,[dvar]),
  collection(VECTOR2,[dvar]),
  collection(PREFERENCE_TABLE,[col([dvar])]),
  same_size(PREFERENCE_TABLE),
  PREFERENCE_TABLE=[[__22219-L]|_R],
  length(VECTOR1,LV1),
  length(VECTOR2,LV2),
  length(L,N),
  N>=1,
  LV1=LV2,
  LV1=N,
  create_collection(PREFERENCE_TABLE,vec,var,PREF),
  eval(lex_alldifferent(PREF)),
  eval(in_relation(VECTOR1,PREFERENCE_TABLE)),
  eval(in_relation(VECTOR2,PREFERENCE_TABLE)),
  cond_lex(VECTOR1,VECTOR2,PREFERENCE_TABLE,I,J),
  I#>J.
B.74  cond_lex_greatereq

◊ META-DATA:

ctr_date(cond_lex_greatereq,['20060416']).

ctr_origin(
   cond_lex_greatereq,
   Inspired by \cite{WallaceWilson06},\[
   ]).

ctr_types(
   cond_lex_greatereq,
   ['TUPLE_OF_VALS'-collection(val-int)]).

ctr_arguments(
   cond_lex_greatereq,
   ['VECTOR1'-collection(var-dvar),
   'VECTOR2'-collection(var-dvar),
   'PREFERENCE_TABLE'-collection(tuple-'TUPLE_OF_VALS')]).

ctr_restrictions(
   cond_lex_greatereq,
   [size('TUPLE_OF_VALS')>=1,
    required('TUPLE_OF_VALS',val),
    required('VECTOR1',var),
    required('VECTOR2',var),
    size('VECTOR1')=size('VECTOR2'),
    size('VECTOR1')=size('TUPLE_OF_VALS'),
    required('PREFERENCE_TABLE',tuple),
    same_size('PREFERENCE_TABLE',tuple),
    distinct('PREFERENCE_TABLE',[]),
    in_relation('VECTOR1','PREFERENCE_TABLE'),
    in_relation('VECTOR2','PREFERENCE_TABLE')]).

ctr_example(
   cond_lex_greatereq,
   cond_lex_greatereq(
     [[var-0],[var-0]],
     [[var-1],[var-0]],
     [[tuple-[[val-1],[val-0]]],
      tuple-[[val-0],[val-1]]],
     [tuple-[[val-0],[val-0]]],
     [tuple-[[val-1],[val-1]]])).

ctr_typical(}
cond_lex_greatereq,
[size('TUPLE_OF_VALS')>1,
 size('VECTOR1')>1,
 size('VECTOR2')>1,
 size('PREFERENCE_TABLE')>1]).

ctr_exchangeable(
   cond_lex_greatereq,
   [items_sync(
       VECTOR1,
       VECTOR2,
       'PREFERENCE_TABLE'\^tuple,
       all),
    vals(
       ['VECTOR1','VECTOR2','PREFERENCE_TABLE'\^tuple],
       int,
       =\=,
       all,
       dontcare))].

ctr_eval(cond_lex_greatereq,[automata(cond_lex_greatereq_a)]).

cond_lex_greatereq_a(VECTOR1,VECTOR2,PREFERENCE_TABLE) :-
collection(VECTOR1,[dvar]),
collection(VECTOR2,[dvar]),
collection(PREFERENCE_TABLE,[col([dvar])]),
same_size(PREFERENCE_TABLE),
PREFERENCE_TABLE=[[22247-L]|_R],
length(VECTOR1,LV1),
length(VECTOR2,LV2),
length(L,N),
N>=1,
LV1=LV2,
LV1=N,
create_collection(PREFERENCE_TABLE,vec,var,PREF),
eval(lex_alldifferent(PREF)),
eval(in_relation(VECTOR1,PREFERENCE_TABLE)),
eval(in_relation(VECTOR2,PREFERENCE_TABLE)),
cond_lex(VECTOR1,VECTOR2,PREFERENCE_TABLE,I,J),
I#>=J.
B.75 cond_lex_less

◊ **META-DATA:**

```prolog
ctr_date(cond_lex_less, ['20060430']).
```

```prolog
ctr_origin(
    cond_lex_less,
    Inspired by \cite{WallaceWilson06},.
).
```

```prolog
ctr_types(cond_lex_less, ['TUPLE_OF_VALS'-collection(val-int)]).
```

```prolog
ctr_arguments(
    cond_lex_less,
    ['VECTOR1'-collection(var-dvar),
     'VECTOR2'-collection(var-dvar),
     'PREFERENCE_TABLE'-collection(tuple-'TUPLE_OF_VALS'))).
```

```prolog
ctr_restrictions(
    cond_lex_less,
    [size('TUPLE_OF_VALS')>=1,
     required('TUPLE_OF_VALS',val),
     required('VECTOR1',var),
     required('VECTOR2',var),
     size('VECTOR1')=size('VECTOR2'),
     size('VECTOR1')=size('TUPLE_OF_VALS'),
     required('PREFERENCE_TABLE',tuple),
     same_size('PREFERENCE_TABLE',tuple),
     distinct('PREFERENCE_TABLE',[]),
     in_relation('VECTOR1','PREFERENCE_TABLE'),
     in_relation('VECTOR2','PREFERENCE_TABLE'))).
```

```prolog
ctr_example(
    cond_lex_less,
    cond_lex_less(
        [[var-1],[var-0]],
        [[var-0],[var-0]],
        [[tuple-[[val-1],[val-0]]],
         [tuple-[[val-0],[val-1]]],
         [tuple-[[val-0],[val-0]]],
         [tuple-[[val-1],[val-1]]])).
```

```prolog
ctr_typical(
    cond_lex_less,
    [size('TUPLE_OF_VALS')>1,
     "\cite{WallaceWilson06}\]
)\]]}).
```

```prolog
ctr_example(
    cond_lex_less,
    cond_lex_less(
        [[var-1],[var-0]],
        [[var-0],[var-0]],
        [[tuple-[[val-1],[val-0]]],
         [tuple-[[val-0],[val-1]]],
         [tuple-[[val-0],[val-0]]],
         [tuple-[[val-1],[val-1]]])).
```

```prolog
ctr_typical(
    cond_lex_less,
    [size('TUPLE_OF_VALS')>1,
     "\cite{WallaceWilson06}\]
)\]]}).
```

```prolog
ctr_example(
    cond_lex_less,
    cond_lex_less(
        [[var-1],[var-0]],
        [[var-0],[var-0]],
        [[tuple-[[val-1],[val-0]]],
         [tuple-[[val-0],[val-1]]],
         [tuple-[[val-0],[val-0]]],
         [tuple-[[val-1],[val-1]]])).
```

```prolog
ctr_typical(
    cond_lex_less,
    [size('TUPLE_OF_VALS')>1,
     "\cite{WallaceWilson06}\]
)\]]}).
```

```prolog
ctr_example(
    cond_lex_less,
    cond_lex_less(
        [[var-1],[var-0]],
        [[var-0],[var-0]],
        [[tuple-[[val-1],[val-0]]],
         [tuple-[[val-0],[val-1]]],
         [tuple-[[val-0],[val-0]]],
         [tuple-[[val-1],[val-1]]])).
```

```prolog
ctr_typical(
    cond_lex_less,
    [size('TUPLE_OF_VALS')>1,
     "\cite{WallaceWilson06}\]
)\]]}).
```

```prolog
ctr_example(
    cond_lex_less,
    cond_lex_less(
        [[var-1],[var-0]],
        [[var-0],[var-0]],
        [[tuple-[[val-1],[val-0]]],
         [tuple-[[val-0],[val-1]]],
         [tuple-[[val-0],[val-0]]],
         [tuple-[[val-1],[val-1]]])).
```

```prolog
ctr_typical(
    cond_lex_less,
    [size('TUPLE_OF_VALS')>1,
     "\cite{WallaceWilson06}\]
)\]]}).
```
size('VECTOR1')>1,
size('VECTOR2')>1,
size('PREFERENCE_TABLE')>1).

ctr_exchangeable(
    cond_lex_less,
    [items_sync(
        VECTOR1,
        VECTOR2,
        'PREFERENCE_TABLE'ˆtuple,
        all),
    vals(
        ['VECTOR1','VECTOR2','PREFERENCE_TABLE'ˆtuple],
        int,
        =\=,
        all,
        dontcare)])).

ctr_eval(cond_lex_less,[automata(cond_lex_less_a)])).

cond_lex_less_a(VECTOR1,VECTOR2,PREFERENCE_TABLE) :-
collection(VECTOR1,[dvar]),
collection(VECTOR2,[dvar]),
collection(PREFERENCE_TABLE,[col([dvar])]),
same_size(PREFERENCE_TABLE),
PREFERENCE_TABLE=([_22153-L]|_R],
length(VECTOR1,LV1),
length(VECTOR2,LV2),
length(L,N),
N>=1,
LV1=LV2,
LV1=N,
create_collection(PREFERENCE_TABLE,vec,var,PREF),
eval(lex_alldifferent(PREF)),
eval(in_relation(VECTOR1,PREFERENCE_TABLE)),
eval(in_relation(VECTOR2,PREFERENCE_TABLE)),
cond_lex(VECTOR1,VECTOR2,PREFERENCE_TABLE,I,J),
I#<J.
B.76 cond_lex_lesseq

◊ Meta-Data:

ctr_date(cond_lex_lesseq, ['20060416']).

ctr_origin(
    cond_lex_lesseq,
    Inspired by \cite{WallaceWilson06}.,
    []).

ctr_types(
    cond_lex_lesseq,
    ['TUPLE_OF_VALS' - collection(val-int)]).

ctr_arguments(
    cond_lex_lesseq,
    ['VECTOR1' - collection(var-dvar),
     'VECTOR2' - collection(var-dvar),
     'PREFERENCE_TABLE' - collection(tuple-'TUPLE_OF_VALS' )]).

ctr_restrictions(
    cond_lex_lesseq,
    [size('TUPLE_OF_VALS')>=1,
     required('TUPLE_OF_VALS',val),
     required('VECTOR1',var),
     required('VECTOR2',var),
     size('VECTOR1')=size('VECTOR2'),
     size('VECTOR1')=size('TUPLE_OF_VALS'),
     required('PREFERENCE_TABLE',tuple),
     same_size('PREFERENCE_TABLE',tuple),
     distinct('PREFERENCE_TABLE',[]),
     in_relation('VECTOR1','PREFERENCE_TABLE'),
     in_relation('VECTOR2','PREFERENCE_TABLE')]).

ctr_example(
    cond_lex_lesseq,
    cond_lex_lesseq(
        [[var-1],[var-0]],
        [[var-0],[var-0]],
        [[tuple-[[val-1],[val-0]]],
         tuple-[[val-0],[val-1]]],
        [tuple-[[val-0],[val-0]]],
        [tuple-[[val-1],[val-1]]])).

ctr_typical(}
cond_lex_lesseq,
[size('TUPLE_OF_VALS')>1,
 size('VECTOR1')>1,
 size('VECTOR2')>1,
 size('PREFERENCE_TABLE')>1]).

ctr_exchangeable(
  cond_lex_lesseq,
  [items_sync(VECTOR1,VECTOR2,'PREFERENCE_TABLE'\^tuple,all),
   vals(['VECTOR1','VECTOR2','PREFERENCE_TABLE'\^tuple],
       int,=\=,all,dontcare)]).

ctr_eval(cond_lex_lesseq,[automata(cond_lex_lesseq_a)]).

cond_lex_lesseq_a (VECTOR1,VECTOR2,PREFERENCE_TABLE) :-
collection(VECTOR1,[dvar]),
collection(VECTOR2,[dvar]),
collection(PREFERENCE_TABLE,[col([dvar])]),
same_size(PREFERENCE_TABLE),
PREFERENCE_TABLE=[\_22181-L|_R],
length(VECTOR1,LV1),
length(VECTOR2,LV2),
length(L,N),
N>=1,
LV1=LV2,
LV1=N,
create_collection(PREFERENCE_TABLE,vec,var,PREF),
eval(lex_alldifferent(PREF)),
eval(in_relation(VECTOR1,PREFERENCE_TABLE)),
eval(in_relation(VECTOR2,PREFERENCE_TABLE)),
cond_lex(VECTOR1,VECTOR2,PREFERENCE_TABLE,I,J),
I#=<J.
B.77 connect_points

◊ META-DATA:

```
ctr_date(
    connect_points,
    ['20000128','20030820','20040530','20060806']).
```

```
ctr_origin(connect_points,'N.~Beldiceanu',[]).
```

```
ctr_arguments(
    connect_points,
    ['SIZE1'-int,
     'SIZE2'-int,
     'SIZE3'-int,
     'NGROUP'-dvar,
     'POINTS'-collection(p-dvar)]).
```

```
ctr_restrictions(
    connect_points,
    ['SIZE1'>0,
     'SIZE2'>0,
     'SIZE3'>0,
     'NGROUP'>=0,
     'NGROUP'=<size('POINTS'),
     'SIZE1'/'SIZE2'/'SIZE3'=size('POINTS'),
     required('POINTS',p)]).
```

```
ctr_example(
    connect_points,
    connect_points(
        8,
        4,
        2,
        2,
        [p-0],
        [p-0],
        [p-1],
        [p-1],
        [p-0],
        [p-2],
        [p-0],
        [p-0],
        [p-0],
        [p-0],
        [p-0],
```
ctr_typical(
    connect_points,
    ['SIZE1'>1,
    'SIZE2'>1,
    'NGROUP'>0,
    'NGROUP'<size('POINTS'),
    size('POINTS')>3]).

ctr_exchangeable(
    connect_points,
    vals(['POINTS' p],int(\=(0)),\=,all,dontcare)).

ctr_graph(
    connect_points,
    ['POINTS'],
    2,
    ['GRID'(['SIZE1','SIZE2','SIZE3'])>>
      collection(point1,point2)],
    [point1 p=\=0,point1 p=point2 p],
    ['NSCC'='NGROUP'],
    ['SYMMETRIC']).

ctr_functional_dependency(connect_points,4,[1,2,3,5]).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.78 connected

◊ META-DATA:

```prolog
ctr_date(connected,['20061001']).

ctr_origin(connected,'\cite{Dooms06}',[]).

ctr_arguments(connected,
    ['NODES'-collection(index-int,succ-svar)]).

ctr_restrictions(connected,
    [required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES'))].

ctr_example(connected,
    connected(
        [[index-1,succ-{1,2,3}],
         [index-2,succ-{1,3}],
         [index-3,succ-{1,2,4}],
         [index-4,succ-{3,5,6}],
         [index-5,succ-{4}],
         [index-6,succ-{4}]]).

ctr_typical(connected,[size('NODES')>1]).

ctr_exchangeable(connected,[items('NODES',all)]).

ctr_graph(connected,['NODES'],2,
    ['CLIQUE']>collection(nodes1,nodes2)],
    [nodes2^index in_set nodes1^succ],
    ['NCC'=1],
    ['SYMMETRIC']).
```
B.79  consecutive_groups_of_ones

◊  **META-DATA:**

```prolog
ctr_date(consecutive_groups_of_ones,['20091227']).

ctr_origin(
    consecutive_groups_of_ones,
    Derived from %c,
    [group]).

ctr_arguments(
    consecutive_groups_of_ones,
    ['GROUP_SIZES'-collection(nb-int),
     'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    consecutive_groups_of_ones,
    [required('GROUP_SIZES',nb),
     size('GROUP_SIZES')>=1,
     'GROUP_SIZES'^nb>=1,
     'GROUP_SIZES'^nb=<size('VARIABLES'),
     required('VARIABLES',var),
     size('VARIABLES')>=2*size('GROUP_SIZES')-1,
     size('VARIABLES')>=
     sum('GROUP_SIZES'^nb)+size('GROUP_SIZES')-1,
     'VARIABLES'^var>=0,
     'VARIABLES'^var=<1]).

ctr_example(
    consecutive_groups_of_ones,
    consecutive_groups_of_ones(
        [[nb-2],[nb-1]],
        [[var-1],
         [var-1],
         [var-0],
         [var-0],
         [var-0],
         [var-0],
         [var-0],
         [var-0]])).

ctr_typical(
    consecutive_groups_of_ones,
    [size('VARIABLES')>1,range('VARIABLES'^var)>1]).

ctr_exchangeable(

consecutive_groups_of_ones,
[items_sync('GROUP_SIZES','VARIABLES',reverse)]).

ctr_eval(
   consecutive_groups_of_ones,
   [automaton(consecutive_groups_of_ones_a)])).

consecutive_groups_of_ones_a(FLAG,GROUP_SIZES,VARIABLES) :-
collection(VARIABLES,[dvar(0,1)]),
length(VARIABLES,N),
collection(GROUP_SIZES,[int(1,N)]),
length(GROUP_SIZES,M),
M>=1,
N>=M,
N>=2*M-1,
get_attr1(GROUP_SIZES,SIZES),
get_attr1(VARIABLES,VARS),
get_sum(SIZES,S),
N>=S+M-1,
consecutive_groups_of_ones_transitions(
   SIZES,
   -1,
   TRANSITIONS,
   LAST),
AUTOMATON=automaton( 
   VARS,
   _20360,
   VARS,
   [source(0),sink(LAST)],
   TRANSITIONS,
   [],
   [],
   []),
automaton_bool(FLAG,[0,1],AUTOMATON).

consecutive_groups_of_ones_transitions([],P,[arc(P,0,P)],P).

consecutive_groups_of_ones_transitions([N|R],P,L,Last) :-
P1 is P+1,
PN is N+P1,
( P>=0 ->
   Li=[arc(P,0,P1),arc(P1,0,P1)]
;   Li=[arc(P1,0,P1)]
),
consecutive_groups_of_ones_trans(N,P1,L2),
consecutive_groups_of_ones_transitions(R,PN,L3,Last),
append(L1,L2,L12),
append(L12,L3,L).

consecutive_groups_of_ones_trans(0,_17070,[[]) :- !.

consecutive_groups_of_ones_trans(I,P,[arc(P,1,P1)|R]) :-
  I>0,
P1 is P+1,
I1 is I-1,
consecutive_groups_of_ones_trans(I1,P1,R).
B.80 consecutive_values

◊ **Meta-Data:**

```prolog
ctr_predefined(consecutive_values).
ctr_date(consecutive_values,[’20100106’]).
```

```prolog
ctr_origin(  
    consecutive_values,  
    Derived from ¢c.,  
    [alldifferent_consecutive_values]).
```

```prolog
ctr_arguments(  
    consecutive_values,  
    [’VARIABLES’-collection(var-dvar)]).
```

```prolog
ctr_restrictions(  
    consecutive_values,  
    [required(’VARIABLES’,var)]).
```

```prolog
ctr_example(  
    consecutive_values,  
    consecutive_values([[var-5],[var-4],[var-3],[var-5]])).
```

```prolog
ctr_typical(  
    consecutive_values,  
    [size(’VARIABLES’)>1,range(’VARIABLES’^var)>1]).
```

```prolog
ctr_exchangeable(  
    consecutive_values,  
    [items(’VARIABLES’,all),translate([’VARIABLES’^var])].
```

```prolog
ctr_eval(  
    consecutive_values,  
    [checker(consecutive_values_c),  
      reformulation(consecutive_values_r)]).
```

```prolog
consecutive_values_c([]) :- !.
```

```prolog
consecutive_values_c(VARIABLES) :-  
    collection(VARIABLES,[int]),  
    get_attr1(VARIABLES,VARS),  
    min_member(MIN,VARS),  
    max_member(MAX,VARS),  
```
sort(VARS,S),
length(S,NVAL),
NVAL is MAX-MIN+1.

consecutive_values_r([]) :- !.

consecutive_values_r(VARIABLES) :-
  collection(VARIABLES,[dvar]),
  get_attr1(VARIABLES,VARS),
  minimum(MIN,VARS),
  maximum(MAX,VARS),
  length(VARIABLES,N),
  NVAL in 1..N,
  nvalue(NVAL,VARS),
  NVAL#=MAX-MIN+1.
B.81  \texttt{contains\_sboxes}

\textbf{Meta-Data:}

\begin{verbatim}
ctr_date(contains\_sboxes, ['20070622', '20090725']).

ctr_origin(contains\_sboxes,
    Geometry, derived from \cite{RandellCuiCohn92}, []).

ctr_synonyms(contains\_sboxes, [contains]).

ctr_types(contains\_sboxes,
    ['VARIABLES'-collection(v-dvar),
     'INTEGERS'-collection(v-int),
     'POSITIVES'-collection(v-int)]).

ctr_arguments(contains\_sboxes,
    ['K'-int,
     'DIMS'-sint,
     'OBJECTS'-collection(oid-int,sid-int,x-'VARIABLES'),
     'SBOXES'-collection(sid-int,t-'INTEGERS',l-'POSITIVES')]).

ctr_restrictions(contains\_sboxes,
    [size('VARIABLES')>=1,
     size('INTEGERS')>=1,
     size('POSITIVES')>=1,
     required('VARIABLES',v),
     size('VARIABLES')='K',
     required('INTEGERS',v),
     size('INTEGERS')='K',
     required('POSITIVES',v),
     size('POSITIVES')='K',
     'POSITIVES'\_v>0,
     'K'>0,
     'DIMS'>=0,
     'DIMS'<'K',
     increasing_seq('OBJECTS', [oid]),
     required('OBJECTS', [oid,sid,x]),
     'OBJECTS'^oid=1,
     'OBJECTS'^oid=<size('OBJECTS'),
     'OBJECTS'^sid=1,
     ...
     ]).
\end{verbatim}
'OBJECTS' \( \text{sid} = \text{size('SBOXES')}, \)
\( \text{size('SBOXES')} \geq 1, \)
\( \text{required('SBOXES', [\text{sid}, \text{t}, \text{l}])}, \)
\( 'SBOXES' \text{sid} = \text{size('SBOXES')}, \)
\( 'SBOXES' \text{sid} = \text{size('SBOXES')}, \)
\( \text{do\_not\_overlap('SBOXES')} \).

ctr\_example(
    contains\_sboxes,
    contains\_sboxes(2,
        {0,1},
        [[\text{oid-1}, \text{sid-1}, x-[\text{v-1}, \text{v-1}]],
        [\text{oid-2}, \text{sid-2}, x-[\text{v-2}, \text{v-2}]],
        [\text{oid-3}, \text{sid-3}, x-[\text{v-3}, \text{v-3}]]],
        [[\text{sid-1}, \text{t}-[\text{v-0}, \text{v-0}], \text{l}-[\text{v-5}, \text{v-5}]],
        [\text{sid-2}, \text{t}-[\text{v-0}, \text{v-0}], \text{l}-[\text{v-3}, \text{v-3}]],
        [\text{sid-3}, \text{t}-[\text{v-0}, \text{v-0}], \text{l}-[\text{v-1}, \text{v-1}]]]).

ctr\_typical(contains\_sboxes, [\text{size('OBJECTS')} > 1]).

ctr\_exchangeable(
    contains\_sboxes,
    [\text{items('SBOXES', all)},
    \text{items\_sync('OBJECTS' \text{^} x, 'SBOXES' \text{^} t, 'SBOXES' \text{^} l, all)]}).

ctr\_eval(contains\_sboxes, [\text{logic(contains\_sboxes\_g)}]).

ctr\_logic(
    contains\_sboxes,
    [\text{DIMENSIONS, OIDS}],
    [(\text{origin(O1}, \text{S1}, \text{D}) ---\rightarrow \text{O1} \text{^} x (\text{D}) + \text{S1} \text{^} t (\text{D})]),
    (\text{end(O1}, \text{S1}, \text{D}) ---\rightarrow \text{O1} \text{^} x (\text{D}) + \text{S1} \text{^} t (\text{D}) + \text{S1} \text{^} l (\text{D})]),
    (contains\_sboxes(Dims, O1, S1, O2, S2) ---\rightarrow
    \forall D, Dims,
    \text{origin(O1}, \text{S1}, \text{D}) \text{< origin(O2}, \text{S2}, \text{D}) /\ \text{end(O2}, \text{S2}, \text{D}) \text{< end(O1}, \text{S1}, \text{D})]),
    (contains\_objects(Dims, O1, O2) ---\rightarrow
    \forall S1,
    \text{sboxes([O1} ^ \text{sid})],
    \exists S2,
    \text{sboxes([O2} ^ \text{sid})],
    \text{do\_not\_overlap('SBOXES')})).
contains_sboxes(Dims,01,S1,O2,S2)),
(all_contains(Dims,OIDS)--->
forall(
  01,
  objects(OIDS),
 forall(
    02,
    objects(OIDS),
    01^oid#<02^oid#=>
    contains_objects(Dims,01,02))),
all_contains(DIMENSIONS,OIDS))].

ctr_contractible(contains_sboxes,[],‘OBJECTS’,suffix).

contains_sboxes_g(K,[28807,[]],[28809) :- !,
  check_type(int_gteq(1),K).

contains_sboxes_g(K,[_DIMS,OBJECTS,SBOXES) :-
  length(OBJECTS,O),
  length(SBOXES,S),
  O>0,
  S>0,
  check_type(int_gteq(1),K),
  collection(OBJECTS,[int(1,O),dvar(1,S),col(K,[dvar])]),
  collection(
    SBOXES,
    [int(1,S),col(K,[int]),col(K,[int_gteq(1)])]),
  get_attr1(OBJECTS,OIDS),
  get_attr2(OBJECTS,SIDES),
  get_col_attr3(OBJECTS,1,XS),
  get_attr1(SBOXES,SIDES),
  get_col_attr2(SBOXES,1,TS),
  get_col_attr3(SBOXES,1,TL),
  collection_increasing_seq(OBJECTS,[1]),
  geost1(OIDS,SIDES,XS,Objects),
  geost2(SIDES,TS,TL,Sboxes),
  geost_dims(1,K,DIMENSIONS),
  ctr_logic(contains_sboxes,[DIMENSIONS,OIDS],Rules),
  geost(Objects,Sboxes,[overlap(true)],Rules).
B.82 correspondence

◊ **META-DATA:**

$$\text{ctr}_\text{date}(\text{correspondence}, ['20030820', '20060806']).$$

$$\text{ctr}_\text{origin}(\text{correspondence}, \text{Derived from %c by removing the sorting condition.}, \text{[sort_permutation]}).$$

$$\text{ctr}_\text{arguments}(\text{correspondence}, \text{[FROM'-collection(from-dvar), 'PERMUTATION'-collection(var-dvar), 'TO'-collection(tvar-dvar)]}).$$

$$\text{ctr}_\text{restrictions}(\text{correspondence}, \text{[size('PERMUTATION')=size('FROM'), size('PERMUTATION')=size('TO'), 'PERMUTATION'\text{^var}>=1, 'PERMUTATION'\text{^var}=<size('PERMUTATION'), alldifferent('PERMUTATION'), required('FROM',from), required('PERMUTATION',var), required('TO',tvar)]}).$$

$$\text{ctr}_\text{example}(\text{correspondence}, \text{correspondence}([\text{from-1}, \text{from-9}, \text{from-1}, \text{from-5}, \text{from-2}, \text{from-1}], [\text{var-6}, \text{var-1}, \text{var-3}, \text{var-5}, \text{var-4}, \text{var-2}], [\text{tvar-9}, \text{tvar-1}, \text{tvar-1}, \text{tvar-2}, \text{tvar-5}, \text{tvar-1}])).$$

$$\text{ctr}_\text{typical}()$$
correspondence,
[size('FROM')>1,range('FROM' \textasciitilde from)>1]).

\texttt{ctr\_exchangeable(}
\texttt{correspondence,}
\texttt{[vals(['FROM' \textasciitilde from,'TO' \textasciitilde tvar],int,=\texttt{\textasciitilde},all,dontcare])].}

\texttt{ctr\_derived\_collections(}
\texttt{correspondence,}
\texttt{[col('FROM\_PERMUTATION'-collection(from-dvar,var-dvar),}
\texttt{[item(from-'FROM' \textasciitilde from,var-'PERMUTATION' \textasciitilde var)])]}.}

\texttt{ctr\_graph(}
\texttt{correspondence,}
\texttt{['FROM\_PERMUTATION','TO'],}
\texttt{2,}
\texttt{['PRODUCT'\textasciitilde collection(from\_permutation,to)],}
\texttt{[from\_permutation\~ from=to\~ tvar,}
\texttt{from\_permutation\~ var=to\~ key],}
\texttt{['NARC'=size('PERMUTATION')],}
\texttt{['ACYCLIC','BIPARTITE','NO\_LOOP']}).

\texttt{ctr\_eval(correspondence, [reformulation(correspondence\_r)])}.}

\texttt{correspondence\_r(FROM,PERMUTATION,TO) :-}
\texttt{collection(FROM,[dvar]),}
\texttt{length(FROM,NFROM),}
\texttt{collection(PERMUTATION,[dvar(1,NFROM)]),}
\texttt{length(PERMUTATION,NPERMUTATION),}
\texttt{collection(TO,[dvar]),}
\texttt{length(TO,NTO),}
\texttt{NPERMUTATION=NFROM,}
\texttt{NPERMUTATION=NTO,}
\texttt{get\_attr1(FROM,FROMS),}
\texttt{get\_attr1(PERMUTATION,PERMS),}
\texttt{get\_attr1(TO,TOS),}
\texttt{all\_different(PERMS),}
\texttt{correspondence1(PERMS,FROMS,TOS).}

\texttt{correspondence1([],[],41494).}

\texttt{correspondence1([Pi|R],[Fi|S],TOS) :-}
\texttt{element(Pi,TOS,Fi),}
\texttt{correspondence1(R,S,TOS).}
B.83 count

◊ **META-DATA:**

ctr_date(count, ['20000128', '20030820', '20040530', '20060806', '20100204']).

ctr_origin(count, '\cite{Sicstus95}', []).

ctr_synonyms(count, [occurencemax, occurencemin, occurrence]).

ctr_arguments(count, ['VALUE'-int, 'VARIABLES'-collection(var-dvar), 'RELOP'-atom, 'LIMIT'-dvar]).

ctr_restrictions(count, [required('VARIABLES', var),
  in_list('RELOP', [=, =\=, <, >, =\>, =\<])]).

ctr_example(count, count(5, [[var-4], [var-5], [var-5], [var-4], [var-5]], >\=, 2)).

ctr_typical(count, [size('VARIABLES')\>1,
  range('VARIABLES'\^var)\>1,
  in_list('RELOP', [=, =\<, >, =\>, =\<]),
  'LIMIT'>0,
  'LIMIT'\<size('VARIABLES'))].

ctr_exchangeable(count, [items('VARIABLES', all),
  vals([VARIABLES\^var],
    int(=\=('VALUE')),
    =\=,
    dontcare,
    dontcare)]).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE


ctr_graph(
    count, ['VARIABLES'],
    1, ['SELF']>>collection(variables),
    variables\$var='VALUE',
    ['RELOP'('NARC','LIMIT')],
    []).

ctr_eval(count,[reformulation(count_r),automaton(count_a)]).

ctr_pure_functional_dependency(count,[in_list('RELOP',[=])]).

ctr_contractible(
    count, [in_list('RELOP',[<,=<])], VARIABLES, any).

ctr_extensible(count,[in_list('RELOP',[>=,>])],VARIABLES,any).

ctr_aggregate(
    count, [in_list('RELOP',[<,=<,>=,>])],
    [id,union,id,+]).

count_r(VALUE,VARIABLES,RELOP,LIMIT) :-
    check_type(int,VALUE),
    collection(VARIABLES,[dvar]),
    memberchk(RELOP,[=,\=,\=,<=,>=,>]),
    check_type(dvar,LIMIT),
    length(VARIABLES,NVARIABLES),
    N in 0..NVARIABLES,
    eval(among(N,VARIABLES,[[val-VALUE]])),
    call_term_relop_value(N,RELOP,LIMIT).

count_a(FLAG,VALUE,VARIABLES,RELOP,LIMIT) :-
    check_type(int,VALUE),
    collection(VARIABLES,[dvar]),
    memberchk(RELOP,[=,\=,\=,<=,>=,>]),
    check_type(dvar,LIMIT),
    count_signature(VARIABLES,SIGNATURE,VALUE),
    automaton(
        SIGNATURE,
        _39632,
        SIGNATURE,
[source(s), sink(s)],
[arc(s, 0, s), arc(s, 1, s, [C+1])],
[C],
[0],
[NIN]),
count_relop(RELOP, NIN, LIMIT, FLAG).

count_signature([], [], _38174).

count_signature([[var-VAR]|VARs], [S|Ss], VALUE) :-
  VAR#=VALUE#=S,
count_signature(VARs, Ss, VALUE).
B.84 counts

◊ Meta-Data:

ctr_date(counts,['20030820','20040530','20060806']).

ctr_origin(counts,'Derived from %c.',[count]).

ctr_arguments(
    counts,
    ['VALUES'-collection(val-int),
     'VARIABLES'-collection(var-dvar),
     'RELOP'-atom,
     'LIMIT'-dvar]).

ctr_restrictions(
    counts,
    [required('VALUES',val),
     distinct('VALUES',val),
     required('VARIABLES',var),
     in_list('RELOP',[=,\=,<,\>,\>=,\=<]))).

ctr_example(
    counts,
    counts(
        [[val-1],[val-3],[val-4],[val-9]],
        [[var-4],[var-5],[var-5],[var-4],[var-1],[var-5]],
        =, 3)).

ctr_typical(
    counts,
    [size('VALUES')>1,
     size('VARIABLES')>1,
     range('VARIABLES'\var)>1,
     size('VARIABLES')\size('VALUES'),
     in_list('RELOP',[=,\=,<,\>,\>=,\=<]),
     'LIMIT'>0,
     'LIMIT'\size('VARIABLES'))).

ctr_exchangeable(
    counts,
    [items('VALUES',all),
     items('VARIABLES',all),
     vals(
         ['VARIABLES'\var],
         [\VALUES][\VARIABLES']\val,
         [\VALUES][\VARIABLES']\var]).
comp('VALUES'\`val),
   =,
   dontcare,
   dontcare))).

ctr_graph(
   counts,
   ['VARIABLES','VALUES'],
   2,
   ['PRODUCT'\>collection(variables,values)],
   [variables\`var=values\`val],
   ['RELOP'('NARC','LIMIT')],
   ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(counts,[reformulation(counts_r),automaton(counts_a)]).

ctr_pure_functional_dependency(counts,[in_list('RELOP',[=])]).

ctr_contractible(
   counts,
   [in_list('RELOP',[<,<=])],
   VARIABLES,
   any).

ctr_extensible(
   counts,
   [in_list('RELOP',[>=,>])],
   VARIABLES,
   any).

ctr_aggregate(
   counts,
   [in_list('RELOP',[<,=<,>=,>])],
   [union,union,id,+]).

counts_r(VALUES,VARIABLES,RELOP,LIMIT) :-
   collection(VALUES,[int]),
   collection(VARIABLES,[dvar]),
   memberchk(RELOP,=[,=\=,\<,\>=,\>,\<=]),
   check_type(dvar,LIMIT),
   get_attr1(VALUES,VALS),
   all_different(VALS),
   length(VARIABLES,NVARIABLES),
   N in 0..NVARIABLES,
   eval(among(N,VARIABLES,VALUES)),
   call_term_relop_value(N,RELOP,LIMIT).
counts_a(FLAG,VALUES,VARIABLES,RELOP,LIMIT) :-
  collection(VALUES,[int]),
  collection(VARIABLES,[dvar]),
  memberchk(RELOP, [=,\=,<,\>=,>,\=<]),
  check_type(dvar,LIMIT),
  get_attr1(VALUES,LIST_VALUES),
  all_different(LIST_VALUES),
  list_to_fdset(LIST_VALUES,SET_OF_VALUES),
  counts_signature(VARIABLES,SIGNATURE,SET_OF_VALUES),
  automaton(
    SIGNATURE,
    _41912,
    SIGNATURE,
    [source(s),sink(s)],
    [arc(s,0,s),arc(s,1,s,[C+1])],
    [C],
    [0],
    [NIN]),
  count_relop(RELOP,NIN,LIMIT,FLAG).

counts_signature([],[],_39968).

counts_signature([[var-VAR]|VARs],[S|Ss],SET_OF_VALUES) :-
  VAR in_set SET_OF_VALUES\=\=S,
  counts_signature(VARs,Ss,SET_OF_VALUES).
B.85 coveredby_sboxes

◊ Meta-Data:

ctr_date(coveredby_sboxes, ['20070622','20090725']).

ctr_origin(
    coveredby_sboxes,
    Geometry, derived from \cite{RandellCuiCohn92}, [1]).

ctr_synonyms(coveredby_sboxes, [coveredby]).

ctr_types(
    coveredby_sboxes,
    ['VARIABLES'-collection(v-dvar),
     'INTEGERS'-collection(v-int),
     'POSITIVES'-collection(v-int)]).

ctr_arguments(
    coveredby_sboxes,
    ['K'-int,
     'DIMS'-sint,
     'OBJECTS'-collection(oid-int,sid-int,x-'VARIABLES'),
     'SBOXES'-collection(sid-int,t-'INTEGERS',l-'POSITIVES')]).

ctr_restrictions(
    coveredby_sboxes,
    [size('VARIABLES')>=1,
    size('INTEGERS')>=1,
    size('POSITIVES')>=1,
    required('VARIABLES',v),
    size('VARIABLES')='K',
    required('INTEGERS',v),
    size('INTEGERS')='K',
    required('POSITIVES',v),
    size('POSITIVES')='K',
    'POSITIVES'\^v>0,
    'K'>0,
    'DIMS'>=0,
    'DIMS'< 'K',
    increasing_seq('OBJECTS',[oid]),
    required('OBJECTS',[oid,sid,x]),
    'OBJECTS'\^oid>=1,
    'OBJECTS'\^oid=<size('OBJECTS'),
    'OBJECTS'\^sid>=1,
'OBJECTS'\^\text{sid}=<\text{size}('SBOXES'),
\text{required}('SBOXES', [\text{sid}, t, 1]),
\text{size}('SBOXES')\geq1,
'SBOXES'\^\text{sid}=1,
'SBOXES'\^\text{sid}<\text{size}('SBOXES'),
do_{\text{not\_overlap}}('SBOXES')).

ctr\_example(
  coveredby\_sboxes,
  coveredby\_sboxes(2,
    [0,1],
    [[oid-1, sid-4, x-[[v-2], [v-3]]],
    [oid-2, sid-2, x-[[v-2], [v-2]]],
    [oid-3, sid-1, x-[[v-1], [v-1]]],
    [sid-1, t-[[v-0], [v-0]], l-[[v-3], [v-3]]],
    [sid-1, t-[[v-3], [v-0]], l-[[v-2], [v-2]]],
    [sid-2, t-[[v-0], [v-0]], l-[[v-2], [v-2]]],
    [sid-2, t-[[v-2], [v-0]], l-[[v-1], [v-1]]],
    [sid-3, t-[[v-0], [v-0]], l-[[v-2], [v-2]]],
    [sid-3, t-[[v-2], [v-1]], l-[[v-1], [v-1]]],
    [sid-4, t-[[v-0], [v-0]], l-[[v-1], [v-1]]])).

ctr\_typical(coveredby\_sboxes, [\text{size}('OBJECTS')\geq1]).

ctr\_exchangeable(
  coveredby\_sboxes,
  [\text{items}('SBOXES', all)],
  \text{items\_sync}('OBJECTS'\^\text{x}, 'SBOXES'\^\text{t}, 'SBOXES'\^\text{l}, all))).

ctr\_eval(coveredby\_sboxes, [\text{logic}(coveredby\_sboxes\_g)]).

ctr\_logic(
  coveredby\_sboxes,
  [\text{DIMENSIONS}, \text{OIDS}],
  [(\text{origin}(O1, S1, D)\rightarrow O1\^\text{x}(D)+S1\^\text{t}(D)),
   (\text{end}(O1, S1, D)\rightarrow O1\^\text{x}(D)+S1\^\text{t}(D)+S1\^\text{l}(D)),
   (\text{coveredby\_sboxes}(\text{Dims}, O1, S1, O2, S2)\rightarrow
    \text{forall}(D, \text{Dims}, \text{origin}(O2, S2, D)\#<\text{origin}(O1, S1, D)\#
     \text{end}(O1, S1, D)\#<\text{end}(O2, S2, D))\#/
    \text{exists}(D, \text{Dims}),
  )].
origin(O2,S2,D)\=origin(O1,S1,D)\/
end(O1,S1,D)\=end(O2,S2,D)),
(coveredby_objects(Dims,O1,O2)--->
forall(
  S1,
sboxes([O1\^sid]),
exists(
  S2,
sboxes([O2\^sid]),
  coveredby_sboxes(Dims,O1,S1,O2,S2)))),
(all_coveredby(Dims,OIDS)--->
forall(
  O1,
objects(OIDS),
forall(
  O2,
objects(OIDS),
  O1\^oid#<O2\^oid#=>
  coveredby_objects(Dims,O1,O2)))),
all_coveredby(DIMENSIONS,OIDS)).

coveredby_sboxes_g(K,_31050,[],_31052) :-
  !,
  check_type(int_gteq(1),K).

coveredby_sboxes_g(K,_DIMS,OBJECTS,SBOXES) :-
  length(OBJECTS,O),
  length(SBOXES,S),
  O>0,
  S>0,
  check_type(int_gteq(1),K),
  collection(OBJECTS,[int(1,O),dvar(1,S),col(K,[dvar])]),
  collection(
    SBOXES,
    [int(1,S),col(K,[int]),col(K,[int_gteq(1)])]),
  get_attr1(OBJECTS,OIDS),
  get_attr2(OBJECTS,SIDS),
  get_col_attr3(OBJECTS,1,XS),
  get_attr1(SBOXES,SIDES),
  get_col_attr2(SBOXES,1,TS),
  get_col_attr3(SBOXES,1,TL),
  collection_increasing_seq(OBJECTS,[1]),
  geost1(OIDS,SIDS,XS,Objects),
  geost2(SIDES,TS,TL,Sboxes),
  geost_dims(1,K,DIMENSIONS),
  ctr_logic(coveredby_sboxes,[DIMENSIONS,OIDS],Rules),
geost(Objects,Sboxes,[overlap(true)],Rules).
B.86 covers_sboxes

◊ Meta-Data:

ctr_date(covers_sboxes, ['20070622', '20090725']).

ctr_origin(covers_sboxes, Geometry, derived from \cite{RandellCuiCohn92}, [1]).

ctr_synonyms(covers_sboxes, [covers]).

ctr_types(covers_sboxes, ['VARIABLES'-collection(v-dvar), 'INTEGERS'-collection(v-int), 'POSITIVES'-collection(v-int)]).

ctr_arguments(covers_sboxes, ['K'-int, 'DIMS'-sint, 'OBJECTS'-collection(oid-int, sid-int, x-'VARIABLES'), 'SBOXES'-collection(sid-int, t-'INTEGERS', l-'POSITIVES')]).

ctr_restrictions(covers_sboxes, [size('VARIABLES')>=1, size('INTEGERS')>=1, size('POSITIVES')>=1, required('VARIABLES', v), size('VARIABLES')='K', required('INTEGERS', v), size('INTEGERS')='K', required('POSITIVES', v), size('POSITIVES')='K', 'POSITIVES' v>0, 'K'>0, 'DIMS'>=0, 'DIMS'<'K', increasing_seq('OBJECTS', [oid]), required('OBJECTS', [oid, sid, x]), 'OBJECTS' oid>=1, 'OBJECTS' oid<size('OBJECTS'), 'OBJECTS' sid>=1,}
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

`'OBJECTS'\text{\`sid}=<\text{size('SBOXES')},
size('SBOXES')\geq 1,
\text{required('SBOXES',}[\text{sid},t,l]),
'SBOXES'\text{\`sid}=1,
'SBOXES'\text{\`sid}=<\text{size('SBOXES')},
do\_\text{not}\_\text{overlap('SBOXES')}\}).

\text{ctr\_example}\(
  \text{covers\_sboxes},
  \text{covers\_sboxes}(\n    2,
    \{0,1\},
    [[[oid-1, sid-1, x-[[v-1],[v-1]]],
      [oid-2, sid-2, x-[[v-2],[v-2]]],
      [oid-3, sid-4, x-[[v-2],[v-3]]]],
    [[sid-1, t-[[v-0],[v-0]], l-[[v-3],[v-3]]],
    [sid-1, t-[[v-3],[v-0]], l-[[v-2],[v-2]]],
    [sid-2, t-[[v-0],[v-0]], l-[[v-2],[v-2]]],
    [sid-2, t-[[v-2],[v-0]], l-[[v-1],[v-1]]],
    [sid-3, t-[[v-0],[v-0]], l-[[v-2],[v-2]]],
    [sid-3, t-[[v-2],[v-1]], l-[[v-1],[v-1]]],
    [sid-4, t-[[v-0],[v-0]], l-[[v-1],[v-1]]]))).

\text{ctr\_typical}(\text{covers\_sboxes}, [\text{size('OBJECTS')}\geq 1]).

\text{ctr\_exchangeable}\(
  \text{covers\_sboxes},
  [\text{items('SBOXES',all)},
   \text{items\_sync('OBJECTS'\text{\`x},'SBOXES'\text{\`t},'SBOXES'\text{\`l},all)])}.

\text{ctr\_eval}(\text{covers\_sboxes}, [\text{logic(covers\_sboxes\_g)}]).

\text{ctr\_logic}\(
  \text{covers\_sboxes},
  [\text{DIMENSIONS,OIDS}],
  [(\text{origin(O1,S1,D)}\rightarrow\text{O1}\text{\`x(D)}+\text{S1}\text{\`t(D))},
    (\text{end(O1,S1,D)}\rightarrow\text{O1}\text{\`x(D)}+\text{S1}\text{\`t(D)}+\text{S1}\text{\`l(D))},
    (\text{covers\_sboxes(Dims,O1,S1,O2,S2)}\rightarrow
      \forall D, Dims,
      \text{origin(O1,S1,D)}\neq\text{origin(O2,S2,D)}\}/\&
      \text{end(O2,S2,D)}\neq\text{end(O1,S1,D)}\}/\&
      \exists D, Dims,
origin(O1,S1,D)#=origin(O2,S2,D)#\end{align*}
end(O1,S1,D)#=end(O2,S2,D)),
(covers_objects(Dims,O1,O2)--->
forall(
  S2,
  sboxes([O2^sid]),
  exists(
    S1,
    sboxes([O1^sid]),
    covers_sboxes(Dims,O1,S1,O2,S2))),
(all_covers(Dims,OIDS)--->
forall(
  O1,
  objects(OIDS),
  forall(
    O2,
    objects(OIDS),
    O1^oid#<O2^oid#=>covers_objects(Dims,O1,O2)))).

ctr_contractible(covers_sboxes,\[\],’OBJECTS’,suffix).

covers_sboxes_g(K,_31285,\[\],_31287) :-
  !,
  check_type(int_gteq(1),K).

covers_sboxes_g(K,_DIMS,OBJECTS,SBOXES) :-
  length(OBJECTS,O),
  length(SBOXES,S),
  O>0,
  S>0,
  check_type(int_gteq(1),K),
  collection(OBJECTS,[int(1,O),dvar(1,S),col(K,[dvar])]),
  collection(  
    SBOXES,  
    [int(1,S),col(K,[int]),col(K,[int_gteq(1)])]),
  get_attr1(OBJECTS,OIDS),
  get_attr2(OBJECTS,SIDS),
  get_col_attr3(OBJECTS,1,XS),
  get_attr1(SBOXES,SIDES),
  get_col_attr2(SBOXES,1,TS),
  get_col_attr3(SBOXES,1,TL),
  collection_increasing_seq(OBJECTS,[1]),
  geost1(OIDS,SIDS,XS,Objects),
  geost2(SIDES,TS,TL,Sboxes),
  geost_dims(1,K,DIMENSIONS),
ctr_logic(covers_sboxes,[DIMENSIONS,OIDS],Rules),
geost(Objects,Sboxes,[overlap(true)],Rules).
B.87 crossing

◊ **META-DATA:**

ctr_date(crossing,['20000128','20030820','20060806']).

ctr_origin(
crossing,
Inspired by \cite{CormenLeisersonRivest90},,
[[]].
)

ctr_arguments(
crossing,
['NCROSS'-dvar,
'SEGMENTS'-collection(ox-dvar,oy-dvar,ex-dvar,ey-dvar)]).

ctr_restrictions(
crossing,
['NCROSS']>=0,
NCROSS=<
(size('SEGMENTS')\*size('SEGMENTS')-size('SEGMENTS'))/2,
required('SEGMENTS',[ox,oy,ex,ey]))).

ctr_example(
crossing,
crossing(
 3,
 [[ox-1,oy-4,ex-9,ey-2],
  [ox-1,oy-1,ex-3,ey-5],
  [ox-3,oy-2,ex-7,ey-4],
  [ox-9,oy-1,ex-9,ey-4]]).
)

ctr_typical(crossing,[size('SEGMENTS')>1]).

ctr_exchangeable(
crossing,
[items('SEGMENTS',all),
 attrs_sync('SEGMENTS',[ox,oy],[ex,ey]),
 translate([‘SEGMENTS’^ox,’SEGMENTS’^ex]),
 translate([‘SEGMENTS’^oy,’SEGMENTS’^ey])].
)

ctr_graph(
crossing,
['SEGMENTS'],
2,
[‘CLIQUE’(<)\>>collection(s1,s2)],
)
\[
\begin{align*}
\max(s_1^{\text{ox}}, s_1^{\text{ex}}) & \geq \min(s_2^{\text{ox}}, s_2^{\text{ex}}), \\
\max(s_2^{\text{ox}}, s_2^{\text{ex}}) & \geq \min(s_1^{\text{ox}}, s_1^{\text{ex}}), \\
\max(s_1^{\text{oy}}, s_1^{\text{ey}}) & \geq \min(s_2^{\text{oy}}, s_2^{\text{ey}}), \\
\max(s_2^{\text{oy}}, s_2^{\text{ey}}) & \geq \min(s_1^{\text{oy}}, s_1^{\text{ey}}), \\
(s_2^{\text{ox}} - s_1^{\text{ex}})(s_1^{\text{ey}} - s_1^{\text{oy}}) - (s_1^{\text{ex}} - s_1^{\text{ox}})(s_2^{\text{oy}} - s_1^{\text{ey}}) & = 0 \\
(s_2^{\text{ex}} - s_1^{\text{ex}})(s_2^{\text{oy}} - s_1^{\text{oy}}) - (s_2^{\text{ox}} - s_1^{\text{ox}})(s_2^{\text{ey}} - s_1^{\text{ey}}) & = 0 \\
\text{sign}\left((s_2^{\text{ox}} - s_1^{\text{ex}})(s_1^{\text{ey}} - s_1^{\text{oy}}) - (s_1^{\text{ex}} - s_1^{\text{ox}})(s_2^{\text{oy}} - s_1^{\text{ey}})\right) & = \text{sign}\left((s_2^{\text{ex}} - s_1^{\text{ex}})(s_2^{\text{oy}} - s_1^{\text{oy}}) - (s_2^{\text{ox}} - s_1^{\text{ox}})(s_2^{\text{ey}} - s_1^{\text{ey}})\right),
\end{align*}
\]

\[
\begin{align*}
\text{NARC} & = \text{NCROSS}, \\
\text{ACYCLIC} & = \text{NO_LOOP}.
\end{align*}
\]

\text{ctr}_\text{pure_functional_dependency}(\text{crossing}, []). \text{ctr}_\text{functional_dependency}(\text{crossing}, 1, [2]).
B.88 cumulative

◊ Meta-Data:

ctr_date(cumulative,
        ['20000128','20030820','20040530','20060806','20090923']).

ctr_origin(cumulative,\cite{AggounBeldiceanu93},[]).

ctr_synonyms(cumulative,[cumulative_max]).

ctr_arguments(cumulative,
              [TASKS-
               collection(
                   origin-dvar,
                   duration-dvar,
                   end-dvar,
                   height-dvar),
               'LIMIT'-int]).

ctr_restrictions(cumulative,
                 [require_at_least(2,'TASKS',[origin,duration,end]),
                  required('TASKS',height),
                  'TASKS'\duration>=0,
                  'TASKS'\origin=<'TASKS'\end,
                  'TASKS'\height>=0,
                  'LIMIT'>=0]).

ctr_example(cumulative,
            cumulative(%
               [[origin-1,duration-3,end-4,height-1],
               [origin-2,duration-9,end-11,height-2],
               [origin-3,duration-10,end-13,height-1],
               [origin-6,duration-6,end-12,height-1],
               [origin-7,duration-2,end-9,height-3]],
               8)).

ctr_typical(cumulative,
            [size('TASKS')>1,
             range('TASKS'\origin)>1,
             range('TASKS'\duration)>1,
range('TASKS'~end)>1,
range('TASKS'~height)>1,
'TASKS'~duration>0,
'TASKS'~height>0,
'LIMIT'<sum('TASKS'~height)).

ctr_exchangeable(
cumulative,
[items('TASKS',all),
 vals([['TASKS'~duration],int(>=0)),>,dontcare,dontcare),
 vals([['TASKS'~height],int(>=0)),>,dontcare,dontcare),
 translate([['TASKS'~origin,'TASKS'~end]),
 vals([['LIMIT'],int,<,dontcare,dontcare])).

ctr_graph(
cumulative,
['TASKS'],
1,
['SELF'>>collection(tasks)],
[tasks~origin+tasks~duration=tasks~end],
['NARC'=size('TASKS')],
[]).

ctr_graph(
cumulative,
['TASKS','TASKS'],
2,
['PRODUCT'>>collection(tasks1,tasks2)],
[tasks1~duration>0,
tasks2~origin=<tasks1~origin,
tasks1~origin<tasks2~end],
[],
['ACYCLIC','BIPARTITE','NO_LOOP'],
[SUCC>>
 [source,
  variables-
  col('VARIABLES'-collection(var-dvar),
    [item(var-'TASKS'~height)])]],
 [sum_ctr(variables,=<,'LIMIT')]).

ctr_eval(cumulative,[builtin(cumulative_b)]).

ctr_contractible(cumulative,[],'TASKS',any).

cumulative_b(TASKS,LIMIT) :-
collection(TASKS,[dvar,dvar_gteq(0),dvar,dvar_gteq(0)]),
integer(LIMIT),
LIMIT>=0,
get_attr1(TASKS,ORIGINS),
get_attr2(TASKS,DURATIONS),
get_attr3(TASKS,ENDS),
get_attr4(TASKS,HEIGHTS),
gen_cum_tasks(ORIGINS,DURATIONS,ENDS,HEIGHTS,1,Tasks),
cumulative(Tasks,[limit(LIMIT)]).
B.89 cumulative_convex

Meta-Data:

\[
\text{ctr\_date}(\text{cumulative\_convex}, ['20050817', '20060807']).
\]

\[
\text{ctr\_origin}(\text{cumulative\_convex}, 'Derived from \%c', [\text{cumulative}]).
\]

\[
\text{ctr\_types}(\text{cumulative\_convex}, ['\text{POINTS}'-\text{collection}(\text{var-dvar})]).
\]

\[
\text{ctr\_arguments}(
\text{cumulative\_convex},
[\text{\text{'TASKS'}-\text{collection}(\text{points-'POINTS'}, \text{height-dvar}),}
\text{'LIMIT'-int}]).
\]

\[
\text{ctr\_restrictions}(
\text{cumulative\_convex},
[\text{required('POINTS', var)},
\text{size('POINTS')}>0,\text{required('TASKS', [points, height])},
\text{'TASKS'\text{\textasciicircum}height}\geq 0,\text{'LIMIT'}\geq 0]).
\]

\[
\text{ctr\_example}(
\text{cumulative\_convex},
\text{cumulative\_convex}(
[\text{points-[[var-2], [var-1], [var-5]], height-1},
[\text{points-[[var-4], [var-5], [var-7]], height-2},
[\text{points-}
[\text{[[var-14], [var-13], [var-9], [var-11], [var-10]],}
\text{height-2}]},
3)).
\]

\[
\text{ctr\_typical}(
\text{cumulative\_convex},
[\text{size('TASKS')}>1,\text{'TASKS'\text{\textasciicircum}height}>0,\text{'LIMIT'}<\text{sum('TASKS'\text{\textasciicircum}height)}]).
\]

\[
\text{ctr\_exchangeable}(
\text{cumulative\_convex},
[\text{items('TASKS', all)},
\text{items('TASKS'\text{\textasciicircum}points, all)},
\text{vals(['TASKS'\text{\textasciicircum}height], int(\geq (0)), >, dontcare, dontcare)},
\text{vals(['LIMIT'], int,<, dontcare, dontcare))}].
\]
ctr_derived_collections(
    cumulative_convex,
    [col('INSTANTS'-collection(instant-dvar),
        [item(instant-'TASKS'\^points\^var)]))].

ctr_graph(
    cumulative_convex,
    ['TASKS'],
    1,
    ['SELF'\>collection(tasks)],
    [alldifferent(tasks\^points)],
    ['NARC'=size('TASKS')],
    []).

ctr_graph(
    cumulative_convex,
    ['INSTANTS', 'TASKS'],
    2,
    ['PRODUCT'\>collection(instants,tasks)],
    [between_min_max(instants\^instant,tasks\^points)],
    [],
    ['ACYCLIC', 'BIPARTITE', 'NO_LOOP'],
    [SUCC\>
        [source,
            variables-
            col('VARIABLES'-collection(var-dvar),
                [item(var-'TASKS'\^height)]),
            [sum_ctr(variables,=<,'LIMIT')]].

ctr_contractible(cumulative_convex, [], 'TASKS', any).
B.90  cumulative_product

◊ **META-DATA:**

ctr_date(cumulative_product, ['20030820', '20060807', '20081227']).

ctr_origin(cumulative_product, 'Derived from %c.', [cumulative]).

ctr_arguments(cumulative_product, [TASKS-
  collection(
    origin-dvar,
    duration-dvar,
    end-dvar,
    height-dvar),
  'LIMIT'-int]).

ctr_restrictions(cumulative_product, [require_at_least(2, 'TASKS', [origin, duration, end]),
  required('TASKS', height),
  'TASKS'\duration>=0,
  'TASKS'\origin='TASKS'\end,
  'TASKS'\height>=1,
  'LIMIT'>=0]).

ctr_example(cumulative_product, cumulative_product(
  [[origin-1,duration-3,end-4,height-1],
  [origin-2,duration-9,end-11,height-2],
  [origin-3,duration-10,end-13,height-1],
  [origin-6,duration-6,end-12,height-1],
  [origin-7,duration-2,end-9,height-3]],
  6)).

ctr_typical(cumulative_product, [size('TASKS')>1,
  range('TASKS'\origin)>1,
  range('TASKS'\duration)>1,
  range('TASKS'\end)>1,
  range('TASKS'\height)>1,
  'TASKS'\duration>0,
  'LIMIT'<prod('TASKS'\height)]).
ctr_exchangeable(
cumulative_product,
[items('TASKS', all),
 vals(['TASKS' \^ height], int \geq (0)), >, dontcare, dontcare),
 translate(['TASKS' \^ origin, 'TASKS' \^ end]),
 vals(['LIMIT'], int, <, dontcare, dontcare)).

ctr_graph(
cumulative_product,
['TASKS'],
1,
['SELF' \>> collection(tasks)],
 [tasks \^ origin + tasks \^ duration = tasks \^ end],
 ['NARC' = size('TASKS')],
 []).

ctr_graph(
cumulative_product,
['TASKS', 'TASKS'],
2,
['PRODUCT' \>> collection(tasks1, tasks2)],
 [tasks1 \^ duration > 0, tasks2 \^ origin \< tasks1 \^ origin, tasks1 \^ origin \< tasks2 \^ end],
 [],
 ['ACYCLIC', 'BIPARTITE', 'NO_LOOP'],
 [SUCC >>
  source, variables-
  col('VARIABLES' - collection(var-dvar),
   [item(var- 'ITEMS' \^ height)]),]
 [product_ctr(variables, =<, 'LIMIT')]).

ctr_eval(
cumulative_product,
 [reformulation(cumulative_product_r)]).

ctr_contractible(cumulative_product, [], 'TASKS', any).

cumulative_product_r(TASKS, LIMIT) :-
  integer(LIMIT),
  LIMIT \geq 1,
  collection(
    TASKS,
    [dvar, dvar \geq (0), dvar, dvar(1, LIMIT)]),
  ...,
get_attr1(TASKS, ORIGINS),
geat_attr2(TASKS, DURATIONS),
get_attr3(TASKS, ENDS),
get_attr4(TASKS, HEIGHTS),
ori_dur_end(ORIGINS, DURATIONS, ENDS),
cumulative_product1(
    ORIGINS,
    ENDS,
    HEIGHTS,
    1,
    ORIGINS,
    ENDS,
    HEIGHTS,
    LIMIT).

cumulative_product1(
    [],
    [],
    [],
    _45906,
    _45952,
    _45998,
    _46044,
    _46090).

cumulative_product1(
    [Oi|RO],
    [Ei|RE],
    [Hi|RH],
    I,
    ORIGINS,
    ENDS,
    HEIGHTS,
    LIMIT) :-
cumulative_product2(
    ORIGINS,
    ENDS,
    HEIGHTS,
    1,
    I,
    Oi,
    Ei,
    Hi,
    PRODi),
call(PRODi#=<LIMIT),
I1 is I+1,
cumulative_product1(
    RO,
    RE,
    RH,
    I1,
    ORIGINS,
    ENDS,
    HEIGHTS,
    LIMIT).

cumulative_product2(
    [],
    [],
    [],
    _45909,
    _45955,
    _46001,
    _46047,
    _46093,
    1).

cumulative_product2(
    [_45545|RO],
    [_45549|RE],
    [_45553|RH],
    J,
    I,
    Oi,
    Ei,
    Hi,
    Hi*R) :-
    I=J,
    !,
    J1 is J+1,
    cumulative_product2(RO,RE,RH,J1,I,Oi,Ei,Hi,R).

cumulative_product2([Oj|RO],[Ej|RE],[Hj|RH],J,I,Oi,Ei,Hi,Hij*R) :-
    I=\=J,
    Hij in 1..Hj,
    Oj=<Oi#/\Ej#/Oj#/\Hij#=Hj#/ /
    (Oj#/Oi#/\Ej#=<Oi)#/\Hij#=1,
    J1 is J+1,
    cumulative_product2(RO,RE,RH,J1,I,Oi,Ei,Hi,R).
B.91  \textbf{cumulative\_two\_d}

\textbf{Meta-Data:}

\texttt{ctr\_predefined(cumulative\_two\_d)}.

\texttt{ctr\_date(cumulative\_two\_d,['20000128','20030820','20060807'])}.

\texttt{ctr\_origin(cumulative\_two\_d,Inspired by %c and %c.,[cumulative,diffn])}.

\texttt{ctr\_arguments(cumulative\_two\_d,}\[
\texttt{[RECTANGLES-collection(start1-dvar,}
\texttt{size1-dvar,}
\texttt{last1-dvar,}
\texttt{start2-dvar,}
\texttt{size2-dvar,}
\texttt{last2-dvar,}
\texttt{height-dvar),}
\texttt{'LIMIT'-int]}\].

\texttt{ctr\_restrictions(cumulative\_two\_d,}\[
\texttt{require\_at\_least(2,'RECTANGLES',[start1,size1,last1]),}
\texttt{require\_at\_least(2,'RECTANGLES',[start2,size2,last2]),}
\texttt{required('RECTANGLES',height),}
\texttt{'RECTANGLES'\^{size1}>=0,}
\texttt{'RECTANGLES'\^{size2}>=0,}
\texttt{'RECTANGLES'\^{height}>=0,}
\texttt{'LIMIT')>=0}\].

\texttt{ctr\_example(cumulative\_two\_d,}\[
\texttt{cumulative\_two\_d([[start1-1,}
\texttt{size1-4,}
\texttt{last1-4,}
\texttt{start2-3,}
\texttt{size2-3,}
\texttt{last2-5,}
\texttt{height-4]}]}\].
[(start1-3,  
  size1-2,  
  last1-4,  
  start2-1,  
  size2-2,  
  last2-2,  
  height-2],
[(start1-1,  
  size1-2,  
  last1-2,  
  start2-1,  
  size2-2,  
  last2-2,  
  height-3],
[(start1-4,  
  size1-1,  
  last1-4,  
  start2-1,  
  size2-1,  
  last2-1,  
  height-1)],  
4)).

ctr_typical(
  cumulative_two_d,
  [size(\text{\textsc{rectangles}})\geq 1,
   \text{\textsc{rectangles}}^\text{size1}\geq 0,
   \text{\textsc{rectangles}}^\text{size2}\geq 0,
   \text{\textsc{rectangles}}^\text{height}\geq 0,
   \text{\textsc{limit}}<\text{sum}(\text{\textsc{rectangles}}^\text{height})].

ctr_exchangeable(
  cumulative_two_d,
  [items(\text{\textsc{rectangles}},\text{all}),
   attrs.sync(
     \text{\textsc{rectangles}},
     [[\text{\textsc{start1}},\text{\textsc{start2}}],
      [\text{\textsc{size1}},\text{\textsc{size2}}],
      [\text{\textsc{last1}},\text{\textsc{last2}}],
      [\text{\textsc{height}}]])],
   vals(
     \text{\textsc{rectangles}}^\text{height},
     \text{int}(\geq(0)),
     >,  
     \text{\textit{dontcare}},
     \text{\textit{dontcare}}),
translate(['RECTANGLES'\textsuperscript{start1}, 'RECTANGLES'\textsuperscript{last1}]),
translate(['RECTANGLES'\textsuperscript{start2}, 'RECTANGLES'\textsuperscript{last2}]),
vals(['LIMIT'], \text{int}, <, \text{dontcare}, \text{dontcare})).

\text{ctr\_contractible}(\text{cumulative\_two\_d}, [], 'RECTANGLES', 'any').
B.92 cumulative_with_level_of_priority

◊ Meta-Data:

ctr_date(
    cumulative_with_level_of_priority,
    [’20040530’, ’20060807’]).

ctr_origin(cumulative_with_level_of_priority,’H. Simonis’,[]).

ctr_arguments(
    cumulative_with_level_of_priority,
    [TASKS-
        collection(
            priority-int,
            origin-dvar,
            duration-dvar,
            end-dvar,
            height-dvar),
        ’PRIORITIES’-collection(id-int,capacity-int)])

ctr_restrictions(
    cumulative_with_level_of_priority,
    [required(’TASKS’, [priority, height]),
    require_at_least(2, ’TASKS’, [origin, duration, end]),
    ’TASKS’^priority>=1,
    ’TASKS’^priority=<size(’PRIORITIES’),
    ’TASKS’^duration>=0,
    ’TASKS’^origin=<’TASKS’^end,
    ’TASKS’^height>=0,
    required(’PRIORITIES’, [id, capacity]),
    ’PRIORITIES’^id>=1,
    ’PRIORITIES’^id=<size(’PRIORITIES’),
    increasing_seq(’PRIORITIES’, id),
    increasing_seq(’PRIORITIES’, capacity)])

ctr_example(
    cumulative_with_level_of_priority,
    cumulative_with_level_of_priority(
        [[priority-1, origin-1, duration-2, end-3, height-1],
        [priority-1, origin-2, duration-3, end-5, height-1],
        [priority-1, origin-5, duration-2, end-7, height-2],
        [priority-2, origin-3, duration-2, end-5, height-2],
        [priority-2, origin-6, duration-3, end-9, height-1]],
        [[id-1, capacity-2], [id-2, capacity-3]])).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

ctr_typical(
    cumulative_with_level_of_priority,
    [size('TASKS') > 1,
     range('TASKS' \^ priority) > 1,
     range('TASKS' \^ origin) > 1,
     range('TASKS' \^ duration) > 1,
     range('TASKS' \^ end) > 1,
     range('TASKS' \^ height) > 1,
     'TASKS' \^ duration > 0,
     'TASKS' \^ height > 0,
     size('PRIORITIES') > 1,
     'PRIORITIES' \^ capacity > 0,
     'PRIORITIES' \^ capacity < sum('TASKS' \^ height),
     size('TASKS') > size('PRIORITIES')]).

ctr_exchangeable(
    cumulative_with_level_of_priority,
    [items('TASKS', all),
     vals(
         ['TASKS' \^ priority],
         int =< (size('PRIORITIES'))),
         <,
         dontcare,
         dontcare),
     vals(['TASKS' \^ height], int = (0), >, dontcare, dontcare),
     translate(['TASKS' \^ origin, 'TASKS' \^ end]),
     vals(['PRIORITIES' \^ capacity], int, <, dontcare, dontcare)).

ctr_derived_collections(
    cumulative_with_level_of_priority,
    [col(TIME_POINTS-
        collection(idp-int, duration-dvar, point-dvar),
        [item(
            idp-'TASKS' \^ priority,
            duration-'TASKS' \^ duration,
            point-'TASKS' \^ origin),
            item(
                idp-'TASKS' \^ priority,
                duration-'TASKS' \^ duration,
                point-'TASKS' \^ end))]).

ctr_graph(
    cumulative_with_level_of_priority,
    ['TASKS'],
    1,
    ['SELF' \> collection(tasks)],
\[\text{tasks}^\text{origin+tasks}^\text{duration=tasks}^\text{end},\]
\['\text{NARC'=size('TASKS')}\],
[]).

ctr\_graph(
 cumulative\_with\_level\_of\_priority,
 ['TIME\_POINTS','TASKS'],
 2,
 foreach(
   PRIORITIES,
   ['PRODUCT'>>collection(time\_points,tasks)]),
 [time\_points\^id='PRIORITIES'\^id,
   time\_points\^id>=tasks\^priority,
   time\_points\^duration>0,
   tasks\^origin=<time\_points\^point,
   time\_points\^point<tasks\^end],
 [],
 ['ACYCLIC','BIPARTITE','NO\_LOOP'],
 [SUCCE>>
   [source,
    variables-
    col('VARIABLES'-collection(var-dvar),
    [item(var-'TASKS'\^height)])],
    [sum\_ctr(variables,=<,'PRIORITIES'\^capacity)])].

ctr\_contractible(
 cumulative\_with\_level\_of\_priority,
 [],
 TASKS,
 any).
B.93  cumulatives

◊ Meta-Data:

ctr_date(
cumulatives,
[‘20000128’,’20030820’,’20040530’,’20060807’]).

ctr_origin(cumulatives,’\cite{BeldiceanuCarlsson02 a’},[]).

ctr_arguments(
cumulatives,
[TASKS-
collection(
  machine-dvar,
  origin-dvar,
  duration-dvar,
  end-dvar,
  height-dvar),
  ’MACHINES’-collection(id-int,capacity-int),
  ’CTR’-atom]).

ctr_restrictions(
cumulatives,
[required(‘TASKS’,[machine,height])],
require_at_least(2,‘TASKS’,[origin,duration,end]),
in_attr(‘TASKS’,machine,’MACHINES’,id),
 ‘TASKS’^duration>=0,
 ‘TASKS’^origin<’TASKS’^end,
size(‘MACHINES’)>0,
required(‘MACHINES’,[id,capacity]),
distinct(‘MACHINES’,id),
in_list(‘CTR’,[=<,>=])).

ctr_example(
cumulatives,
cumulatives(
  [[machine-1,origin-2,duration-2,end-4,height- -2],
    [machine-1,origin-1,duration-4,end-5,height-1],
    [machine-1,origin-4,duration-2,end-6,height- -1],
    [machine-1,origin-2,duration-3,end-5,height-2],
    [machine-1,origin-5,duration-2,end-7,height-2],
    [machine-2,origin-3,duration-2,end-5,height- -1],
    [machine-2,origin-1,duration-4,end-5,height-1]],
  [[id-1,capacity-0],[id-2,capacity-0]],
  >=))).
ctr_typical(
cumulatives,
[size('TASKS')>1,
  range('TASKS'\machine)>1,
  range('TASKS'\origin)>1,
  range('TASKS'\duration)>1,
  range('TASKS'\end)>1,
  range('TASKS'\height)>1,
  'TASKS'\duration>0,
  'TASKS'\height=\=0,
  size('MACHINES')>1,
  'MACHINES'\capacity<sum('TASKS'\height),
  size('TASKS')>size('MACHINES'))).

ctr_exchangeable(
cumulatives,
[items('TASKS',all),
  items('MACHINES',all),
  vals(
    ['TASKS'\machine,'MACHINES'\id],
    int,
    =\=,
    all,
    dontcare))).

ctr_derived_collections(
cumulatives,
[col(TIME_POINTS-
  collection(idm-int,duration-dvar,point-dvar),
  [item(
    idm-'TASKS'\machine,
    duration-'TASKS'\duration,
    point-'TASKS'\origin),
    item(
    idm-'TASKS'\machine,
    duration-'TASKS'\duration,
    point-'TASKS'\end)])]).

ctr_graph(
cumulatives,
['TASKS'],
1,
['SELF'\collection(tasks)],
[tasks\origin+tasks\duration=tasks\end],
['NARC'=size('TASKS')],

...
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

[]).

ctr_graph(
    cumulatives,
    ['TIME_POINTS','TASKS'],
    2,
    foreach(
        MACHINES,
        ['PRODUCT'>>collection(time_points,tasks)],
        [time_points`idm='MACHINES'`id,
            time_points`idm=tasks`machine,
            time_points`duration>0,
            tasks`origin=<time_points`point,
            time_points`point<tasks`end],
        [],
        ['ACYCLIC','BIPARTITE','NO_LOOP'],
        [SUC>>
            [source,
                variables-
                    col('VARIABLES'-collection(var-dvar),
                        [item(var-`TASKS`'height)])],
            [sum_ctr(variables,'CTR','MACHINES'`capacity)])].

ctr_eval(cumulatives,[builtin(cumulatives_b)]).

ctr_contractible(
    cumulatives,
    [in_list('RELOP',[=<]),minval('TASKS`height')>=0],
    TASKS,
    any).

cumulatives_b(TASKS,MACHINES,=<) :-
    !,
    collection(TASKS,[dvar,dvar,dvar_gteq(0),dvar,dvar]),
    get_attr1(TASKS,VMACHINES),
    get_attr2(TASKS,ORIGINS),
    get_attr3(TASKS,DURATIONS),
    get_attr4(TASKS,ENDS),
    get_attr5(TASKS,HEIGHTS),
    collection(MACHINES,[int,int]),
    get_attr1(MACHINES,IDS),
    get_attr2(MACHINES,CAPACITIES),
    all_different(IDS),
    cumulatives1(
        VMACHINES,
        ORIGINS,
DURATIONS,
ENDS,
HEIGHTS,
Tasks),
cumulatives2(IDS,CAPACITIES,Machines),
cumulatives(Tasks,Machines,\{bound(upper)\}).

cumulatives_b(TASKS,MACHINES,\geq\) :-
collection(TASKS,\{dvar,dvar,dvar_{\geq 0},dvar,dvar\}),
get_attr1(TASKS,VMACHINES),
get_attr2(TASKS,ORIGINS),
get_attr3(TASKS,DURATIONS),
get_attr4(TASKS,ENDS),
get_attr5(TASKS,HEIGHTS),
collection(MACHINES,\{int,int\}),
get_attr1(MACHINES,IDS),
get_attr2(MACHINES,CAPACITIES),
all_different(IDS),
cumulatives1(VMACHINES,
ORIGINS,
DURATIONS,
ENDS,
HEIGHTS,
Tasks),
cumulatives2(IDS,CAPACITIES,Machines),
cumulatives(Tasks,Machines,\{bound(upper)\}).

cumulatives1([],[],[],[],[],[]).

cumulatives1([M|RM],
[O|RO],
[D|RD],
[E|RE],
[H|RH],
\{task(O,D,E,H,M)|R\}) :-
cumulatives1(RM,RO,RD,RE,RH,R).

cumulatives2([],[],[]).

cumulatives2([I|RI],[C|RC],[machine(I,C)|R]) :-
cumulatives2(RI,RC,R).
B.94 cutset

◊ Meta-Data:

\[
\begin{align*}
\text{ctr}_\text{date}(\text{cutset}, [\text{"20030820"}, \text{"20040530"}, \text{"20060807"}]). \\
\text{ctr}_\text{origin}(\text{cutset}, \\text{"\cite{PagesLa103}"}, []). \\
\text{ctr}_\text{arguments}(\text{cutset}, \\
[\text{"SIZE\_CUTSET"}-\text{dvar}, \\
\text{"NODES"}-\text{collection(index-int, succ-sint, bool-dvar)}]). \\
\text{ctr}_\text{restrictions}(\text{cutset}, \\
[\text{"SIZE\_CUTSET"}\geq 0, \\
\text{"SIZE\_CUTSET"}\leq \text{size(\text{"NODES"})}, \\
\text{required(\text{"NODES"}, [\text{index, succ, bool}])}, \\
\text{"NODES"}^\text{\text{-index}}\geq 1, \\
\text{"NODES"}^\text{\text{-index}}\leq \text{size(\text{"NODES"})}, \\
\text{distinct(\text{"NODES"}, \text{index})}, \\
\text{"NODES"}^\text{\text{-bool}}\geq 0, \\
\text{"NODES"}^\text{\text{-bool}}\leq 1]). \\
\text{ctr}_\text{example}(\text{cutset}, \\
\text{cutset}(\text{1}, \\
[\text{index-1, succ-{2,3,4}, bool-1}], \\
[\text{index-2, succ-{3}, bool-1}], \\
[\text{index-3, succ-{4}, bool-1}], \\
[\text{index-4, succ-{1}, bool-0}])). \\
\text{ctr}_\text{typical}(\text{cutset}, \\
[\text{"SIZE\_CUTSET"}> 0, \\
\text{"SIZE\_CUTSET"}\leq \text{size(\text{"NODES"}),} \\
\text{\text{size(\text{"NODES"})}}> 1]). \\
\text{ctr}_\text{exchangeable}(\text{cutset}, [\text{\text{items(\text{"NODES"}, all)}]}]). \\
\text{ctr}_\text{graph}(\text{cutset}, \\
[\text{"NODES"}], \\
2, \\
[\text{"CLIQUE"}>>\text{\text{collection(nodes1, nodes2)}}]).
\end{align*}
\]

\[
\begin{align*}
\text{contra}(\text{cutset}, [\text{"20030820"}, \text{"20040530"}, \text{"20060807"}]). \\
\text{contra}_\text{origin}(\text{cutset}, \\text{"\cite{PagesLa103}"}, []). \\
\text{contra}_\text{arguments}(\text{cutset}, \\
[\text{"SIZE\_CUTSET"}-\text{dvar}, \\
\text{"NODES"}-\text{collection(index-int, succ-sint, bool-dvar)}]). \\
\text{contra}_\text{restrictions}(\text{cutset}, \\
[\text{"SIZE\_CUTSET"}\geq 0, \\
\text{"SIZE\_CUTSET"}\leq \text{size(\text{"NODES"})}, \\
\text{required(\text{"NODES"}, [\text{index, succ, bool}])}, \\
\text{"NODES"}^\text{\text{-index}}\geq 1, \\
\text{"NODES"}^\text{\text{-index}}\leq \text{size(\text{"NODES"})}, \\
\text{distinct(\text{"NODES"}, \text{index})}, \\
\text{"NODES"}^\text{\text{-bool}}\geq 0, \\
\text{"NODES"}^\text{\text{-bool}}\leq 1]). \\
\text{contra}_\text{example}(\text{cutset}, \\
\text{cutset}(\text{1}, \\
[\text{index-1, succ-{2,3,4}, bool-1}], \\
[\text{index-2, succ-{3}, bool-1}], \\
[\text{index-3, succ-{4}, bool-1}], \\
[\text{index-4, succ-{1}, bool-0}])). \\
\text{contra}_\text{typical}(\text{cutset}, \\
[\text{"SIZE\_CUTSET"}> 0, \\
\text{"SIZE\_CUTSET"}\leq \text{size(\text{"NODES"}),} \\
\text{\text{size(\text{"NODES"})}}> 1]). \\
\text{contra}_\text{exchangeable}(\text{cutset}, [\text{\text{items(\text{"NODES"}, all)}]}]). \\
\text{contra}_\text{graph}(\text{cutset}, \\
[\text{"NODES"}], \\
2, \\
[\text{"CLIQUE"}>>\text{\text{collection(nodes1, nodes2)}}]).
\end{align*}
\]
[nodes2\`index in_set nodes1\`succ, 
nodes1\`bool=1, 
nodes2\`bool=1], 
['MAX_NSNC'=<1,'NVERTEX'=size('NODES')-'SIZE_CUTSET'], 
['ACYCLIC','NO_LOOP']).
B.95 cycle

◊ Meta-Data:

\begin{verbatim}
ctr_date(cycle,['20000128','20030820','20060807','20111223']).
ctr_origin(cycle,'\cite{BeldiceanuContejean94}',[]).
ctr_arguments(cycle,
               ['NCYCLE'-dvar,'NODES'-collection(index-int,succ-dvar)]).
ctr_restrictions(cycle,
                 ['NCYCLE'\geq 1,
                  'NCYCLE'\leq size('NODES'),
                  required('NODES',[index,succ]),
                  'NODES'\checkmark index\geq 1,
                  'NODES'\checkmark index\leq size('NODES'),
                  distinct('NODES',index),
                  'NODES'\checkmark succ\geq 1,
                  'NODES'\checkmark succ\leq size('NODES')].
ctr_example(cycle,
            cycle(2,
                  [[index-1,succ-2],
                   [index-2,succ-1],
                   [index-3,succ-5],
                   [index-4,succ-3],
                   [index-5,succ-4]])).
ctr_typical(cycle,['NCYCLE'\leq size('NODES'),size('NODES')>2]).
ctr_exchangeable(cycle,[items('NODES',all)]).
ctr_graph(cycle,
          ['NODES'],
          2,
          ['CLIQUE'\gg collection(nodes1,nodes2)],
          [nodes1\checkmark succ=nodes2\checkmark index],
          ['NTREE'=0,'NCC'='NCYCLE'],
          ['ONE_SUCC']).
\end{verbatim}
ctr_eval(cycle, [checker(cycle_c), reformulation(cycle_r)]).

ctr_functional_dependency(cycle, 1, [2]).

ctr_sol(cycle, _A000142, [1, 2, 6, 24, 120, 720, 5040]).

cycle_c(NCYCLE, NODES) :-
    length(NODES, N),
    check_type(dvar(1, N), NCYCLE),
    collection(NODES, [int(1, N), dvar(1, N)]),
    get_attr1(NODES, INDEXES),
    get_attr2(NODES, SUCCS),
    length(Term, N),
    list_to_tree(Term, Tree),
    (for(J, 1, N),
    foreach(X, SUCCS),
    foreach(Free, Term),
    foreach(J, Js), param(Tree) do
    get_label(X, Tree, Free)),
    sort(INDEXES, Js),
    sort(SUCCS, Js),
    sort(Term, Cs),
    length(Cs, NCYCLE).

cycle_r(NCYCLE, NODES) :-
    length(NODES, N),
    check_type(dvar(1, N), NCYCLE),
    collection(NODES, [int(1, N), dvar(1, N)]),
    get_attr1(NODES, IND),
    sort(IND, SIND),
    length(SIND, N),
    get_attr12(NODES, IND_SUCC),
    keysort(IND_SUCC, SIND_SUCC),
    remove_key_from_collection(SIND_SUCC, Succ),
    all_different(Succ),
    (for(I, 1, N),
    foreach(Min, Mins), param(Succ, N) do
    length([I|Ss], N),
    minimum(Min, [I|Ss]),
    (foreach(S2, Ss), fromto(I, S1, S2, _46112), param(Succ) do
    element(S1, Succ, S2))),
    nvalue(NCYCLE, Mins).
B.96  cycle_card_on_path

◊ Meta-Data:

ctr_date(
    cycle_card_on_path,
    ['20000128','20030820','20040530','20060807']).

ctr_origin(cycle_card_on_path,'\index{CHIP|indexused}CHIP',[]).

ctr_arguments(
    cycle_card_on_path,
    ['NCYCLE'-dvar,
     'NODES'-collection(index-int,succ-dvar,colour-dvar),
     'ATLEAST'-int,
     'ATMOST'-int,
     'PATH_LEN'-int,
     'VALUES'-collection(val-int)]).

ctr_restrictions(
    cycle_card_on_path,
    ['NCYCLE'>=1,
     'NCYCLE'=<size('NODES'),
     required('NODES',[index,succ,colour]),
     'NODES' ^index>=1,
     'NODES' ^index=<size('NODES'),
     distinct ('NODES',index),
     'NODES' ^succ>=1,
     'NODES' ^succ=<size('NODES'),
     'ATLEAST'>=0,
     'ATLEAST'=<'PATH_LEN',
     'ATMOST'='ATLEAST',
     'PATH_LEN'>=0,
     size('VALUES')>=1,
     required('VALUES',val),
     distinct ('VALUES',val)]).

ctr_example(
    cycle_card_on_path,
    cycle_card_on_path(2,
    [[index-1,succ-7,colour-2],
     [index-2,succ-4,colour-3],
     [index-3,succ-8,colour-2],
     [index-4,succ-9,colour-1],
     [index-5,succ-1,colour-2],
     ...])
[\[\text{index-6, succ-2, colour-1}\],
[\text{index-7, succ-5, colour-1}\],
[\text{index-8, succ-6, colour-1}\],
[\text{index-9, succ-3, colour-1}\],
1,
2,
3,
[[\text{val-1}]]).

\text{ctr}_\text{typical}(\text{cycle_card_on_path},
[\text{size('NODES')} > 2,
'NCYCLE' < \text{size('NODES')},
'ATLEAST' < 'PATH_LEN',
'ATMOST' > 0,
'PATH_LEN' > 1,
\text{size('NODES')} > \text{size('VALUES')},
'ATLEAST' > 0#/'ATMOST' < 'PATH_LEN']).

\text{ctr}_\text{exchangeable}(\text{cycle_card_on_path},
[\text{items('NODES', all)},
\text{vals}(
[\text{\textquotesingle NODES\textquotesingle}^\text{\textquoteright colour}],
\text{comp('VALUES\textquotesingle}^\text{\textquoteright val}),
=,\text{d dontcare,}
\text{d dontcare),
\text{vals}(['ATLEAST'], \text{int}(\geq (0)), >, \text{d dontcare, dontcare),
\text{vals}(['ATMOST'], \text{int}, <, \text{d dontcare, dontcare),
\text{items('VALUES', all)\text{)}}).

\text{ctr}_\text{graph}(\text{cycle_card_on_path},
['\text{NODES}\text{']},
2,
['\text{CLIQUE'}>>\text{collection(nodes1, nodes2)},
[\text{nodes1\text{\textquoteright succ=}nodes2\text{\textquoteright index)},
['\text{NTREE'}=0, 'NCC'='NCYCLE'],
['\text{ONE_SUCCE'],
['\text{PATH_LENGTH'}('PATH_LEN')>>
[\text{variables-col('\text{VARIABLES'}-\text{collection(var-dvar)},
[\text{item(var-'\text{NODES'}\text{\textquoteright colour})\text{\text{)}}\text{\text{]}},
[\text{among_low_up('ATLEAST', 'ATMOST', variables, 'VALUES')\text{\text{)}}\text{\text{)}\}}.}
B.97 cycle_or_accessibility

◊ Metadate:

ctr_date(  
cycle_or_accessibility,  
[‘20000128’, ‘20030820’, ‘20060807’]).

ctr_origin(  
cycle_or_accessibility,  
Inspired by \cite{LabbeLaporteRodriguezMartin98}.).

ctr_arguments(  
cycle_or_accessibility,  
[‘MAXDIST’-int,  
‘NCYCLE’-dvar,  
‘NODES’-collection(index-int, succ-dvar, x-int, y-int)]).

ctr_restrictions(  
cycle_or_accessibility,  
[‘MAXDIST’>=0,  
‘NCYCLE’>=1,  
‘NCYCLE’=<size(‘NODES’),  
required(‘NODES’, [index, succ, x, y]),  
‘NODES’\_index>=1,  
‘NODES’\_index=<size(‘NODES’),  
distinct(‘NODES’, index),  
‘NODES’\_succ>=0,  
‘NODES’\_succ=<size(‘NODES’),  
‘NODES’\_x>=0,  
‘NODES’\_y>=0]).

ctr_example(  
cycle_or_accessibility,  
cycle_or_accessibility(  
3,  
2,  
[[index-1, succ-6, x-4, y-5],  
[index-2, succ-0, x-9, y-1],  
[index-3, succ-0, x-2, y-4],  
[index-4, succ-1, x-2, y-6],  
[index-5, succ-5, x-7, y-2],  
[index-6, succ-4, x-4, y-7],  
[index-7, succ-0, x-6, y-4]]).)
ctr_typical(
    cycle_or_accessibility,
    ['MAXDIST'>0,'NCYCLE'<size('NODES'),size('NODES')>2]).

ctr_exchangeable(
    cycle_or_accessibility,
    [items('NODES',all),
      attrs_sync('NODES',[[index],[succ],[x,y]]),
      translate(['NODES'\x]),
      translate(['NODES'\y])].

ctr_graph(
    cycle_or_accessibility,
    ['NODES'],
    2,
    ['CLIQUE'\collection(nodes1,nodes2)],
    [nodes1\succ=nodes2\index],
    ['NVERTEX'=0,'NCC'='NCYCLE'],
    []).

ctr_graph(
    cycle_or_accessibility,
    ['NODES'],
    2,
    ['CLIQUE'\collection(nodes1,nodes2)],
    [nodes1\succ=nodes2\index\/
      nodes1\succ=0\/
      nodes2\succ=\=0\/
      abs(nodes1\x-nodes2\x)+abs(nodes1\y-nodes2\y)<='MAXDIST'],
    ['NVERTEX'=size('NODES')],
    [],
    [PRED>>
      [variables-
        col('VARIABLES'\collection(var-dvar),
          [item(var\NODES\succ)],
          destination)],
      [nvalues_except_0(variables,=,1)]).

ctr_functional_dependency(cycle_or_accessibility,2,[3]).
B.98 cycle_resource

♦ Meta-Data:

ctr_date(cycle_resource, [’20030820’, ’20040530’, ’20060807’]).

ctr_origin(cycle_resource, ’\\index{CHIP|indexuse}CHIP’, []).

ctr_arguments(
    cycle_resource,
    [RESOURCE-
        collection(id-int, first_task-dvar, nb_task-dvar),
        ’TASK’-collection(id-int, next_task-dvar, resource-dvar)]).

ctr_restrictions(
    cycle_resource,
    [required(’RESOURCE’, [id, first_task, nb_task]),
        ’RESOURCE’\^id>=1,  
        ’RESOURCE’\^id=<size(’RESOURCE’),
        distinct(’RESOURCE’, id),
        ’RESOURCE’\^first_task>=1,  
        ’RESOURCE’\^first_task=<size(’RESOURCE’)+size(’TASK’),
        ’RESOURCE’\^nb_task>=0,  
        ’RESOURCE’\^nb_task=<size(’TASK’),
        required(’TASK’, [id, next_task, resource]),
        ’TASK’\^id>=size(’RESOURCE’),
        ’TASK’\^id=<size(’RESOURCE’)+size(’TASK’),
        distinct(’TASK’, id),
        ’TASK’\^next_task>=1,  
        ’TASK’\^next_task=<size(’RESOURCE’)+size(’TASK’),
        ’TASK’\^resource>=1,  
        ’TASK’\^resource=<size(’RESOURCE’)]).

ctr_example(
    cycle_resource,
    cycle_resource(
        [[id-1, first_task-5, nb_task-3],
        [id-2, first_task-2, nb_task-0],
        [id-3, first_task-8, nb_task-2]],
        [[id-4, next_task-7, resource-1],
        [id-5, next_task-4, resource-1],
        [id-6, next_task-3, resource-3],
        [id-7, next_task-1, resource-1],
        [id-8, next_task-6, resource-3]])).

ctr_typical{
cycle_resource,
[size('RESOURCE')>1,
 size('TASK')>1,
 size('TASK')>size('RESOURCE')]).

ctr_exchangeable(
cycle_resource,
[items('RESOURCE',all),
 items('TASK',all),
 vals(['RESOURCE'\~id,'TASK'\~resource],int,=\=,all,in))].

ctr_derived_collections(
cycle_resource,
[col('RESOURCE_TASK-
 collection(index-int,succ-dvar,name-dvar),
 [item(
   index='RESOURCE'\~id,
   succ='RESOURCE'\~first_task,
   name='RESOURCE'\~id),
   item(
   index='TASK'\~id,
   succ='TASK'\~next_task,
   name='TASK'\~resource)])]).

ctr_graph(
cycle_resource,
['RESOURCE_TASK'],
2,
['CLIQUE'>>collection(resource_task1,resource_task2)],
[resource_task1\~succ=resource_task2\~index,
 resource_task1\~name=resource_task2\~name],
['NTREE'=0,
 'NCC'=size('RESOURCE'),
 'NVERTEX'=size('RESOURCE')+size('TASK'),
 ['ONE_SUCC']]).

ctr_graph(
cycle_resource,
['RESOURCE_TASK'],
2,
foreach(
 RESOURCE,
 ['CLIQUE'>>collection(resource_task1,resource_task2)],
 [resource_task1\~succ=resource_task2\~index,
 resource_task1\~name=resource_task2\~name,
 resource_task1\~name='RESOURCE'\~id],
 [resource_task2\~name='RESO
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

['NVERTEX'='RESOURCE'\nb_task+1],
[]).
**B.99 cyclic_change**

◇ **META-DATA:**

```prolog
ctr_date(
cyclic_change,
['20000128','20030820','20040530','20060807']).

ctr_origin(cyclic_change,'Derived from %c.',[change]).

ctr_arguments(
cyclic_change,
['NCHANGE'-dvar,
 'CYCLE_LENGTH'-int,
 'VARIABLES'-collection(var-dvar),
 'CTR'-atom]).

ctr_restrictions(
cyclic_change,
['NCHANGE'>=0,
 'NCHANGE'<size('VARIABLES'),
 'CYCLE_LENGTH'>0,
 required('VARIABLES',var),
 'VARIABLES'\^var>=0,
 'VARIABLES'\^var<'CYCLE_LENGTH',
 in_list('CTR',[=,\=,<,>,>=,=]))).

ctr_example(
cyclic_change,
 cyclic_change(2,
 4,
 4,[[var-3],[var-0],[var-2],[var-3],[var-1]],
 4\=)).

ctr_typical(
cyclic_change,
['NCHANGE'>0,
 size('VARIABLES')>1,
 range('VARIABLES'\^var)>1,
 in_list('CTR',[=\=]))).

ctr_exchangeable(cyclic_change,[items('VARIABLES',shift)]).

ctr_graph(
cyclic_change,
 ...)```
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

[‘VARIABLES’],
2,
[‘PATH’>>collection(variables1,variables2)],
[‘CTR’((variables1\^var+1)mod ‘CYCLE_LENGTH’, variables2\^var)],
[‘NARC’=‘NCHANGE’],
[‘ACYCLIC’,'BIPARTITE’,'NO_LOOP’]).

cyclic_change_a(FLAG,NCHANGE,CYCLE_LENGTH,VARIABLES,CTR) :-
  integer(CYCLE_LENGTH),
  CYCLE_LENGTH>0,
  CYCLE_LENGTH_1 is CYCLE_LENGTH-1,
  collection(VARIABLES,[dvar(0,CYCLE_LENGTH_1)]),
  length(VARIABLES,N),
  N_1 is N-1,
  check_type(dvar(0,N_1),NCHANGE),
  memberchk(CTR,[=,\=,\<,\\ge,\>,\=<]),
  cyclic_change_signature(
    VARIABLES,
    SIGNATURE,
    CYCLE_LENGTH,
    CTR),
    automaton(
      SIGNATURE,
      _34102,
      SIGNATURE,
      [source(s),sink(s)],
      [arc(s,0,s),arc(s,1,s,[C+1])],
      [C],
      [0],
      [COUNT]),
      COUNT#=NCHANGE#<=>FLAG.

cyclic_change_signature([],[],_31537,_31538).

cyclic_change_signature([_31542],[],_31540,_31541) :- !.

cyclic_change_signature(
  [[var-VAR1],[var-VAR2]|VARs],
cyclic_change_signature(
    [[var-VAR1], [var-VAR2]|VARs],
    [S|Ss],
    CYCLE_LENGTH,
    =) :-
    !,
    (VAR1+1)mod CYCLE_LENGTH#=VAR2#<=S,
    cyclic_change_signature(
        [[var-VAR2]|VARs],
        Ss,
        CYCLE_LENGTH,
        =).

cyclic_change_signature(
    [[[var-VAR1], [var-VAR2]|VARs],
      [S|Ss],
      CYCLE_LENGTH,
      =\=) :-
    !,
    (VAR1+1)mod CYCLE_LENGTH\=VAR2#<=S,
    cyclic_change_signature(
        [[var-VAR2]|VARs],
        Ss,
        CYCLE_LENGTH,
        =\=).

cyclic_change_signature(
    [[var-VAR1], [var-VAR2]|VARs],
    [S|Ss],
    CYCLE_LENGTH,
    <) :-
    !,
    (VAR1+1)mod CYCLE_LENGTH#<VAR2#<=S,
    cyclic_change_signature(
        [[var-VAR2]|VARs],
        Ss,
        CYCLE_LENGTH,
        <).

cyclic_change_signature(
    [[var-VAR1], [var-VAR2]|VARs],
    [S|Ss],
    CYCLE_LENGTH,
    >=) :-
    !,
    (VAR1+1)mod CYCLE_LENGTH#>=VAR2#<=S,
    cyclic_change_signature(
        [[var-VAR2]|VARs],
        Ss,
\[\text{cyclic\_change\_signature} (\text{[[var-VAR1],[var-VAR2]|VARs], [S|Ss], CYCLE\_LENGTH,} >) :- !, \]
\[\text{(VAR1+1)mod CYCLE\_LENGTH}\#>VAR2\#<=>S,}\]
\[\text{cyclic\_change\_signature} (\text{[[var-VAR2]|VARs], Ss, CYCLE\_LENGTH,} >).\]

\[\text{cyclic\_change\_signature} (\text{[[var-VAR1],[var-VAR2]|VARs], [S|Ss], CYCLE\_LENGTH,} =<) :- !, \]
\[\text{(VAR1+1)mod CYCLE\_LENGTH}\#=<VAR2\#<=>S,}\]
\[\text{cyclic\_change\_signature} (\text{[[var-VAR2]|VARs], Ss, CYCLE\_LENGTH,} =<).\]
B.100  cyclic_change_joker

◊ Meta-Data:

\[
\begin{align*}
\text{ctr}\_date(} & \\
& \text{ cyclic\_change\_joker,} \\
& \text{ ['20000128','20030820','20040530','20060807'].} \\
\end{align*}
\]

\[
\begin{align*}
\text{ctr}\_origin(} & \\
& \text{ cyclic\_change\_joker,} \\
& \text{ Derived from %c.,} \\
& \text{ [cyclic\_change].} \\
\end{align*}
\]

\[
\begin{align*}
\text{ctr}\_arguments(} & \\
& \text{ cyclic\_change\_joker,} \\
& \text{ ['NCHANGE'-dvar,} \\
& \text{'CYCLE\_LENGTH'-int,} \\
& \text{'VARIABLES'-collection(var-dvar),} \\
& \text{'CTR'-atom].} \\
\end{align*}
\]

\[
\begin{align*}
\text{ctr}\_restrictions(} & \\
& \text{ cyclic\_change\_joker,} \\
& \text{ ['NCHANGE']>=0,} \\
& \text{'NCHANGE'<size('VARIABLES'),} \\
& \text{'CYCLE\_LENGTH'>0,} \\
& \text{ required('VARIABLES',var),} \\
& \text{'VARIABLES'~var>=0,} \\
& \text{ in\_list('CTR',[=,\text{\textless},\text{\geq},>,\text{\leq}]).} \\
\end{align*}
\]

\[
\begin{align*}
\text{ctr}\_example(} & \\
& \text{ cyclic\_change\_joker,} \\
& \text{ cyclic\_change\_joker(} \\
& \text{ 2,} \\
& \text{ 4,} \\
& \text{ [['var-3],} \\
& \text{ [var-0],} \\
& \text{ [var-2],} \\
& \text{ [var-4],} \\
& \text{ [var-4],} \\
& \text{ [var-4],} \\
& \text{ [var-3],} \\
& \text{ [var-1],} \\
& \text{ [var-4],} \\
& \text{ =\text{\textless}]).} \\
\end{align*}
\]

\[
\begin{align*}
\text{ctr}\_typical(} & \\
& \text{ cyclic\_change\_joker,} \\
& \text{ cyclic\_change\_joker(} \\
& \text{ 2,} \\
& \text{ 4,} \\
& \text{ [['var-3],} \\
& \text{ [var-0],} \\
& \text{ [var-2],} \\
& \text{ [var-4],} \\
& \text{ [var-4],} \\
& \text{ [var-4],} \\
& \text{ [var-4],} \\
& \text{ [var-3],} \\
& \text{ [var-1],} \\
& \text{ [var-4],} \\
& \text{ =\text{\textless}]).} \\
\end{align*}
\]
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

cyclic_change_joker,
['NCHANGE'>0,
'CYLE_LENGTH'>1,
size('VARIABLES')>1,
range('VARIABLES'\var)>1,
maxval('VARIABLES'\var)>='CYCLE_LENGTH',
in_list('CTR',[\=\=])].

ctr_exchangeable{
cyclic_change_joker,
[items('VARIABLES',shift)].

ctr_graph{
cyclic_change_joker,
['VARIABLES'],
2,
['PATH'>>collection(variables1,variables2)],
['CTR'((variables1\var+1)mod 'CYCLE_LENGTH',
    variables2\var),
    variables1\var<='CYCLE_LENGTH',
    variables2\var<='CYCLE_LENGTH'],
['NARC'='NCHANGE'],
['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval{
cyclic_change_joker,
[automaton(cyclic_change_joker_a)].

ctr_pure_functional_dependency(cyclic_change_joker,[]).

ctr_functional_dependency(cyclic_change_joker,1,[2,3,4]).

cyclic_change_joker_a(FLAG,NCHANGE,CYCLE_LENGTH,VARIABLES,CTR) :-
    integer(CYCLE_LENGTH),
    CYCLE_LENGTH>0,
collection(VARIABLES,[dvar_gteq(0)]),
length(VARIABLES,N),
N_1 is N-1,
check_type(dvar(0,N_1),NCHANGE),
memberchk(CTR,[\=\=\<\>,\>=\<]),
cyclic_change_joker_signature(
    VARIABLES,
    SIGNATURE,
    CYCLE_LENGTH,
    CTR),
automaton(
SIGNATURE,
_37169,
SIGNATURE,
[source(s),sink(s)],
[arc(s,0,s),arc(s,1,s,[C+1])],
[C],
[0],
[COUNT]),
COUNT#=NCHANGE#<=>FLAG.
cyclic_change_joker_signature([],[],_34819,_34820).
cyclic_change_joker_signature([_34824],[],_34822,_34823) :- !.
cyclic_change_joker_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    CYCLE_LENGTH,
    =) :- !,
    (VAR1+1)mod CYCLE_LENGTH#VAR2#/\ 
    VAR1#<CYCLE_LENGTH#/\ 
    VAR2#<CYCLE_LENGTH#<=>S,
    cyclic_change_joker_signature(
        [[var-VAR2]|VARs],
        Ss,
        CYCLE_LENGTH,
        =).
cyclic_change_joker_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    CYCLE_LENGTH,
    =\=) :- !,
    (VAR1+1)mod CYCLE_LENGTH\=VAR2#/\ 
    VAR1#<CYCLE_LENGTH#/\ 
    VAR2#<CYCLE_LENGTH#<=>S,
    cyclic_change_joker_signature(
        [[var-VAR2]|VARs],
        Ss,
        CYCLE_LENGTH,
        =\=).
cyclic_change_joker_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    CYCLE_LENGTH,
    <) :-
    !,
    (VAR1+1) mod CYCLE_LENGTH #<VAR2#/\ 
    VAR1#<CYCLE_LENGTH#/\ 
    VAR2#<CYCLE_LENGTH#<=>
    S, 
    cyclic_change_joker_signature(
        [[var-VAR2]|VARs],
        Ss,
        CYCLE_LENGTH,
        <).

cyclic_change_joker_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    CYCLE_LENGTH,
    >=) :-
    !,
    (VAR1+1) mod CYCLE_LENGTH #=VAR2#/\ 
    VAR1#<CYCLE_LENGTH#/\ 
    VAR2#<CYCLE_LENGTH#<=>
    S, 
    cyclic_change_joker_signature(
        [[var-VAR2]|VARs],
        Ss,
        CYCLE_LENGTH,
        >=).

cyclic_change_joker_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    CYCLE_LENGTH,
    >) :-
    !,
    (VAR1+1) mod CYCLE_LENGTH #>VAR2#/\ 
    VAR1#<CYCLE_LENGTH#/\ 
    VAR2#<CYCLE_LENGTH#<=>
    S, 
    cyclic_change_joker_signature(
        [[var-VAR2]|VARs],
        Ss,
cyclic_change_joker_signature([var-VAR1],[var-VAR2]|VARs],[S|Ss],CYCLE_LENGTH,=<) :-
!,
(VAR1+1)mod CYCLE_LENGTH=<VAR2/\VAR1\<CYCLE_LENGTH/\VAR2<CYCLE_LENGTH<=>
S,
cyclic_change_joker_signature([var-VAR2]|VARs],Ss,CYCLE_LENGTH,=<).
B.101 dag

◊ Meta-Data:

ctr_date(dag,['20061001']).

ctr_origin(dag, '\cite{Dooms06}', []).

ctr_arguments(dag, ['NODES'-collection(index-int, succ-svar)]).

ctr_restrictions(
    dag,
    [required('NODES', [index, succ]),
     'NODES'\^index>=1,
     'NODES'\^index=<size('NODES'),
     distinct('NODES', index),
     'NODES'\^succ>=1,
     'NODES'\^succ=<size('NODES'))].

ctr_example(
    dag,
    dag([[index-1, succ-{2,4}],
         [index-2, succ-{3,4}],
         [index-3, succ-{}],
         [index-4, succ-{}],
         [index-5, succ-{6}],
         [index-6, succ-{}]])).

ctr_typical(dag, [size('NODES')>2]).

ctr_exchangeable(dag, [items('NODES', all)]).

ctr_graph(
    dag,
    ['NODES'],
    1,
    ['SELF'\>collection(nodes)],
    [nodes\key in_set nodes\succ],
    ['NARC'=0],
    []).

ctr_graph(
    dag,
    ['NODES'],
    2,
    ['CLIQUE'\>collection(nodes1, nodes2)],
    [nodes1\key in_set nodes1\succ],
    [nodes2\key in_set nodes2\succ],
    ['NARC'=0],
    []).
[nodes2`index in_set nodes1`succ],
[\'MAX_NSCC\'=<1],
[[]].
B.102 decreasing

◊ **Meta-Data:**

ctr_date(decreasing, ['20040814', '20060808']).

ctr_origin(decreasing, 'Inspired by %c.', [increasing]).

ctr_arguments(decreasing, ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(decreasing, [required('VARIABLES', var)]).

ctr_example(  
decreasing,  
decreasing([[var-8],[var-4],[var-1],[var-1]]).  
)

ctr_typical(  
decreasing,  
[size('VARIABLES')>1, range('VARIABLES'\^var)>1]).  
)

ctr_exchangeable(decreasing, [translate(['VARIABLES'\^var])]).

ctr_graph(  
decreasing,  
['VARIABLES'],  
2,  
['PATH'>>collection(variables1,variables2)],  
[variables1\^var>=variables2\^var],  
['NARC'=size('VARIABLES')-1],  
['ACYCLIC','BIPARTITE','NO_LOOP']].

ctr_eval(  
decreasing,  
[checker(decreasing_c),automaton(decreasing_a)]).

ctr_contractible(decreasing, [], 'VARIABLES', any).

decreasing_c([]) :- !.

decreasing_c(VARIABLES) :-  
collection(VARIABLES, [int]),  
get_attr1(VARIABLES, VARS),  
decreasing_c1(VARS).

decreasing_c1([]) :-
decreasing_c1([X,Y|R]) :-
    X>=Y,
    decreasing_c1([Y|R]).

decreasing_a(1,[]) :-
    !.

decreasing_a(0,[]) :-
    !,
    fail.

decreasing_a(FLAG,VARIABLES) :-
collection(VARIABLES,[dvar]),
decreasing_signature(VARIABLES,SIGNALATURE),
AUTOMATON=automaton(
    SIGNATURE,
    _31636,
    SIGNALATURE,
    [source(s),sink(s)],
    [arc(s,1,s)],
    [],
    [],
    []),
automaton_bool(FLAG,[0,1],AUTOMATON).

decreasing_signature([],[]) :-
    !.

decreasing_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss]) :-
    S in 0..1,
    VAR1#>=VAR2#<=S,
    decreasing_signature([[var-VAR2]|VARs],Ss).
B.103 deepest_valley

Μeta-Data:

ctr_date(deepest_valley, [’20040530’]).

ctr_origin(deepest_valley, ’Derived from %c.’ , [valley]).

ctr_arguments(
    deepest_valley,
    [’DEPTH’-dvar,’VARIABLES’-collection(var-dvar)]).

ctr_restrictions(
    deepest_valley,
    [’DEPTH’>=0,’VARIABLES’ˆvar>=0,required(’VARIABLES’,var)]).

ctr_example(
    deepest_valley,
    deepest_valley(
        2,
        [[var-5],
         [var-3],
         [var-4],
         [var-8],
         [var-8],
         [var-2],
         [var-7],
         [var-1]])).

ctr_typical(
    deepest_valley,
    [’DEPTH’=<maxval(’VARIABLES’ˆvar),
     size(’VARIABLES’) >1,
     range(’VARIABLES’ˆvar)>1]).

ctr_exchangeable(deepest_valley,[items(’VARIABLES’,reverse)]).

ctr_eval(deepest_valley,[automaton(deepest_valley_a)]).

deepest_valley_a(FLAG,DEPTH,VARIABLES) :-
    check_type(dvar_gteq(0),DEPTH),
    collection(VARIABLES,[dvar_gteq(0)]),
    MAXINT=1000000,
    deepest_valley_signature(VARIABLES,SIGNATURE,PAIRS),
    automaton( PAIRS,
VAR1–VAR2,
SIGNATURE,
[source(s), sink(s), sink(u)],
[arc(s,0,s),
 arc(s,1,s),
 arc(s,2,u),
 arc(u,0,s,[min(C,VAR1)]),
 arc(u,1,u),
 arc(u,2,u)],
[C],
[MAXINT],
[COUNT]),
COUNT#=DEPTH#<=>FLAG.

deepest_valley_signature([],[],[]).

deepest_valley_signature([_14415],[],[]) :-
 !.

deepest_valley_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    [VAR1–VAR2|PAIRS]) :-
    S in 0..2,
    VAR1<VAR2<=>S#=0,
    VAR1=VAR2<=>S#=1,
    VAR1>VAR2<=>S#=2,
    deepest_valley_signature([[var-VAR2]|VARs],Ss,PAIRS).
B.104 derangement

◊ **Meta-Data:**

```prolog
ctr_date(
derangement,
['20000128', '20030820', '20040530', '20060808']).
ctr_origin(derangement, 'Derived from %c.', [cycle]).
ctr_arguments(
derangement,
['NODES' - collection(index-int, succ-dvar)]).
ctr_restrictions(
derangement,
[size('NODES') > 1,
 required('NODES', [index, succ]),
 'NODES'^index >= 1,
 'NODES'^index =< size('NODES'),
 distinct('NODES', index),
 'NODES'^succ >= 1,
 'NODES'^succ =< size('NODES')]).
ctr_example(
derangement,
derangement(
[[index-1, succ-2],
[index-2, succ-1],
[index-3, succ-5],
[index-4, succ-3],
[index-5, succ-4]]).
ctr_typical(derangement, [size('NODES') > 2]).
ctr_exchangeable(
derangement,
[items('NODES', all), attr_sync('NODES', '[[index, succ]]')].
ctr_graph(
derangement,
['NODES'],
2,
['CLIQUE' > collection(nodes1, nodes2)],
[nodes1^succ = nodes2^index, nodes1^succ = \= nodes1^index],
['NTREE' = 0],
```
ctr_eval(derangement,[reformulation(derangement_r)]).

derangement_r(NODES) :-
    length(NODES,N),
    collection(NODES,[int(1,N),dvar(1,N)]),
    get_attr1(NODES,INDEXES),
    get_attr2(NODES,SUCCS),
    all_different(INDEXES),
    derangement1(SUCCS,INDEXES),
    all_different(SUCCS).
B.105  differ_from_at_least_k_pos

◊ Meta-Data:

ctr_date(
    differ_from_at_least_k_pos,
    ['20030820','20040530','20060808']).

ctr_origin(
    differ_from_at_least_k_pos,
    Inspired by \cite{Frutos97}.,
    []).

ctr_types(
    differ_from_at_least_k_pos,
    ['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    differ_from_at_least_k_pos,
    ['K'-int,'VECTOR1'-'VECTOR','VECTOR2'-'VECTOR']).

ctr_restrictions(
    differ_from_at_least_k_pos,
    [size('VECTOR')>=1,
     required('VECTOR',var),
     'K'>=0,
     'K'=<size('VECTOR1'),
     size('VECTOR1')=size('VECTOR2')].

ctr_example(
    differ_from_at_least_k_pos,
    differ_from_at_least_k_pos(2,
        [[var-2],[var-5],[var-2],[var-0]],
        [[var-3],[var-6],[var-2],[var-1]])).

ctr_typical(
    differ_from_at_least_k_pos,
    ['K'>0,size('VECTOR1')>1]).

ctr_exchangeable(
    differ_from_at_least_k_pos,
    [args([['K'],['VECTOR1','VECTOR2']]),
     vals([['K'],int(>=(0)),>,dontcare,dontcare),
     items_sync('VECTOR1','VECTOR2',all)]).
ctr_graph(
  differ_from_at_least_k_pos,
  ['VECTOR1','VECTOR2'],
  2,
  ['PRODUCT'(=)>>collection(vector1,vector2)],
  ['VECTOR1'\'VAR'\'VECTOR2'\'VAR'],
  ['NARC'='K'],
  []).

ctr_eval(
  differ_from_at_least_k_pos,
  reformulation(differ_from_at_least_k_pos_r),
  automaton(differ_from_at_least_k_pos_a)).

ctr_extensible(
  differ_from_at_least_k_pos,
  [],
  ['VARIABLES1','VARIABLES2'],
  any).

differ_from_at_least_k_pos_r(K,VECTOR1,VECTOR2) :-
  integer(K),
  collection(VECTOR1,[dvar]),
  collection(VECTOR2,[dvar]),
  length(VECTOR1,N1),
  length(VECTOR2,N2),
  K>=0,
  K=<N1,
  N1=N2,
  N1>=1,
  differ_from_at_least_k_pos1(VECTOR1,VECTOR2,SumBool),
  call(K#=<SumBool).

differ_from_at_least_k_pos1([],[],0).

differ_from_at_least_k_pos1( [[\_31657-V1]\_R1],
  [[\_31668-V2]\_R2],
  B+R) :-
  V1\'VAR'\=V2\'VAR'\<=B,
  differ_from_at_least_k_pos1(R1,R2,R).

differ_from_at_least_k_pos_a(FLAG,K,VECTOR1,VECTOR2) :-
  integer(K),
  collection(VECTOR1,[dvar]),
  collection(VECTOR2,[dvar]),
length(VECTOR1,N1),
length(VECTOR2,N2),
K>=0,
K=<N1,
N1=N2,
N1>=1,
differ_from_at_least_k_pos_signature(
    VECTOR1,
    VECTOR2,
    SIGNATURE),
automaton(
    SIGNATURE,
    _34217,
    SIGNATURE,
    [source(s),sink(s)],
    [arc(s,0,s,[C+1]),arc(s,1,s)],
    [C],
    [0],
    [COUNT]),
COUNT#>=K#<=>FLAG.

differ_from_at_least_k_pos_signature([],[],[]).

differ_from_at_least_k_pos_signature(
    [[var-VAR1]|VARS1],
    [[var-VAR2]|VARS2],
    [S|Ss]) :-
    VAR1#=VAR2#<=>S,
    differ_from_at_least_k_pos_signature(VARS1,VARS2,Ss).
B.106  diffn

◇ Meta-Data:

ctr_date(diffn, ['20000128', '20030820', '20040530', '20051001', '20060808']).

ctr_origin(diffn, '\cite{BeldiceanuContejean94}', []).

ctr_synonyms(diffn, [disjoint, disjoint1, disjoint2, diff2]).

ctr_types(diffn, ['ORTHOTOPE'-collection(ori-dvar, siz-dvar, end-dvar)]).

ctr_arguments(diffn, ['ORTHOTOPES'-collection(orth-'ORTHOTOPE')]).

ctr_restrictions(diffn, [size('ORTHOTOPE')>0, require_at_least(2, 'ORTHOTOPE', [ori, siz, end]), 'ORTHOTOPE'\^siz=0, 'ORTHOTOPE'\^ori=<'ORTHOTOPE'\^end, required('ORTHOTOPES', orth), same_size('ORTHOTOPES', orth)]).

ctr_example(diffn, diffn([orth-[[ori-2, siz-2, end-4], [ori-1, siz-3, end-4]], [ori-4, siz-4, end-8], [ori-3, siz-3, end-6]], [ori-9, siz-2, end-11], [ori-4, siz-3, end-7]])].)

ctr_typical(diffn, [size('ORTHOTOPE')>1, 'ORTHOTOPE'\^siz>0, size('ORTHOTOPES')>1]).

ctr_exchangeable(diffn, [items('ORTHOTOPES', all), items_sync('ORTHOTOPES'\^orth, all),
vals(
    ["ORTHOTOPES"^orth^siz],
    int(>=0)),
>,
dontcare,
dontcare),
translate(["ORTHOTOPES"^orth^ori,"ORTHOTOPES"^orth^end])).

ctr_graph(
    dfnn,
    ["ORTHOTOPES"],
    1,
    ["SELF">>collection(orthotopes)],
    [orth_link_ori_siz_end(orthotopes^orth)],
    ["NARC"=size("ORTHOTOPES")],
    []).

ctr_graph(
    dfnn,
    ["ORTHOTOPES"],
    2,
    ["CLIQUE"(\=)>>collection(orthotopes1,orthotopes2)],
    [two_orth_do_not_overlap(orthotopes1^orth,
                              orthotopes2^orth)],
    [NARC= size("ORTHOTOPES")^2 size("ORTHOTOPES")-size("ORTHOTOPES")],
    []).

ctr_eval(dfnn,[reformulation(dfnn_r)]).

ctr_contractible(dfnn,[],"ORTHOTOPES",any).

dfnn_r([]) :-
    !.

dfnn_r(ORTHOTOPES) :-
    ORTHOTOPES=[[_65238-ORTH1] _65234],
    length(ORTH1,K),
    collection("ORTHOTOPES",
        [col(K,[dvar,dvar_gteq(0),dvar])]),
    get_col_attr1(ORTHOTOPES,1,ORIS),
    get_col_attr1(ORTHOTOPES,2,SIZS),
    get_col_attr1(ORTHOTOPES,3,ENDS),
    ( K=2 ->
diffn0(ORIS,SIZS,ENDS,RECTS),
disjoint2(RECTS)
;  diffn_fixed_size(SIZS) ->
  length(Zeros,K),
  domain(Zeros,0,0),
  diffn5(ORIS,SIZS,ENDS,1,Zeros,OBJS,SHAPES),
  geost(OBJS,SHAPES)
;  diffn1(ORIS,SIZS,ENDS)
).

diffn_fixed_size([]).

diffn_fixed_size([L|R]) :-
  diffn_fixed_size1(L),
  diffn_fixed_size(R).

diffn_fixed_size1([]).

diffn_fixed_size1([S|R]) :-
  integer(S),
  diffn_fixed_size1(R).

diffn0([],[],[],[]).

diffn0([[X,Y]|ORIS],[[L,H]|SIZS],[END|ENDS],[t(X,L,Y,H)|R]) :-
  diffn2([X,Y],[L,H],END),
  diffn0(ORIS,SIZS,ENDS,R).

diffn1([ORI1],[SIZ1],[END1]) :-
  !,
  diffn2(ORI1,SIZ1,END1).

diffn1([ORI1,ORI2|ORIS],[SIZ1,SIZ2|SIZS],[END1,END2|ENDS]) :-
  diffn2(ORI1,SIZ1,END1),
  diffn3([ORI2|ORIS],[END2|ENDS],ORI1,END1),
  diffn1([ORI2|ORIS],[SIZ2|SIZS],[END2|ENDS]).

diffn2([],[],[],[]).

diffn2([O|RO],[S|RS],[E|RE]) :-
  E#=O+S,
  diffn2(RO,RS,RE).

diffn3([],[],_65223,_65224).

diffn3([ORI2|ORIS],[END2|ENDS],ORI1,END1) :-
diffn4(ORI1, END1, ORI2, END2, Disjunction),
call(Disjunction),
diffn3(ORIS, ENDS, ORI1, END1).

diffn4([], [], [], [], 0).

diffn4([O1|R], [E1|S], [O2|T], [E2|U], E1#=<O2#\/E2#=<O1#\/V) :-

diffn5([], [], [], _65224, _65225, [], []).

diffn5(
    [ORI|ORIS],
    [SIZ|SIZS],
    [END|ENDS],
    I, Zeros,
    [object(I, I, ORI)|OBS],
    [sbox(I, Zeros, SIZ)|SHAPES]) :-
diffn2(ORI, SIZ, END),
    I1 is I+1,
    diffn5(ORIS, SIZS, ENDS, I1, Zeros, OBS, SHAPES).
B.107 diffn_column

◊ META-DATA:

ctr_date(diffn_column,['20030820']).

ctr_origin(
        diffn_column,
        \index{CHIP|indexuse}CHIP: option guillotine cut (column) of %c.,
        [diffn]).

ctr_types(
        diffn_column,
        ['ORTHOTOPE'-collection(ori-dvar,siz-dvar,end-dvar)]).

ctr_arguments(
        diffn_column,
        ['ORTHOTOOPES'-collection(orth-'ORTHOTOPE'),'DIM'-int]).

ctr_restrictions(
        diffn_column,
        [size('ORTHOTOPE')>0,
         require_at_least(2,'ORTHOTOPE',[ori,siz,end]),
         'ORTHOTOPE'~siz>=0,
         'ORTHOTOPE'~ori=<'ORTHOTOPE'~end,
         required('ORTHOTOOPES',orth),
         same_size('ORTHOTOOPES',orth),
         'DIM'>0,
         'DIM'=<size('ORTHOTOPE'),
         diffn('ORTHOTOOPES')]).

ctr_example(
        diffn_column,
        diffn_column(
                [[orth-[[ori-1,siz-3,end-4],[ori-3,siz-2,end-5]]],
                [orth-[[ori-9,siz-1,end-10],[ori-4,siz-3,end-7]]],
                [orth-[[ori-4,siz-2,end-6],[ori-3,siz-4,end-7]]],
                [orth-[[ori-1,siz-3,end-4],[ori-6,siz-1,end-7]]],
                [orth-[[ori-6,siz-2,end-8],[ori-1,siz-4,end-5]]],
                [orth-[[ori-10,siz-1,end-11],[ori-1,siz-1,end-2]]],
                [orth-[[ori-9,siz-1,end-10],[ori-1,siz-1,end-2]]],
                [orth-[[ori-6,siz-2,end-8],[ori-6,siz-1,end-7]]]],
       1)).

ctr_typical(
        diffn_column,
[size('ORTHOTOPE')>1,
 'ORTHOTOPE'\textasciicircum{size}>0,
 size('ORTHOTOPES')>1]).

\texttt{ctr\textunderscore exchangeable(}
\texttt{diffn\textunderscore column,}
\texttt{[items('ORTHOTOPES',all),
\texttt{translate(['ORTHOTOPES'\textasciicircum{orth\textunderscore ori},
'ORTHOTOPES'\textasciicircum{orth\textunderscore end}])].}

\texttt{ctr\textunderscore graph(}
\texttt{diffn\textunderscore column,}
\texttt{['ORTHOTOPES'],
2,
['CLIQUE'(<)\textasciicircum{}collection(orthotopes1,orthotopes2)),
[two\textunderscore orth\textunderscore column(orthotopes1\textasciicircum{orth},orthotopes2\textasciicircum{orth},'DIM')),
['NARC'=size('ORTHOTOPES')\times(size('ORTHOTOPES')-1)/2],
[]).

\texttt{ctr\textunderscore eval(diffn\textunderscore column,[reformulation(diffn\textunderscore column\_r)]).}

\texttt{ctr\textunderscore contractible(diffn\textunderscore column,[],'ORTHOTOPES',any).}

\texttt{diffn\textunderscore column\_r([],DIM) :-
\texttt{integer(DIM),
DIM>0.}

\texttt{diffn\textunderscore column\_r(ORTHOTOPES,DIM) :-
\texttt{ORTHOTOPES=\{[\_32290-ORTH1]|\_32286\},
length(ORTH1,K),
collection(\texttt{ORTHOTOPES,}
\texttt{[col(K,[dvar,dvar\_gteq(0),dvar])]}),
check\texttt{_type(int(1,K),DIM),}
eval\texttt{diffn(ORTHOTOPES)),}
get\texttt{_attr1(ORTHOTOPES,ORTHOTOPES1),}
diffn\texttt{_column1(ORTHOTOPES1,DIM).}

\texttt{diffn\textunderscore column1([],_32271).
\texttt{diffn\textunderscore column1([\_32275],_32274) :-}
\texttt{!.
\texttt{diffn\textunderscore column1([01,02|R],DIM) :-
eval(two\textunderscore orth\textunderscore column(01,02,DIM)),
diffn\textunderscore column1([02|R],DIM).}
B.108  diffn_include

◊ Meta-Data:

ctr_date(diffn_include,['20030820','20090523']).

ctr_origin(diffn_include,
\index{CHIP|indexuse}CHIP: option guillotine cut (include) of %c.,
[diffn]).

ctr_types(diffn_include,
[‘ORTHOTOPE’-collection(ori-dvar,siz-dvar,end-dvar)]).

ctr_arguments(diffn_include,
[‘ORTHOTOPE’-collection(orth-‘ORTHOTOPE’),’DIM’-int]).

ctr_restrictions(diffn_include,
[size(‘ORTHOTOPE’)>0,
require_at_least(2,’ORTHOTOPE’,[ori,siz,end]),
’ORTHOTOPE’`siz>=0,
’ORTHOTOPE’`ori<’ORTHOTOPE’`end,
required(‘ORTHOTOPE’,orth),
same_size(‘ORTHOTOPE’,orth),
’DIM’>0,
’DIM’=<size(‘ORTHOTOPE’),
diffn(‘ORTHOTOPE’))).

ctr_example(diffn_include,
diffn_include(
[[orth-[[ori-8,siz-1,end-9],[ori-4,siz-1,end-5]]],
 [orth-[[ori-9,siz-1,end-10],[ori-4,siz-3,end-7]]],
 [orth-[[ori-6,siz-3,end-9],[ori-5,siz-2,end-7]]],
 [orth-[[ori-1,siz-3,end-4],[ori-6,siz-1,end-7]]],
 [orth-[[ori-4,siz-2,end-6],[ori-3,siz-4,end-7]]],
 [orth-[[ori-6,siz-4,end-10],[ori-1,siz-1,end-2]]],
 [orth-[[ori-10,siz-1,end-11],[ori-1,siz-1,end-2]]],
 [orth-[[ori-6,siz-5,end-11],[ori-2,siz-2,end-4]]],
 [orth-[[ori-6,siz-2,end-8],[ori-4,siz-1,end-5]]],
 [orth-[[ori-1,siz-5,end-6],[ori-1,siz-2,end-3]]],
 [orth-[[ori-1,siz-3,end-4],[ori-3,siz-2,end-5]]],
 [orth-[[ori-1,siz-2,end-3],[ori-5,siz-1,end-6]]],

[2269]
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

1).

\[
\text{ctr\_typical}(\ \text{diffn\_include}, \\
\quad [\text{size('ORTHOTOPE')}>1, \\
\quad \text{ORTHOTOPE}^\text{\textasciitilde siz}>0, \\
\quad \text{size('ORTHOTOPES')}>1]).
\]

\[
\text{ctr\_exchangeable}(\ \text{diffn\_include}, \\
\quad [\text{items('ORTHOTOPES',all),} \\
\quad \text{translate}([\text{ORTHOTOPE}^\text{\textasciitilde\textasciitilde ori},\text{ORTHOTOPE}^\text{\textasciitilde\textasciitilde end}])]).
\]

\[
\text{ctr\_graph}(\ \text{diffn\_include}, \\
\quad ['\text{ORTHOTOPES'}], \\
\quad 2, \\
\quad ['\text{CLIQUES'}(\text{<})\text{collection}(\text{ORTHOTOPES1,ORTHOTOPES2}), \\
\quad \text{two\_orth\_include}(
\quad \quad \text{ORTHOTOPES1}^\text{\textasciitilde orth}, \\
\quad \quad \text{ORTHOTOPES2}^\text{\textasciitilde orth}, \\
\quad \quad \text{DIM}), \\
\quad ['\text{NARC'}=\text{size('ORTHOTOPES')}*(\text{size('ORTHOTOPES')}-1)/2], \\
\quad [])).
\]

\[
\text{ctr\_eval}(\text{diffn\_include},[\text{reformulation(\text{diffn\_include\_r})}]).
\]

\[
\text{ctr\_contractible}(\text{diffn\_include},[],'\text{ORTHOTOPES'},\text{any}).
\]

\[
\text{diffn\_include\_r}([],\text{DIM}) :- \\
\quad \text{integer(DIM)}, \\
\quad \text{DIM}>0.
\]

\[
\text{diffn\_include\_r}(\text{ORTHOTOPES},\text{DIM}) :- \\
\quad \text{ORTHOTOPES=}[[_34299-\text{ORTH1}]_34295], \\
\quad \text{length(ORTH1,K)}, \\
\quad \text{collection(} \\
\quad \quad \text{ORTHOTOPES,} \\
\quad \quad \quad [\text{col}(K,\text{[dvar,dvar\_gteq(0),dvar]])]), \\
\quad \text{check\_type(int(1,K),DIM)}, \\
\quad \text{eval(\text{diffn}(\text{ORTHOTOPES})),} \\
\quad \text{get\_attr1(ORTHOTOPES,ORTHOTOPES1),} \\
\quad \text{diffn\_include1(ORTHOTOPES1,DIM)}.
\]

\[
\text{diffn\_include1}([],_34280).
\]
diffn_include1([__34284],__34283) :-
  !.

diffn_include1([O1,O2|R],DIM) :-
  eval(two_orth_include(O1,O2,DIM)),
  diffn_include1([O2|R],DIM).
B.109 discrepancy

◇ Meta-Data:

ctr_date(discrepancy,[''20050506',''20060808']).

ctr_origin(
    discrepancy,
    \cite{Focacci01} and \cite{vanHoeve05},[]).

ctr_arguments(
    discrepancy,
    [''VARIABLES'-collection(var-dvar,bad-sint),''K'-int]).

ctr_restrictions(
    discrepancy,
    [required('VARIABLES',var),
     required('VARIABLES',bad),
     'K'\geq0,
     'K'\leq<size('VARIABLES')]].

ctr_example(
    discrepancy,
    discrepancy(
        [[var-4,bad-{1,4,6}],
         [var-5,bad-{0,1}],
         [var-5,bad-{1,6,9}],
         [var-4,bad-{1,4}],
         [var-1,bad-{}]],
        2)).

ctr_typical(
    discrepancy,
    [size('VARIABLES')\geq1,'K'\leq<size('VARIABLES')]].

ctr_exchangeable(
    discrepancy,
    [items('VARIABLES',all),
    vals(
        [''VARIABLES''\var,''VARIABLES''\bad],
        int,
        =\neq,
        all,
        dontcare)]).
ctr_graph(
    discrepancy,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables),
     [variables`var in_set variables`bad],
     ['NARC'='K'],
     []).

ctr_pure_functionalDependency(discrepancy,[]).

ctr_functionalDependency(discrepancy,2,[1]).

ctr_aggregate(discrepancy,[],[union,+]).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.110 disj

◊ Meta-Data:

ctr_date(disj,'[20070527]').

ctr_origin(disj,'\cite{MonetteDevilleDupont07}',[]).

ctr_arguments(disj,
  [TASKS-
    collection(
      start-dvar,
      duration-dvar,
      before-svar,
      position-dvar))].

ctr_restrictions(disj,
  [required('TASKS',[start,duration,before,position]),
   'TASKS'\duration\Textless;=1,
   'TASKS'\position\Textless;=0,
   'TASKS'\position\Textless;\textsize('TASKS')].

ctr_example(disj,
  disj(
    [[start-1,duration-3,before-{},position-0],
    [start-9,duration-1,before-{1,3,4},position-3],
    [start-7,duration-2,before-{1,4},position-2],
    [start-4,duration-1,before-{1},position-1]]).)

ctr_typical(disj,[\textsize('TASKS')\textgreater;1]).

ctr_exchangeable(disj,
  translate(['TASKS'\start]),
  vals(['TASKS'\duration],\textsize(\geq (1)),\textgreater;,\textsize(dontcare,dontcare))].

ctr_graph(disj,
  ['TASKS'],
  2,
  ['CLIQUE' (=\=)>collection(tasks1,tasks2)],
  [tasks1\textasciitilde start\textasciitilde tasks1\textasciitilde duration\textless;\textasciitilde tasks2\textasciitilde start\#\textasciitilde
   tasks2\textasciitilde start\textasciitilde tasks2\textasciitilde duration\textless;\textasciitilde tasks1\textasciitilde start,
tasks1\start+tasks1\duration=<tasks2\start#<=>
tasks1\key in_set tasks2\before,
tasks1\start+tasks1\duration=<tasks2\start#<=>
tasks1\position<tasks2\position],
[\NARC'=size('TASKS')*size('TASKS')-size('TASKS')],
[\]}. 

B.111 disjoint

◊ **Meta-Data:**

```prolog
ctr_date(disjoint, 
    ['20000315', '20031017', '20040530', '20060808']).

ctr_origin(disjoint, 'Derived from %c.', [alldifferent]).

ctr_arguments(disjoint, 
    ['VARIABLES1'~collection(var-dvar), 
    'VARIABLES2'~collection(var-dvar)]).

ctr_restrictions(disjoint, 
    [required('VARIABLES1', var), required('VARIABLES2', var)]).

ctr_example(disjoint, 
    disjoint( 
        [[var-1], [var-9], [var-1], [var-5]], 
        [[var-2], [var-7], [var-7], [var-0], [var-6], [var-8]])).

ctr_typical(disjoint, 
    [size('VARIABLES1')>1, size('VARIABLES2')>1]).

ctr_exchangeable(disjoint, 
    [args([['VARIABLES1', 'VARIABLES2']]), 
    items('VARIABLES1', all), 
    items('VARIABLES2', all), 
    vals(['VARIABLES1'~var], int, =\=, dontcare, in), 
    vals(['VARIABLES2'~var], int, =\=, dontcare, in), 
    vals([ 
        ['VARIABLES1'~var, 'VARIABLES2'~var], 
        int, 
        =\=, 
        all, 
        dontcare])].

ctr_graph(disjoint, 
    ['VARIABLES1', 'VARIABLES2'],
```

2,
['PRODUCT'>>collection(variables1,variables2)],
[variables1\^\text{var}=variables2\^\text{var}],
['NARC'=0],
[]).

ctr_eval(disjoint,[reformulation(disjoint_r)]).

ctr_contractible(disjoint,[],'VARIABLES1',any).

ctr_contractible(disjoint,[],'VARIABLES2',any).

disjoint_r(VARIABLES1,VARIABLES2) :-
collection(VARIABLES1,[dvar]),
collection(VARIABLES2,[dvar]),
get_attr1(VARIABLES1,VARS1),
get_attr1(VARIABLES2,VARS2),
disjoint1_(VARS1,VARS2).

disjoint1_([],_33614).

disjoint1_([V|R],VARS2) :-
disjoint2_(VARS2,V),
disjoint1_(R,VARS2).

disjoint2_([],_33614).

disjoint2_([U|R],V) :-
U\#\neq V,
disjoint2_(R,V).
B.112 disjoint_sboxes

Meta-Data:

ctr_date(disjoint_sboxes, ['20070622', '20090725']).

ctr_origin(
    disjoint_sboxes,
    Geometry, derived from \cite{RandellCuiCohn92}, []).

ctr_synonyms(disjoint_sboxes, [disjoint]).

ctr_types(
    disjoint_sboxes,
    ['VARIABLES'-collection(v-dvar),
    'INTEGERS'-collection(v-int),
    'POSITIVES'-collection(v-int)]).

ctr_arguments(
    disjoint_sboxes,
    ['K'-int,
    'DIMS'-sint,
    'OBJECTS'-collection(oid-int, sid-int, x-'VARIABLES'),
    'SBOXES'-collection(sid-int, t-'INTEGERS', l-'POSITIVES')]).

ctr_restrictions(
    disjoint_sboxes,
    [size('VARIABLES')>=1,
    size('INTEGERS')>=1,
    size('POSITIVES')>=1,
    required('VARIABLES', v),
    size('VARIABLES')='K',
    required('INTEGERS', v),
    size('INTEGERS')='K',
    required('POSITIVES', v),
    size('POSITIVES')='K',
    'POSITIVES'~v>0,
    'K'>0,
    'DIMS'>=0,
    'DIMS'<'K',
    increasing_seq('OBJECTS', [oid]),
    required('OBJECTS', [oid, sid, x]),
    'OBJECTS'~oid>=1,
    'OBJECTS'~oid=size('OBJECTS'),
    'OBJECTS'~sid>=1,
'OBJECTS'\textasciitilde{\texttt{sid}}=<\texttt{size('SBOXES')},
\texttt{size('SBOXES')}>=1,
\texttt{required('SBOXES', [\texttt{sid}, \texttt{t}, \texttt{l}]),}
'SBOXES'\texttt{\textasciitilde{\texttt{sid}}}=1,
'SBOXES'\texttt{\textasciitilde{\texttt{sid}}}<\texttt{size('SBOXES')},
do\_\texttt{not\_}\texttt{overlap('SBOXES'))}.

ctr\_example(
  disjoint\_sboxes,
  disjoint\_sboxes(2,
   [0,1],
   [[oid-1, sid-1, x-[[v-1], [v-1]]],
    [oid-2, sid-2, x-[[v-4], [v-1]]],
    [oid-3, sid-4, x-[[v-2], [v-4]]],
    [[sid-1, t-[[v-0], [v-0]], l-[[v-1], [v-2]]],
    [sid-2, t-[[v-0], [v-0]], l-[[v-1], [v-1]]],
    [sid-2, t-[[v-1], [v-0]], l-[[v-1], [v-3]]],
    [sid-2, t-[[v-0], [v-2]], l-[[v-1], [v-1]]],
    [sid-3, t-[[v-0], [v-0]], l-[[v-3], [v-1]]],
    [sid-3, t-[[v-0], [v-1]], l-[[v-1], [v-1]]],
    [sid-3, t-[[v-2], [v-1]], l-[[v-1], [v-1]]],
    [sid-4, t-[[v-0], [v-0]], l-[[v-1], [v-1]]])).

ctr\_typical(disjoint\_sboxes, [\texttt{size('OBJECTS')}>1]).

ctr\_exchangeable(
  disjoint\_sboxes,
  [\texttt{items('OBJECTS', all)},
   \texttt{items('SBOXES', all)},
   \texttt{vals(['SBOXES'\textasciitilde{l}\textasciitilde{v}], \texttt{int(\texttt{\textasciitilde{\texttt{>=}}}(1)), \texttt{>}, \texttt{dontcare}, \texttt{dontcare}]))}.

ctr\_eval(disjoint\_sboxes, [\texttt{logic(disjoint\_sboxes\_g)}]).

ctr\_logic(
  disjoint\_sboxes,
  [\texttt{DIMENSIONS}, \texttt{OIDS}],
  [(\texttt{origin(O1, S1, D)}\texttt{-}\texttt{-}\texttt{O1}\texttt{\textasciitilde{x}(D)}+\texttt{S1}\texttt{\textasciitilde{t}(D)})],
  (\texttt{end(O1, S1, D)}\texttt{-}\texttt{-}\texttt{O1}\texttt{\textasciitilde{x}(D)}+\texttt{S1}\texttt{\textasciitilde{t}(D)}+\texttt{S1}\texttt{\textasciitilde{l}(D)})],
  (disjoint\_sboxes(Dims, O1, S1, O2, S2)\texttt{-}\texttt{-}\texttt{exists(D, Dims),}
   \texttt{origin(O1, S1, D)}\texttt{-}\texttt{-}\texttt{end(O2, S2, D)}\texttt{-}\texttt{-}\texttt{/}
   \texttt{origin(O2, S2, D)}\texttt{-}\texttt{-}\texttt{end(O1, S1, D))},
  (disjoint\_objects(Dims, O1, O2)\texttt{-}\texttt{-}\texttt{)}.
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

forall(
    S1,
    sboxes([O1\^sid]),
    forall(
        S2,
        sboxes([O2\^sid]),
        disjoint_sboxes(Dims,O1,S1,O2,S2))))),
(all_disjoint(Dims,OIDS)--->
forall(
    O1,
    objects(OIDS),
   forall(
        O2,
        objects(OIDS),
        O1\^oid<O2\^oid=>
        disjoint_objects(Dims,O1,O2)))).

ctr_contractible(disjoint_sboxes,[],'OBJECTS',suffix).

disjoint_sboxes_g(K,_30642,[],_30644) :-
    !,
    check_type(int_gteq(1),K).

disjoint_sboxes_g(K,_DIMS,OBJECTS,SBOXES) :-
    length(OBJECTS,O),
    length(SBOXES,S),
    O>0,
    S>0,
    check_type(int_gteq(1),K),
    collection(OBJECTS,[int(1,O),dvar(1,S),col(K,[dvar]))],
    collection(    
        SBOXES,
        [int(1,S),col(K,[int]),col(K,[int_gteq(1)]))],
    get_attr1(OBJECTS,OIDS),
    get_attr2(OBJECTS,SIDS),
    get_col_attr3(OBJECTS,1,XS),
    get_attr1(SBOXES,SIDES),
    get_col_attr2(SBOXES,1,TS),
    get_col_attr3(SBOXES,1,TL),
    collection_increasing_seq(OBJECTS,[1]),
    geost1(OIDS,SIDS,XS,Objects),
    geost2(SIDES,TS,TL,Sboxes),
    geost_dims(1,K,DIMENSIONS),
    ctr_logic(disjoint_sboxes,[DIMENSIONS,OIDS],Rules),
    geost(Objects,Sboxes,[overlap(true)],Rules).
B.113 disjoint\_tasks

\[\text{\textbf{Meta-Data:}}\]

\begin{verbatim}
ctr_date(disjoint\_tasks, [’20030820’, ’20060808’]).
ctr_origin(disjoint\_tasks, ’Derived from %c.’, [disjoint]).
ctr_arguments( disjoint\_tasks, ’TASKS1’\-collection(origin\-dvar, duration\-dvar, end\-dvar),
 ’TASKS2’\-collection(origin\-dvar, duration\-dvar, end\-dvar)).
ctr_restrictions( disjoint\_tasks, require\_at\_least(2, ’TASKS1’, [origin, duration, end]),
 ’TASKS1’\^duration>=0,
 ’TASKS1’\^origin=<’TASKS1’\^end,
 require\_at\_least(2, ’TASKS2’, [origin, duration, end]),
 ’TASKS2’\^duration>=0,
 ’TASKS2’\^origin=<’TASKS2’\^end)).
ctr_example( disjoint\_tasks, disjoint\_tasks( [[origin-6, duration-5, end-11], [origin-8, duration-2, end-10]],
 [[origin-2, duration-2, end-4], [origin-3, duration-3, end-6],
 [origin-12, duration-1, end-13]])).
ctr_typical( disjoint\_tasks, [size(’TASKS1’)>1,
 ’TASKS1’\^duration>0,
 size(’TASKS2’)>1,
 ’TASKS2’\^duration>0]).
ctr_exchangeable( disjoint\_tasks, [args([[’TASKS1’, ’TASKS2’]]),
 items(’TASKS1’, all),
 items(’TASKS2’, all),
 translate( [’TASKS1’\^origin,
 ’TASKS1’\^end,}
\end{verbatim}
ctr_graph(
    disjoint_tasks,
    ['TASKS1'],
    1,
    ['SELF'>>collection(tasks1)],
    [tasks1\^origin+tasks1\^duration=tasks1\^end],
    ['NARC'=size('TASKS1')],
    []).
ctr_graph(
    disjoint_tasks,
    ['TASKS2'],
    1,
    ['SELF'>>collection(tasks2)],
    [tasks2\^origin+tasks2\^duration=tasks2\^end],
    ['NARC'=size('TASKS2')],
    []).
ctr_graph(
    disjoint_tasks,
    ['TASKS1','TASKS2'],
    2,
    ['PRODUCT'>>collection(tasks1,tasks2)],
    [tasks1\^duration>0,
     tasks2\^duration>0,
     tasks1\^origin<tasks2\^end,
     tasks2\^origin<tasks1\^end],
    ['NARC'=0],
    []).
get_attr1(TASKS2,ORIGINS2),
get_attr2(TASKS2,DURATIONS2),
get_attr3(TASKS2,ENDS2),
ori_dur_end(ORIGINS2,DURATIONS2,ENDS2),
disjoint_tasks1(ORIGINS1,ENDS1,ORIGINS2,ENDS2).

disjoint_tasks1([],[],_35491,_35492).

disjoint_tasks1([O|R],[E|S],ORIGINS2,ENDS2) :-
  disjoint_tasks2(ORIGINS2,ENDS2,O,E),
  disjoint_tasks1(R,S,ORIGINS2,ENDS2).

disjoint_tasks2([],[],_35491,_35492).

disjoint_tasks2([Oj|R],[Ej|S],Oi,Ei) :-
  Ei#=<Oj#\Ej#=<Oi,
  disjoint_tasks2(R,S,Oi,Ei).
B.114 disjunctive

◊ META-DATA:

ctr_date(disjunctive, ['20050506', '20060808']).

ctr_origin(disjunctive, '\cite{Carlier82}', []).

ctr_synonyms(disjunctive, [one_machine]).

ctr_arguments(disjunctive, ['TASKS'\-collection(origin-dvar,duration-dvar)]).

ctr_restrictions(disjunctive, [required('TASKS',[origin,duration]),'TASKS'\^duration>=0]).

ctr_example(disjunctive, disjunctive(
    [[origin-1,duration-3],
     [origin-2,duration-0],
     [origin-7,duration-2],
     [origin-4,duration-1]]).

ctr_typical(disjunctive, [size('TASKS')>1,'TASKS'\^duration>=1]).

ctr_exchangeable(disjunctive, [items('TASKS',all),
     vals(['TASKS'\^duration],int(>=(0)),>,dontcare,dontcare),
     translate(['TASKS'\^origin])].

ctr_graph(disjunctive, ['TASKS'],
   ['CLIQUE'(<)>>collection(tasks1,tasks2)],
   [tasks1\^duration=0#\tasks2\^duration=0#\tasks1\^origin+tasks1\^duration=<tasks2\^origin#\tasks2\^origin+tasks2\^duration=<tasks1\^origin],
   ['NARC'=size('TASKS')*(size('TASKS')-1)/2],
   []).

ctr_eval(disjunctive, [builtin(disjunctive_b)]).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

\[
\text{ctr_contractible} (\text{disjunctive}, [], \text{'TASKS'}, \text{any}).
\]

\[
\text{disjunctive_b}([],) :-
!.
\]

\[
\text{disjunctive_b} (\text{TASKS}) :-
\quad \text{length} (\text{TASKS}, N),
\quad N \gt 1,
\quad \text{collection} (\text{TASKS}, \text{[dvar, dvar_gteq(0)]}),
\quad \begin{cases}
\quad \text{N=1} \rightarrow \\
\quad \text{true}
\end{cases}
\quad ;
\quad \text{get_attr1} (\text{TASKS}, \text{ORIGINS}),
\quad \text{get_attr2} (\text{TASKS}, \text{DURATIONS}),
\quad \text{length} (\text{ENDS}, N),
\quad \text{ori_dur_end} (\text{ORIGINS}, \text{DURATIONS}, \text{ENDS}),
\quad \text{length} (\text{HEIGHTS}, N),
\quad \text{domain} (\text{HEIGHTS}, 1, 1),
\quad \text{gen_cum_tasks} (\text{ORIGINS},
\quad \text{DURATIONS},
\quad \text{ENDS},
\quad \text{HEIGHTS},
\quad 1,
\quad \text{Tasks}),
\quad \text{cumulative} (\text{Tasks}, [\text{limit}(1)])
\quad )
\].
B.115 disjunctive_or_same_end

◊ **META-DATA:**

```prolog
ctr_date(disjunctive_or_same_end, ['20120205']).

ctr_origin(disjunctive_or_same_end, 'Scheduling.', []).

ctr_synonyms(
    disjunctive_or_same_end,
    [same_end_or_disjunctive, non_overlap_or_same_end, same_end_or_non_overlap]).

ctr_arguments(
    disjunctive_or_same_end,
    ['TASKS'-collection(origin-dvar, duration-dvar)]).

ctr_restrictions(
    disjunctive_or_same_end,
    [required('TASKS', [origin, duration]), 'TASKS'\^duration>=0]).

ctr_example(
    disjunctive_or_same_end,
    disjunctive_or_same_end(
        [[origin-4, duration-3],
         [origin-7, duration-2],
         [origin-5, duration-2]])).

ctr_typical(
    disjunctive_or_same_end,
    [size('TASKS')>1, 'TASKS'\^duration>=1]).

ctr_exchangeable(
    disjunctive_or_same_end,
    [items('TASKS', all),
     vals(['TASKS'\^duration, int(>=0)], >, dontcare, dontcare),
     translate(['TASKS'\^origin])]).

ctr_graph(
    disjunctive_or_same_end,
    ['TASKS'],
    2,
    ['CLIQUE'(<)\>>collection(tasks1, tasks2)],
    [tasks1^duration=0#\tasks2^duration=0#/
     tasks1^origin+tasks1^duration=<tasks2^origin#/
     ...]
)
```
tasks2^origin+tasks2^duration=<tasks1^origin#
\-
tasks1^origin+tasks1^duration=
tasks2^origin+tasks2^duration],
[‘NARC’=size(‘TASKS’)*(size(‘TASKS’)-1)/2],
[]).

ctr_eval(
    disjunctive_or_same_end,
    [builtin(disjunctive_or_same_end_r)]).

ctr_contractible(disjunctive_or_same_end,[],‘TASKS’,any).

disjunctive_or_same_end_r([]) :-
    !.

disjunctive_or_same_end_r(TASKS) :-
    collection(TASKS,[dvar,dvar_gteq(0)]),
    get_attr1(TASKS,ORIGINS),
    get_attr2(TASKS,DURATIONS),
    disjunctive_or_same_end1(ORIGINS,DURATIONS).

disjunctive_or_same_end1([],[]).

disjunctive_or_same_end1([ORI|RO],[DUR|RD]) :-
    disjunctive_or_same_end2(RO,RD,ORI,DUR),
    disjunctive_or_same_end1(RO,RD).

disjunctive_or_same_end2([],[],_27246,_27247).

disjunctive_or_same_end2([O2|RO],[D2|RD],O1,D1) :-
    \D1#0\D2#0\O1+D1=<O2\O2+D2=<O1\O1+D1=O2+D2,
    disjunctive_or_same_end2(RO,RD,O1,D1).
B.116 disjunctive_or_same_start

◊ **META-DATA:**

```prolog
ctr_date(disjunctive_or_same_start, [’20120205’]).

ctr_origin(disjunctive_or_same_start, ’Scheduling.’, []).

ctr_synonyms(
    disjunctive_or_same_start,
    [same_start_or_disjunctive, non_overlap_or_same_start, same_start_or_non_overlap]).

ctr_arguments(
    disjunctive_or_same_start,
    [’TASKS’-collection(origin-dvar, duration-dvar)]).

ctr_restrictions(
    disjunctive_or_same_start,
    [required(’TASKS’, [origin, duration]), ’TASKS’^duration>=0]).

ctr_example(
    disjunctive_or_same_start,
    disjunctive_or_same_start(
        [[origin-4, duration-3],
        [origin-7, duration-2],
        [origin-4, duration-1]])).

ctr_typical(
    disjunctive_or_same_start,
    [size(’TASKS’) > 1, ’TASKS’^duration>=1]).

ctr_exchangeable(
    disjunctive_or_same_start,
    [items(’TASKS’, all),
     vals([’TASKS’^duration], int(>=0), >, dontcare, dontcare),
     translate([’TASKS’^origin])]).

ctr_graph(
    disjunctive_or_same_start,
    [’TASKS’],
    2,
    [’CLIQUE’(<>)>collection(tasks1, tasks2)],
    [tasks1^duration=0\ tasks2^duration=0\ tasks1^origin+tasks1^duration=<tasks2^origin]))
```
tasks2^origin+tasks2^duration=<tasks1^origin#
  tasks1^origin=tasks2^origin],
[NARC'=size('TASKS')*(size('TASKS')-1)/2],
[]).

ctr_eval(
  disjunctive_or_same_start,
  [builtin(disjunctive_or_same_start_r)]).

ctr_contractible(disjunctive_or_same_start,[],'TASKS',any).

disjunctive_or_same_start_r([],[]):-
  !.

disjunctive_or_same_start_r(TASKS):=-
  collection(TASKS,[dvar,dvar_gteq(0)]),
  get_attr1(TASKS,ORIGINS),
  get_attr2(TASKS,DURATIONS),
  disjunctive_or_same_start1(ORIGINS,DURATIONS).

disjunctive_or_same_start1([],[]).

disjunctive_or_same_start1([ORI|R0],[DUR|RD]):=-
  disjunctive_or_same_start2(R0,RD,ORI,DUR),
  disjunctive_or_same_start1(R0,RD).

disjunctive_or_same_start2([],[],_27082,_27083).

disjunctive_or_same_start2([O2|R0],[D2|RD],O1,D1):=-
  D1#=0#\d2#=0#\o1+d1=#<o2#\o2+d2=#<o1#,\o1=#o2,
  disjunctive_or_same_start2(R0,RD,O1,D1).
B.117 distance

◊ **META-DATA:**

ctr_predefined(distance).

ctr_date(distance,[’20090416’]).

ctr_origin(distance,’Arithmetic constraint.’,[]).

ctr_arguments(distance,[’X’-dvar,’Y’-dvar,’Z’-dvar]).

ctr_restrictions(distance,[’Z’>=0]).

ctr_example(distance,distance(5,7,2)).

ctr_typical(distance,[’Z’>0]).

ctr_exchangeable(distance,[args([[’X’,’Y’],[’Z’]])]).

ctr_eval(distance,[builtin(distance_b)]).

ctr_pure_functional_dependency(distance,[]).

ctr_functional_dependency(distance,3,[1,2]).

distance_b(X,Y,Z) :-
    check_type(dvar,X),
    check_type(dvar,Y),
    check_type(dvar_gteq(0),Z),
    Z#=abs(X-Y).
B.118 distance_between

◊ **Meta-Data:**

```prolog
ctr_date(
    distance_between,
    ['20000128', '20030820', '20060808', '20090428']).

ctr_origin(distance_between, 'N. Beldiceanu', []).

ctr_synonyms(distance_between, [distance]).

ctr_arguments(
    distance_between,
    ['DIST'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'CTR'-atom]).

ctr_restrictions(
    distance_between,
    ['DIST'>=0,
     DIST=<
     size('VARIABLES1')*size('VARIABLES2')-size('VARIABLES1'),
     required('VARIABLES1', var),
     required('VARIABLES2', var),
     size('VARIABLES1')=size('VARIABLES2'),
     in_list('CTR', [=,\=,<,\>,>,=<])).

ctr_example(
    distance_between,
    distance_between(2,
        [[var-3],[var-4],[var-6],[var-2],[var-4],
         [[var-2],[var-6],[var-9],[var-3],[var-6]],
         <)).

ctr_typical(
    distance_between,
    ['DIST']>0,
    DIST<
    size('VARIABLES1')*size('VARIABLES2')-size('VARIABLES1'),
    size('VARIABLES1')>1,
    in_list('CTR', [=,\=])].

ctr_exchangeable(
```
distance_between,
[\texttt{args([['DIST'],['VARIABLES1','VARIABLES2'],['CTR']])}],
\texttt{items_sync('VARIABLES1','VARIABLES2',all)},
\texttt{translate(['VARIABLES1'^\texttt{var})],}
\texttt{translate(['VARIABLES2'^\texttt{var}])}].

\texttt{ctr_graph(}
\texttt{distance_between,}
\texttt{[['VARIABLES1'],['VARIABLES2']],}
\texttt{2,}
\texttt{['CLIQUE'(\texttt{=}\texttt{=})\texttt{>}}\texttt{collection(variables1,variables2)],}
\texttt{['CTR'(variables1^\texttt{var},variables2^\texttt{var}]},
\texttt{['DISTANCE'='DIST'],}
\texttt{[]}.\texttt{)}

\texttt{ctr_eval(distance_between,[\texttt{reformulation(distance_between_r)}]).}

\texttt{ctr_pure_functional_dependency(distance_between,[]).}

\texttt{ctr_functional_dependency(distance_between,1,[2,3,4]).}

\texttt{distance_between_r(DIST,VARIABLES1,VARIABLES2,CTR) :-}
\texttt{collection(VARIABLES1,[dvar]),}
\texttt{collection(VARIABLES2,[dvar]),}
\texttt{length(VARIABLES1,L1),}
\texttt{length(VARIABLES2,L2),}
\texttt{L1=L2,}
\texttt{L12 is L1+L2-L1,}
\texttt{check_type(dvar(0,L12),DIST),}
\texttt{memberchk(CTR,[=,\texttt{=}\texttt{=},\texttt{<},\texttt{\ge},\texttt{>},\texttt{\le}]),}
\texttt{get_attr1(VARIABLES1,VARS1),}
\texttt{get_attr1(VARIABLES2,VARS2),}
\texttt{distance_between1(VARS1,VARS2,1,VARS1,VARS2,CTR,TERM),}
\texttt{call(DIST#=TERM)\texttt{).}}

\texttt{distance_between1([],[],28792,28793,28794,28795,0).}

\texttt{distance_between1(}
\texttt{[VAR1|RVARS1],}
\texttt{[VAR2|RVARS2],}
\texttt{IVAR,}
\texttt{VARS1,}
\texttt{VARS2,}
\texttt{CTR,}
\texttt{TERM+R) :-}
\texttt{distance_between2(}
\begin{verbatim}
VARS1,
VARS2,
VAR1,
VAR2,
IVAR,
1,
CTR,
TERM),
IVAR1 is IVAR+1,
distance_between1(
    RVARS1,
    RVARS2,
    IVAR1,
    VARS1,
    VARS2,
    CTR,
    R).

distance_between2([],[],_28792,_28793,_28794,_28795,_28796,0).

distance_between2(
    [UAR1|RUARS1],
    [UAR2|RUARS2],
    VAR1,
    VAR2,
    IVAR,
    IUAR, =,
    B12+S) :-
    !,
    ( 
        IVAR=\=IUAR ->
        B12\=<>
        
        VAR1=UAR1\=/VAR2\=/UAR2\=UAR1\=/VAR1\=/UAR2
    ;
    
    B12=0
    ),
    IUAR1 is IUAR+1,
distance_between2(
    RUARS1,
    RUARS2,
    VAR1,
    VAR2,
    IVAR,
    IUAR1,
    =,
    S).
\end{verbatim}
distance_between2(  
  [UAR1|RUARS1],  
  [UAR2|RUARS2],  
  VAR1,  
  VAR2,  
  IVAR,  
  IUAR,  
  =\=,  
  B12+S) :-  
  !,  
  (  
    IVAR=\=IUAR ->  
    B12#<=>  
    VAR1#=UAR1#/VAR2#=UAR2#/VAR1#=UAR1#/VAR2#=UAR2  
    ;  
    B12=0  
  ),  
  IUAR1 is IUAR+1,  
  distance_between2(  
    RUARS1,  
    RUARS2,  
    VAR1,  
    VAR2,  
    IVAR,  
    IUAR,  
    =\=,  
    S).

distance_between2(  
  [UAR1|RUARS1],  
  [UAR2|RUARS2],  
  VAR1,  
  VAR2,  
  IVAR,  
  IUAR,  
  <,  
  B12+S) :-  
  !,  
  (  
    IVAR=\=IUAR ->  
    B12#<=>  
    VAR1#<UAR1#/VAR2#=UAR2#/VAR1#=UAR1#/VAR2#<UAR2  
    ;  
    B12=0  
  ),  
  IUAR1 is IUAR+1,  
  distance_between2(  
    RUARS1,  
    RUARS2,  
    VAR1,
distance_between2(
    [UAR1|RUARS1],
    [UAR2|RUARS2],
    VAR1,
    VAR2,
    IVAR,
    IUAR,
    <,
    S).

distance_between2(
    [UAR1|RUARS1],
    [UAR2|RUARS2],
    VAR1,
    VAR2,
    IVAR,
    IUAR,
    >=,
    B12+S) :-
    !,
    ( IVAR=\=IUAR ->
      B12#<=>
      VAR1#>=UAR1#/
      VAR2#<UAR2#/
      VAR1#=UAR1#/\VAR2#/<-UAR2#<-UAR1#/
      VAR2#=UAR2#<-UAR1#
      ;
      B12=0
    ),
    IUAR1 is IUAR+1,
    distance_between2(
        RUARS1,
        RUARS2,
        VAR1,
        VAR2,
        IVAR,
        IUAR1,
        >=,
        S).

distance_between2(
    [UAR1|RUARS1],
    [UAR2|RUARS2],
    VAR1,
    VAR2,
    IVAR,
    IUAR,
    >,
    B12+S) :-
    !,
    ( IVAR=\=IUAR ->
      B12#<>
      VAR1#>UAR1#/
      VAR2#=<UAR2#/
      VAR1#=uAR1#/\VAR2#/<UAR2#<UAR1#/\VAR2#=UAR2
      ;
      B12=0
    )
IUAR1 is IUAR+1,
distance_between2(  
  RUARS1,  
  RUARS2,  
  VAR1,  
  VAR2,  
  IVAR,  
  IUAR1,  
  >,  
  S).

distance_between2(  
  [UAR1|RUARS1],  
  [UAR2|RUARS2],  
  VAR1,  
  VAR2,  
  IVAR,  
  IUAR,  
  =<,  
  B12+S) :-  
  ( IVAR=\=IUAR -*  
    B12<=  
    VAR1=<UAR1#\VAR2#>UAR2#/VAR1#>UAR1#/\VAR2#=UAR2  
    ;  
    B12=0  
  ),  
  IUAR1 is IUAR+1,
distance_between2(  
    RUARS1,  
    RUARS2,  
    VAR1,  
    VAR2,  
    IVAR,  
    IUAR1,  
    =<,  
    S).
B.119  distance_change

◊ **Meta-Data:**

```prolog
ctr_date(
    distance_change,
    ['20000128','20030820','20040530','20060808']).

ctr_origin(distance_change,'Derived from %c.',[change]).

ctr_synonyms(distance_change,[distance]).

ctr_arguments(
    distance_change,
    ['DIST'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'CTR'-atom]).

ctr_restrictions(
    distance_change,
    ['DIST'>=0,
     'DIST'<size('VARIABLES1'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     size('VARIABLES1')=size('VARIABLES2'),
     in_list('CTR',[=,=\=,<,>,>=,<=])).

ctr_example(
    distance_change,
    distance_change(
        1,
        [[var-3],[var-3],[var-1],[var-2],[var-2]],
        [[var-4],[var-4],[var-3],[var-3],[var-3]],
        =\=)).

ctr_typical(
    distance_change,
    ['DIST'>$0,size('VARIABLES1')>1,in_list('CTR',[=,\=])).

ctr_exchangeable(
    distance_change,
    [args([['DIST'],['VARIABLES1','VARIABLES2'],['CTR']]),
     translate([['VARIABLES1'\'var]),
     translate([['VARIABLES2'\'var])].
```
ctr_graph(
    distance_change,
    [['VARIABLES1'], ['VARIABLES2']],
    2,
    ['PATH' >> collection(variables1, variables2)],
    ['CTR' (variables1 ^ var, variables2 ^ var)],
    ['DISTANCE' = 'DIST'],
    []). 

ctr_eval(
    distance_change,
    [reformulation(distance_change_r),
     automaton(distance_change_a)]).

ctr_pure_functional_dependency(distance_change, []). 

ctr_functional_dependency(distance_change, 1, [2, 3, 4]).

distance_change_r(DIST, VARIABLES1, VARIABLES2, CTR) :-
    collection(VARIABLES1, [dvar]),
    collection(VARIABLES2, [dvar]),
    length(VARIABLES1, L1),
    length(VARIABLES2, L2),
    L1 = L2,
    L is L1 - 1,
    check_type(dvar(0, L), DIST),
    memberchk(CTR, [=, =\, =\, <, =\, >, =\, <=]),
    get_attr1(VARIABLES1, VARS1),
    get_attr1(VARIABLES2, VARS2),
    distance_change1(VARS1, VARS2, CTR, TERM),
    call(DIST#=TERM).

distance_change1([], [], _28727, 0).

distance_change1([_28732], [_28734], _28730, 0) :- !.

distance_change1([UAR1, UAR2 | R], [VAR1, VAR2 | S], =, B12 + T) :- !,
    B12 #<=>
    UAR1#=UAR2#/\VAR1#=VAR2#/\VAR1#\=VAR2#/\VAR1#=VAR2,
    distance_change1([UAR2 | R], [VAR2 | S], =, T).

distance_change1([UAR1, UAR2 | R], [VAR1, VAR2 | S], =\, =, B12 + T) :- !,
    B12 #<=>
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

\[ UAR1 \neq UAR2 \wedge VAR1 \neq VAR2 \wedge UAR1 \neq UAR2 \wedge VAR1 \neq VAR2 , \]
\[ distance\_change1([UAR2|R],[VAR2|S],\neq,T). \]

\[ distance\_change1([UAR1,UAR2|R],[VAR1,VAR2|S],<,B12+T) :- !, \]
\[ B12 \neq UAR1 \neq UAR2 \wedge VAR1 \geq VAR2 \wedge UAR1 \geq UAR2 \wedge VAR1 \leq VAR2 , \]
\[ distance\_change1([UAR2|R],[VAR2|S],<,T). \]

\[ distance\_change1([UAR1,UAR2|R],[VAR1,VAR2|S],\geq,B12+T) :- !, \]
\[ B12 \neq UAR1 \neq UAR2 \wedge VAR1 \leq VAR2 \wedge UAR1 \leq UAR2 \wedge VAR1 \geq VAR2 , \]
\[ distance\_change1([UAR2|R],[VAR2|S],\geq,T). \]

\[ distance\_change1([UAR1,UAR2|R],[VAR1,VAR2|S],>,B12+T) :- !, \]
\[ B12 \neq UAR1 \neq UAR2 \wedge VAR1 < VAR2 \wedge UAR1 < UAR2 \wedge VAR1 > VAR2 , \]
\[ distance\_change1([UAR2|R],[VAR2|S],>,T). \]

\[ distance\_change1([UAR1,UAR2|R],[VAR1,VAR2|S],=<,B12+T) :- \]
\[ B12 \neq UAR1 \neq UAR2 \wedge VAR1 > VAR2 \wedge UAR1 > UAR2 \wedge VAR1 \leq VAR2 , \]
\[ distance\_change1([UAR2|R],[VAR2|S],=<,T). \]

\[ distance\_change\_a(FLAG,DIST,VARIABLES1,VARIABLES2,CTR) :- \]
\[ collection(VARIABLES1,[dvar]), \]
\[ collection(VARIABLES2,[dvar]), \]
\[ length(VARIABLES1,L1), \]
\[ length(VARIABLES2,L2), \]
\[ L1 = L2, \]
\[ L is L1-1, \]
\[ check\_type(dvar(0,L),DIST), \]
\[ memberchk(CTR,[=,\neq,\leq,\geq,\lt,\gt]), \]
\[ distance\_change\_signature( \]
\[ VARIABLES1, \]
\[ VARIABLES2, \]
\[ SIGNATURE, \]
\[ CTR), \]
\[ automaton( \]
\[ SIGNATURE, \]
\[ _31244, \]
\[ SIGNATURE, \]
\[ [source(s),sink(s)], \]
\[ [arc(s,0,s),arc(s,1,s,[C+1])]. \]
distance_change_signature([],[],[],_28728).

distance_change_signature([_28732],[_28734],[],_28731) :- !.

distance_change_signature([var-VAR1i], [var-VAR1j]|VAR1s,
[[var-VAR2i], [var-VAR2j]|VAR2s],
[S|Ss],
=) :- !,
VAR1i#/=VAR1j#/VAR2i#/=VAR2j#
VAR1i#/=VAR1j#/VAR2i#=VAR2j#<=>
S,
distance_change_signature([[var-VAR1j]|VAR1s],
[[var-VAR2j]|VAR2s],
Ss,
=).

distance_change_signature([var-VAR1i], [var-VAR1j]|VAR1s,
[[var-VAR2i], [var-VAR2j]|VAR2s],
[S|Ss],
=\=) :- !,
VAR1i#/=VAR1j#/VAR2i#=VAR2j#
VAR1i#/=VAR1j#/VAR2i#=VAR2j#<=>
S,
distance_change_signature([[var-VAR1j]|VAR1s],
[[var-VAR2j]|VAR2s],
Ss,
=\=).

distance_change_signature([var-VAR1i], [var-VAR1j]|VAR1s,
[[var-VAR2i], [var-VAR2j]|VAR2s],
[S|Ss],
<) :- !,
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

\[
\begin{align*}
\text{distance_change_signature}\left( & \left[\text{var-VAR1}_j, \text{var-VAR1}_i\right] | \text{VAR1}_s, \\
& \left[\text{var-VAR2}_j, \text{var-VAR2}_i\right] | \text{VAR2}_s, \\
& S, \\
& S_s, \\
& \right) \right) :- \\
& !, \\
& \text{VAR1}_i \geq \text{VAR1}_j \lor \text{VAR2}_i < \text{VAR2}_j \lor \\
& \text{VAR1}_i < \text{VAR1}_j \lor \text{VAR2}_i \geq \text{VAR2}_j \right) \right) \\
& \text{distance_change_signature}\left( & \left[\text{var-VAR1}_j, \text{var-VAR1}_i\right] | \text{VAR1}_s, \\
& \left[\text{var-VAR2}_j, \text{var-VAR2}_i\right] | \text{VAR2}_s, \\
& S, \\
& S_s, \\
& >\right) :- \\
& !, \\
& \text{VAR1}_i > \text{VAR1}_j \lor \text{VAR2}_i \leq \text{VAR2}_j \lor \\
& \text{VAR1}_i \leq \text{VAR1}_j \lor \text{VAR2}_i > \text{VAR2}_j \right) \right) \\
& \text{distance_change_signature}\left( & \left[\text{var-VAR1}_j, \text{var-VAR1}_i\right] | \text{VAR1}_s, \\
& \left[\text{var-VAR2}_j, \text{var-VAR2}_i\right] | \text{VAR2}_s, \\
& S, \\
& S_s, \\
& =\right) :- \\
& !, \\
& \text{VAR1}_i \leq \text{VAR1}_j \lor \text{VAR2}_i > \text{VAR2}_j \lor \\
& \text{VAR1}_i > \text{VAR1}_j \lor \text{VAR2}_i \leq \text{VAR2}_j \right) \right)
\end{align*}
\]
VAR1i#→VAR1j#/VAR2i#=<VAR2j#<=>S,
distance_change_signature(
    [[var-VAR1j]|VAR1s],
    [[var-VAR2j]|VAR2s],
    Ss,
    =<).
B.120 divisible

◊ Meta-Data:

ctr_predefined(divisible).
ctr_date(divisible,['20110612']).
ctr_origin(divisible,'Arithmetic.',[]).
ctr_synonyms(divisible,[div]).
ctr_arguments(divisible,['Q'-dvar,'D'-dvar]).
ctr_restrictions(divisible,['Q'>=0,'D'>0]).
ctr_example(divisible,divisible(12,4)).
ctr_typical(divisible,['Q'>1,'D'<'Q']).
ctr_eval(divisible,[builtin(divisible_b)]).

divisible_b(Q,D) :-
    check_type(dvar,Q),
    check_type(dvar,D),
    Q#>=0,
    D#>0,
    Q mod D#=0.
B.121  divisible_or

◊ Meta-Data:

ctr_predefined(divisible_or).

ctr_date(divisible_or, ['20120212']).

ctr_origin(divisible_or, 'Arithmetic.', []).

ctr_synonyms(divisible_or, [div_or]).

ctr_arguments(divisible_or, ['C'-dvar, 'D'-dvar]).

ctr_restrictions(divisible_or, ['C'>0, 'D'>0]).

ctr_example(divisible_or, divisible_or(4, 12)).

ctr_eval(divisible_or, [builtin(divisible_or_b)]).

divisible_or_b(C, D) :-
    check_type(dvar, C),
    check_type(dvar, D),
    C#>0,
    D#>0,
    C mod D#=0\D mod C#=0.
B.122 dom_reachability

◊ Metadata:

ctr_predefined(dom_reachability).

ctr_date(dom_reachability,[‘20061011’]).

ctr_origin(
    dom_reachability,
    cite{QuesadaVanRoyDevilleCollet06}, []).

ctr_arguments(
    dom_reachability,
    [‘SOURCE’-int,
     ‘FLOW_GRAPH’-collection(index-int,succ-svar),
     ‘DOMINATOR_GRAPH’-collection(index-int,succ-sint),
     TRANSITIVE_CLOSURE_GRAPH-
     collection(index-int,succ-svar)]).

ctr_restrictions(
    dom_reachability,
    [‘SOURCE’>=1,
     ‘SOURCE’=<size(‘FLOW_GRAPH’),
     required(‘FLOW_GRAPH’,[index,succ]),
     ‘FLOW_GRAPH’\index>=1,
     ‘FLOW_GRAPH’\index=<size(‘FLOW_GRAPH’),
     ‘FLOW_GRAPH’\succ>=1,
     ‘FLOW_GRAPH’\succ=<size(‘FLOW_GRAPH’),
     distinct(‘FLOW_GRAPH’,index),
     required(‘DOMINATOR_GRAPH’,[index,succ]),
     size(‘DOMINATOR_GRAPH’)=size(‘FLOW_GRAPH’),
     ‘DOMINATOR_GRAPH’\index>=1,
     ‘DOMINATOR_GRAPH’\index=<size(‘DOMINATOR_GRAPH’),
     ‘DOMINATOR_GRAPH’\succ>=1,
     ‘DOMINATOR_GRAPH’\succ=<size(‘DOMINATOR_GRAPH’),
     distinct(‘DOMINATOR_GRAPH’,index),
     required(‘TRANSITIVE_CLOSURE_GRAPH’,[index,succ]),
     size(‘TRANSITIVE_CLOSURE_GRAPH’)=size(‘FLOW_GRAPH’),
     ‘TRANSITIVE_CLOSURE_GRAPH’\index>=1,
     ‘TRANSITIVE_CLOSURE_GRAPH’\index=<
     size(‘TRANSITIVE_CLOSURE_GRAPH’),
     ‘TRANSITIVE_CLOSURE_GRAPH’\succ>=1,
     ‘TRANSITIVE_CLOSURE_GRAPH’\succ=<
     size(‘TRANSITIVE_CLOSURE_GRAPH’),
     ‘TRANSITIVE_CLOSURE_GRAPH’\succ=1]
)
distinct('TRANSITIVE_CLOSURE_GRAPH',index)).

ctr_example(
    dom_reachability,
    dom_reachability(
        1,
        [[index-1,succ-{2}],
         [index-2,succ-{3,4}],
         [index-3,succ-{}],
         [index-4,succ-{}]],
        [[index-1,succ-{2,3,4}],
         [index-2,succ-{3,4}],
         [index-3,succ-{}],
         [index-4,succ-{}]],
        [[index-1,succ-{1,2,3,4}],
         [index-2,succ-{2,3,4}],
         [index-3,succ-{3}],
         [index-4,succ-{4}]]).

ctr_typical(dom_reachability,[size('FLOW_GRAPH')>2]).

ctr_exchangeable(
    dom_reachability,
    [items('FLOW_GRAPH',all),
     items('DOMINATOR_GRAPH',all),
     items('TRANSITIVE_CLOSURE_GRAPH',all)]).
B.123 domain

◊ Meta-Data:

ctr_predefined(domain).

ctr_date(domain,['20070821']).

ctr_origin(domain,'Domain definition.',[]).

ctr_synonyms(domain,[dom]).

ctr_arguments(
    domain,
    ['VARIABLES'-collection(var-dvar),'LOW'-int,'UP'-int]).

ctr_restrictions(
    domain,
    [required('VARIABLES',var), 'LOW'='='='UP']).

ctr_example(domain,domain([[var-2],[var-8],[var-2]],1,9)).

ctr_typical(domain,[size('VARIABLES')>1,'LOW'='='='UP']).

ctr_exchangeable(
    domain,
    [items('VARIABLES',all),
     vals(
         ['VARIABLES'\var],
         int('LOW' in 'UP'),
         =\=,
         dontcare,
         dontcare),
     vals(['LOW'],int,>,dontcare,dontcare),
     vals(['UP'],int,<,dontcare,dontcare),
     translate([['VARIABLES'\var,'LOW','UP']])).

ctr_eval(domain,[builtin(domain_b)]).

ctr_contractible(domain,[],'VARIABLES',any).

domain_b(VARIABLES,LOW,UP) :-
    check_type(int,LOW),
    check_type(int,UP),
    LOW=<UP,
    collection(VARIABLES,[fdvar(LOW,UP)])},
get_attr1(VARIABLES,VARS),
domain(VARS,LOW,UP).
B.124  domain_constraint

◇ Meta-Data:

ctr_date(domain_constraint,[‘20030820’,’20040530’,’20060808’]).

ctr_origin(domain_constraint,’\cite{Refalo00}’,[]).

ctr_synonyms(domain_constraint,[domain]).

ctr_arguments(
  domain_constraint,
  [‘VAR’–dvar,’VALUES’–collection(var01–dvar,value–int)])).

ctr_restrictions(
  domain_constraint,
  [required(‘VALUES’,[var01,value]),
   ‘VALUES’ˆvar01>=0,
   ‘VALUES’ˆvar01=<1,
   distinct(‘VALUES’,value)]).

ctr_example(
  domain_constraint,
  domain_constraint(5,
    [[var01-0,value-9],
    [var01-1,value-5],
    [var01-0,value-2],
    [var01-0,value-7]])).

ctr_typical(domain_constraint,[size(‘VALUES’)>1]).

ctr_exchangeable(domain_constraint,[items(‘VALUES’,all)]).

ctr_derived_collections(
  domain_constraint,
  [col(‘VALUE’–collection(var01–int,value–dvar),
    [item(var01-1,value–’VAR’)])]).

ctr_graph(
  domain_constraint,
  [‘VALUE’,’VALUES’],
  2,
  [’PRODUCT’>>collection(value,values)],
  [valueˆvalue=valueˆvaluesˆvalue#<=>valuesˆvar01=1],
  [’NARC’=size(‘VALUES’)],
ctr_eval(
    domain_constraint,
    [reformulation(domain_constraint_r),
      automaton(domain_constraint_a)]).

domain_constraint_r(VAR,VALUES) :-
    check_type(dvar,VAR),
    collection(VALUES, [dvar(0,1),int]),
    get_attr1(VALUES, VARS01),
    get_attr2(VALUES, VALS),
    all_different(VALS),
    domain_constraint1(VARS01, VALS, VAR, Term),
    call(Term).

domain_constraint1([], [], _35825, 0).

   domain_constraint1(
      [VAR01|R],
      [VAL|S],
      VAR,
      VAR#=VAL#/VAR01#=1#/T) :-
      domain_constraint1(R, S, VAR, T).

domain_constraint_a(FLAG, VAR, VALUES) :-
    check_type(dvar, VAR),
    collection(VALUES, [dvar(0,1),int]),
    get_attr2(VALUES, VALS),
    all_different(VALS),
    domain_constraint_signature(VALUES, SIGNATURE, VAR),
    AUTOMATON=
    automaton(
        SIGNATURE, _37464,
        SIGNATURE, [source(s), sink(s)],
        [arc(s,1,s)], [], [], [],
        automaton_bool(FLAG, [0,1], AUTOMATON).

domain_constraint_signature([], [], _35825).

domain_constraint_signature(}
[[\text{var01-VAR01}, \text{value-VALUE}] | \text{VALUES}],
[\text{S}| \text{Ss}],
\text{VAR}) :-
\text{VAR} \#= \text{VALUE} \# \leftarrow \text{VAR01} \# \leftarrow \text{S},
\text{domain_constraint_signature(VALUES, Ss, VAR).}
B.125  elem

◊ META-DATA:

ctr_date(elem,[‘20030820’,‘20040530’,‘20060808’]).

ctr_origin(elem,’Derived from %c.’,[element]).

ctr_usual_name(elem,element).

ctr_synonyms(elem,[nth,array]).

ctr_arguments(
  elem,
  [‘ITEM’-collection(index-dvar,value-dvar),
   ‘TABLE’-collection(index-int,value-dvar)]).

ctr_restrictions(
  elem,
  [required(‘ITEM’,[index,value]),
   ‘ITEM’^index>=1,
   ‘ITEM’^index=<size(‘TABLE’),
   size(‘ITEM’) =1,
   size(‘TABLE’) >0,
   required(‘TABLE’,[index,value]),
   ‘TABLE’^index>=1,
   ‘TABLE’^index=<size(‘TABLE’),
   distinct(‘TABLE’,index)]).

ctr_example(
  elem,
  elem(
    [[index-3,value-2]],
    [[index-1,value-6],
     [index-2,value-9],
     [index-3,value-2],
     [index-4,value-9]])).

ctr_typical(elem,[size(‘TABLE’) >1,range(‘TABLE’^value)>1]).

ctr_exchangeable(
  elem,
  [items(‘TABLE’,all),
   vals([‘ITEM’^value,’TABLE’^value],int,=\=,all,dontcare)]).

ctr_graph(}
elem,
  ['ITEM','TABLE'],
2,
  ['PRODUCT'>>collection(item,table)],
  [item^index=table^index,item^value=table^value],
  ['NARC'=1],
[]).

ctr_eval(elem,[builtin(elem_b),automaton(elem_a)]).

ctr_pure_functional_dependency(elem,[]).

ctr_functional_dependency(elem,1-2,[1-1,2-2]).

elem_b(ITEM,TABLE) :-
  length(ITEM,1),
  length(TABLE,N),
  collection(ITEM,[dvar(1,N),dvar]),
  collection(TABLE,[int(1,N),dvar]),
  get_attr1(ITEM,[INDEX]),
  get_attr2(ITEM,[VALUE]),
  get_attr1(TABLE,INDEXES),
  get_attr2(TABLE,VALUES),
  all_different(INDEXES),
  element(INDEX,VALUES,VALUE).

elem_a(FLAG,ITEM,TABLE) :-
  length(ITEM,1),
  length(TABLE,N),
  collection(ITEM,[dvar(1,N),dvar]),
  collection(TABLE,[int(1,N),dvar]),
  get_attr1(TABLE,INDEXES),
  all_different(INDEXES),
  ITEM=[[index-ITEM_INDEX,value-ITEM_VALUE]],
  elem_signature(TABLE,SIGNALTURE,ITEM_INDEX,ITEM_VALUE),
  AUTOMATON=automaton(
    SIGNALTURE,
    _44555,
    SIGNALTURE,
    [source(s),sink(t)],
    [arc(s,0,s),arc(s,1,t),arc(t,0,t),arc(t,1,t)],
    [],
    [],
    []),
  automaton_bool(FLAG,[0,1],AUTOMATON).
elem_signature([],[],_42235,_42236).

elem_signature(
    [[index-TABLE_INDEX,value-TABLE_VALUE]|TABLEs],
    [S|Ss],
    ITEM_INDEX,
    ITEM_VALUE)
    :-
        ITEM_INDEX# = TABLE_INDEX# \ ITEM_VALUE# = TABLE_VALUE# <= > S, 
        elem_signature(TABLEs,Ss,ITEM_INDEX,ITEM_VALUE).
B.126 elem_from_to

◊ Meta-Data:

ctr_date(elem_from_to, ['20091115']).

ctr_origin(elem_from_to, 'Derived from %c.', [elem]).

ctr_synonyms(elem_from_to, [element_from_to]).

ctr_arguments(
    elem_from_to,
    [ITEM-
        collection(
            from-dvar,
            cst_from-int,
            to-dvar,
            cst_to-int,
            value-dvar),
        'TABLE'-collection(index-int, value-dvar)]).

ctr_restrictions(
    elem_from_to,
    [required('ITEM', [from, cst_from, to, cst_to, value]),
        'ITEM'°from>=1,
        'ITEM'°from=<size('TABLE'),
        'ITEM'°to>=1,
        'ITEM'°to=<size('TABLE'),
        'ITEM'°from=<ITEM°to,
        size('ITEM')=1,
        required('TABLE', [index, value]),
        'TABLE'°index>=1,
        'TABLE'°index=<size('TABLE'),
        increasing_seq('TABLE', [index])]).

ctr_example(
    elem_from_to,
    elem_from_to(
        [[from-1, cst_from-1, to-4, cst_to- -1, value-2]],
        [[index-1, value-6],
        [index-2, value-2],
        [index-3, value-2],
        [index-4, value-9],
        [index-5, value-9]])).
elem_from_to,
['ITEM'\textasciitilde cst_from>0,
'ITEM'\textasciitilde cst_from=<1,
'ITEM'\textasciitilde cst_to> -1,
'ITEM'\textasciitilde cst_to=<1,
size('TABLE')>1,
range('TABLE'\textasciitilde value)>1]).

ctr_exchangeable(
  elem_from_to,
  [vals(['ITEM'\textasciitilde value,'TABLE'\textasciitilde value],int,=\textasciitilde all,dont care)]).

ctr_eval(elem_from_to,[automaton(elem_from_to_a)]).

elem_from_to_a(FLAG,ITEM,TABLE) :-
  length(TABLE,N),
  collection(ITEM,[dvar(1,N),int,dvar(1,N),int,dvar]),
  collection(TABLE,[int(1,N),dvar]),
  collection_increasing_seq(TABLE,[1]),
  ITEM= [['from-FROM,
cst_from-CST_FROM,
to-TO,
cst_to-CST_TO,
value-VALUE]],
  FROM#=<TO,
  F#=max(1,FROM+CST_FROM),
  T#=min(N,TO+CST_TO),
  elem_from_to_signature(
    TABLE,
    SIGNATURE,
    N,
    FROM,
    TO,
    F,
    T,
    VALUE),
  AUTOMATON=
  automaton(
    SIGNATURE,
    _24943,
    SIGNATURE,
    [source(s),sink(s)],
    [arc(s,0,s),arc(s,1,s),arc(s,2,s),arc(s,3,s)],
    [],
    [],
    ]
}
elem_from_to_signature([], [], _20482, _20528, _20574, _20620, _20666, _20712).

elem_from_to_signature([[index-TABLE_INDEX,value-TABLE_VALUE]|TABLEs], [S|Ss], N, FROM, TO, F, T, VALUE) :-
    S in 0..4,
    1#=<FROM#/\FROM#=<TO#/\TO#=<N#/\F#=<T#/\F#=TABLE_INDEX#<=>
    S#=0,
    1#=<FROM#/\FROM#=<TO#/\TO#=<N#/\F#<T#/\F#=<TABLE_INDEX#/\TABLE_INDEX#=<T#/\TABLE_INDEX=#<T#/\VALUE#=TABLE_VALUE#<=>
    S#=1,
    1#=<FROM#/\FROM#=<TO#/\TO#=<N#/\F#=<T#/\F#=TABLE_INDEX#/\TABLE_INDEX#=<T#/\TABLE_INDEX=#<T#/\VALUE#=TABLE_VALUE#<=>
    S#=2,
    1#=<FROM#/\FROM#=<TO#/\TO#=<N#/\F#=<T#/\F#=TABLE_INDEX#/\TABLE_INDEX#=<T#/\TABLE_INDEX=#<T#/\VALUE#=TABLE_VALUE#<=>
    S#=3,
    1#=<FROM#/\FROM#=<TO#/\TO#=<N#/\F#=<T#/\F#=TABLE_INDEX#/\TABLE_INDEX#=<T#/\TABLE_INDEX=#<T#/\VALUE#=TABLE_VALUE#<=>
    S#=4,
    elem_from_to_signature(TABLEs,Ss,N,FROM,TO,F,T,VALUE).
B.127 element

◊ **META-DATA:**

ctr_date(element,
          ['20000128','20030820','20040530','20060808','20090923']).

ctr_origin(element,'\cite{VanHentenryckCarillon88}',[]).

ctr_synonyms(element,[nth,element_var,array]).

ctr_arguments(element,
              ['INDEX'-dvar,'TABLE'-collection(value-dvar),'VALUE'-dvar]).

ctr_restrictions(element,
                ['INDEX'>=1,
                 'INDEX'=<size('TABLE'),
                 size('TABLE')>0,
                 required('TABLE',value)]).

ctr_example(element,
             element(3,[[value-6],[value-9],[value-2],[value-9]],2)).

ctr_typical(element,[size('TABLE')>1,range('TABLE'\^value)>1]).

ctr_exchangeable(element,
                 [vals(['TABLE'\^value,'VALUE'],int,=\=,all,dontcare)]).

ctr_derived_collections(element,
                          [col('ITEM'-collection(index-dvar,value-dvar),
                           [item(index-'INDEX',value-'VALUE')])]).

ctr_graph(element,
          ['ITEM','TABLE'],
          2,
          ['PRODUCT'>>collection(item,table)],
          [item^index=table^key,item^value=table^value],
          ['NARC'=1],
          []).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

ctr_eval(element,[builtin(element_b),automaton(element_a)]).

ctr_pure_functional_dependency(element,[]).

ctr_functional_dependency(element,3,[1,2]).

ctr_extensible(element,[],'TABLE',suffix).

element_b(INDEX,TABLE,VALUE) :-
  check_type(dvar,INDEX),
  collection(TABLE,[dvar]),
  check_type(dvar,VALUE),
  length(TABLE,N),
  N>0,
  INDEX#>=1,
  INDEX#=<N,
  get_attr1(TABLE,VALUES),
  element(INDEX,VALUES,VALUE).

element_a(FLAG,INDEX,TABLE,VALUE) :-
  check_type(dvar,INDEX),
  collection(TABLE,[dvar]),
  check_type(dvar,VALUE),
  length(TABLE,N),
  N>0,
  INDEX#>=1,
  INDEX#=<N,
  element_signature(TABLE,INDEX,VALUE,1,SIGNATURE),
  AUTOMATON=automaton(
    SIGNATURE,_47234,
    SIGNATURE,
    [source(s),sink(t)],
    [arc(s,0,s),arc(s,1,t),arc(t,0,t),arc(t,1,t)],
    [],
    [],
    []),
  automaton_bool(FLAG,[0,1],AUTOMATON).

element_signature([],_44978,_44979,_44980,[]).

element_signature(
  [[value-TABLE_VALUE]|Ts],
  INDEX,
VALUE,
TABLE_KEY,
[B|Bs]) :-
    INDEX#=TABLE_KEY#\VALUE#=TABLE_VALUE#<=>B,
    TABLE_KEY1 is TABLE_KEY+1,
    element_signature(Ts,INDEX,VALUE,TABLE_KEY1,Bs).
B.128  element_greatereq

◊  **Meta-Data:**

```prolog
ctr_date(element_greatereq, ['20030820', '20040530', '20060808']).

ctr_origin(
    element_greatereq,
    cite{OttossonThorsteinssonHooker99}, []).

ctr_arguments(
    element_greatereq,
    ['ITEM'-collection(index-dvar, value-dvar),
     'TABLE'-collection(index-int, value-int)]).

ctr_restrictions(
    element_greatereq,
    [required('ITEM', [index, value]),
     'ITEM'\u207b\u00b4index>=1,
     'ITEM'\u207b\u00b4index=<size('TABLE'),
     size('ITEM')=1,
     size('TABLE')>0,
     required('TABLE', [index, value]),
     'TABLE'\u207b\u00b4index>=1,
     'TABLE'\u207b\u00b4index=<size('TABLE'),
     distinct('TABLE', index)]).

ctr_example(
    element_greatereq,
    element_greatereq(
        [[index-1, value-8]],
        [[index-1, value-6],
         [index-2, value-9],
         [index-3, value-2],
         [index-4, value-9]]).

ctr_typical(
    element_greatereq,
    [size('TABLE')>1, range('TABLE'\u207b\u00b4value)>1]).

ctr_exchangeable(
    element_greatereq,
    [items('TABLE', all),
     vals([\u0027ITEM\u207b\u00b4value, \u0027TABLE\u207b\u00b4value], int, =\\u003d, all, dontcare)]).
```
ctr_graph(
  element_greatereq,
  ['ITEM','TABLE'],
  2,
  ['PRODUCT'>>collection(item,table)],
  [item\^index=table\^index,item\^value>=table\^value],
  ['NARC'=1],
  []).

ctr_eval(
  element_greatereq,
  [reformulation(element_greatereq_r),
   automaton(element_greatereq_a)]).

element_greatereq_r(ITEM,TABLE) :-
  length(ITEM,1),
  length(TABLE,N),
  N>0,
  collection(ITEM,[dvar(1,N),dvar]),
  collection(TABLE,[int(1,N),dvar]),
  get_attr1(ITEM,[INDEX]),
  get_attr2(ITEM,[VALUE]),
  get_attr1(TABLE,INDEXES),
  get_attr2(TABLE,VALUES),
  all_different(INDEXES),
  element(INDEX,VALUES,VAL),
  VALUE#>=VAL.

element_greatereq_a(FLAG,ITEM,TABLE) :-
  length(ITEM,1),
  length(TABLE,N),
  N>0,
  collection(ITEM,[dvar(1,N),dvar]),
  collection(TABLE,[int(1,N),dvar]),
  get_attr1(TABLE,INDEXES),
  all_different(INDEXES),
  ITEM=[[index-ITEM_INDEX,value-ITEM_VALUE]],
  element_greatereq_signature(
    TABLE,
    SIGNATURE,
    ITEM_INDEX,
    ITEM_VALUE),
  AUTOMATON=
  automaton(
    SIGNATURE,
    _36545,
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

SIGNATURE,
[source(s), sink(t)],
[arc(s,0,s), arc(s,1,t), arc(t,0,t), arc(t,1,t)],
[],
[],
[],

automaton_bool(FLAG, [0,1], AUTOMATON).

element_greatereq_signature([], [], _33731, _33732).

element_greatereq_signature(
    [[index-TABLE_INDEX, value-TABLE_VALUE]|TABLEs],
    [S|Ss],
    ITEM_INDEX,
    ITEM_VALUE) :-
    \ITEM_INDEX#=TABLE_INDEX//\ITEM_VALUE#>=TABLE_VALUE#<>S,
    element_greatereq_signature(
        TABLEs,
        Ss,
        ITEM_INDEX,
        ITEM_VALUE).

B.129  element_lesseq

◊ Meta-Data:

ctr_date(element_lesseq, ['20030820', '20040530', '20060808']).

ctr_origin(
    element_lesseq,
    \cite{OttossonThorsteinssonHooker99},
    []).

ctr_arguments(
    element_lesseq,
    ['ITEM'-collection(index-dvar,value-dvar),
     'TABLE'-collection(index-int,value-int)]).

ctr_restrictions(
    element_lesseq,
    [required('ITEM', [index,value]),
     'ITEM'\index>=1,
     'ITEM'\index=<size('TABLE'),
     size('ITEM')=1,
     size('TABLE')>0,
     required('TABLE', [index,value]),
     'TABLE'\index>=1,
     'TABLE'\index=<size('TABLE'),
     distinct('TABLE',index)]).

ctr_example(
    element_lesseq,
    element_lesseq(
        [[index-3,value-1]],
        [[index-1,value-6],
         [index-2,value-9],
         [index-3,value-2],
         [index-4,value-9]]).

ctr_typical(
    element_lesseq,
    [size('TABLE')>1, range('TABLE'\value)>1]).

ctr_exchangeable(
    element_lesseq,
    [items('TABLE', all),
     vals(['ITEM'\value,'TABLE'\value],int,=\=,all,dontcare)]).
ctr_graph(
    element_lesseq,
    ['ITEM','TABLE'],
    2,
    ['PRODUCT'>>collection(item,table)],
    [item\^index=table\^index,item\^value=<table\^value],
    ['NARC'=1],
    []).

ctr_eval(
    element_lesseq,
    [reformulation(element_lesseq_r),
      automaton(element_lesseq_a)]).

element_lesseq_r(ITEM,TABLE) :-
    length(ITEM,1),
    length(TABLE,N),
    N>0,
    collection(ITEM,[dvar(1,N),dvar]),
    collection(TABLE,[int(1,N),dvar]),
    get_attr1(ITEM,[INDEX]),
    get_attr2(ITEM,[VALUE]),
    get_attr1(TABLE,INDEXES),
    get_attr2(TABLE,VALUES),
    all_different(INDEXES),
    element(INDEX,VALUES,VAL),
    VALUE#=<VAL.

element_lesseq_a(FLAG,ITEM,TABLE) :-
    length(ITEM,1),
    length(TABLE,N),
    N>0,
    collection(ITEM,[dvar(1,N),dvar]),
    collection(TABLE,[int(1,N),dvar]),
    get_attr1(TABLE,INDEXES),
    all_different(INDEXES),
    ITEM=[[index-ITEM_INDEX,value-ITEM_VALUE]],
    element_lesseq_signature(
        TABLE,
        SIGNATURE,
        ITEM_INDEX,
        ITEM_VALUE),
    AUTOMATON=automaton(
        SIGNATURE,
        _36443,
SIGNATURE,
[source(s), sink(t)],
[arc(s,0,s), arc(s,1,t), arc(t,0,t), arc(t,1,t)],
[],
[],
[]),
automaton_bool(FLAG, [0, 1], AUTOMATON).
element_lesseq_signature([], [], _33629, _33630).
element_lesseq_signature([[[index-TABLE_INDEX, value-TABLE_VALUE]|TABLEs],
[S|Ss],
ITEM_INDEX,
ITEM_VALUE]) :-
  ITEM_INDEX#=TABLE_INDEX#/ITEM_VALUE#=<TABLE_VALUE#<=>S,
element_lesseq_signature(
    TABLEs,
    Ss,
    ITEM_INDEX,
    ITEM_VALUE).
B.130 \textbf{element\_matrix}

\textbf{Meta-Data:}

\begin{verbatim}
ctr_date(element_matrix, ['20031101', '20060808']).
ctr_origin(element_matrix, '\\index{CHIP\indexuse}CHIP', []).
ctr_synonyms(element_matrix, [elem_matrix, matrix]).
ctr_arguments(element_matrix, ['MAX\_I'-int, 'MAX\_J'-int, 'INDEX\_I'\text{-dvar}, 'INDEX\_J'\text{-dvar}, 'MATRIX'\text{-collection}(i\text{-int}, j\text{-int}, v\text{-int}), 'VALUE'\text{-dvar}]).
ctr_restrictions(element_matrix, ['MAX\_I'>=1, 'MAX\_J'>=1, 'INDEX\_I'>=1, 'INDEX\_I'=<'MAX\_I', 'INDEX\_J'>=1, 'INDEX\_J'=<'MAX\_J', required('MATRIX', [i, j, v]), increasing_seq('MATRIX', [i, j]), 'MATRIX'\text{^i}>=1, 'MATRIX'\text{^i}=<'MAX\_I', 'MATRIX'\text{^j}>=1, 'MATRIX'\text{^j}=<'MAX\_J', size('MATRIX')='MAX\_I'\text{\^*}MAX\_J').
ctr_example(element_matrix, element_matrix(4, 3, 1, 3), [([i-1, j-1, v-4], [i-1, j-2, v-1], [i-1, j-3, v-7], [i-2, j-1, v-1], [i-1, j-1, v-4]), [i-1, j-2, v-1], [i-1, j-3, v-7], [i-2, j-1, v-1], [i-1, j-1, v-4]), [i-1, j-2, v-1], [i-1, j-3, v-7], [i-2, j-1, v-1]), [i-1, j-1, v-4], [i-1, j-2, v-1], [i-1, j-3, v-7], [i-2, j-1, v-1], [i-1, j-1, v-4])].
\end{verbatim}
\[
\begin{align*}
\text{ctr\_typical} & (\text{element\_matrix}, \\
& ['\text{MAX\_I'}>1, \\
& '\text{MAX\_J'}>1, \\
& \text{size('MATRIX')}>3, \\
& \text{maxval('MATRIX' }^i)>1, \\
& \text{maxval('MATRIX' }^j)>1, \\
& \text{range('MATRIX' }^v)>1]).
\end{align*}
\]

\[
\text{ctr\_exchangeable} (\text{element\_matrix}, \\
\text{vals(['MATRIX' }^v,'VALUE'],\text{int,}=\text{all,dontcare})).
\]

\[
\text{ctr\_derived\_collections} (\text{element\_matrix}, \\
\text{col(ITEM-} \\
\text{collection(index\_i-dvar,index\_j-dvar,value-dvar),} \\
\text{[item(} \\
\text{index\_i-'INDEX\_I',} \\
\text{index\_j-'INDEX\_J',} \\
\text{value-'VALUE'})])))
\]

\[
\text{ctr\_graph} (\text{element\_matrix}, \\
['\text{ITEM'},'\text{MATRIX'}], \\
2, \\
['\text{PRODUCT'}\text{>>}\text{collection(item,matrix)}], \\
\text{[item'}^\text{index\_i=matrix'}^i,} \\
\text{item'}^\text{index\_j=matrix'}^j,} \\
\text{item'}^\text{value=matrix'}^v], \\
['\text{NARC'}=1],} \\
[{}].
\]

\[
\text{ctr\_eval} (\text{element\_matrix}, \\
\text{[reformulation(element\_matrix}_r),} \\
\text{7}).
\]
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

element_matrix_r(MAX_I,MAX_J,INDEX_I,INDEX_J,MATRIX,VALUE) :-
    check_type(int,MAX_I),
    MAX_I>=1,
    check_type(int,MAX_J),
    MAX_J>=1,
    check_type(dvar,INDEX_I),
    INDEX_I#>=1,
    INDEX_I#==<MAX_I,
    check_type(dvar,INDEX_J),
    INDEX_J#=1,
    INDEX_J#==<MAX_J,
    collection(MATRIX,[int(1,MAX_I),int(1,MAX_J),int]),
    length(MATRIX,N),
    N is MAX_I*MAX_J,
    collection_increasing_seq(MATRIX,[1,2]),
    check_type(dvar,VALUE),
    get_attr3(MATRIX,VALUES),
    element_matrix1(MAX_I,MAX_J,INDEX_J,VALUES,TABLE_VARS),
    element(INDEX_I,TABLE_VARS,VALUE).

element_matrix1(0,40963,40964,40965,[]):- !.

element_matrix1(I,MAX_J,INDEX_J,VALUES,[V_J|R]) :-
    I>0,
    element_matrix2(MAX_J,VALUES,TABLE_VARS,REST_VALUES),
    element(INDEX_J,TABLE_VARS,V_J),
    I1 is I-1,
    element_matrix1(I1,MAX_J,INDEX_J,REST_VALUES,R).

element_matrix2(0,VALUES,[],VALUES) :- !.

element_matrix2(J,[V|R],[V|S],REST_VALUES) :-
    J>0,
    J1 is J-1,
    element_matrix2(J1,R,S,REST_VALUES).

element_matrix_a(FLAG,MAX_I,MAX_J,INDEX_I,INDEX_J,MATRIX,VALUE) :-
    check_type(int,MAX_I),
    MAX_I>=1,
    check_type(int,MAX_J),
    MAX_J>=1,
    check_type(dvar,INDEX_I),
INDEX_I#>=1,  
INDEX_I#=<MAX_I,  
check_type(dvar,INDEX_J),  
INDEX_J#>=1,  
INDEX_J#=<MAX_J,  
collection(MATRIX,[int(1,MAX_I),int(1,MAX_J),int]),  
length(MATRIX,N),  
N is MAX_I*MAX_J,  
collection_increasing_seq(MATRIX,[1,2]),  
check_type(dvar,VALUE),  
element_matrix_signature(  
  MATRIX,  
  INDEX_I,  
  INDEX_J,  
  VALUE,  
  SIGNATURE),  
AUTOMATON=  
automaton(  
  SIGNATURE,  
_45037,  
  SIGNATURE,  
  [source(s),sink(t)],  
  [arc(s,0,s),arc(s,1,t),arc(t,0,t),arc(t,1,t)],  
  [],  
  [],  
  []),  
automaton_bool(FLAG,[0,1],AUTOMATON).
element_matrix_signature([],_40960,_40961,_40962,[]).
element_matrix_signature(  
  [[i-I,j-J,v-V]|Ms],  
  INDEX_I,  
  INDEX_J,  
  VALUE,  
  [S|Ss]) :-  
  INDEX_I#=I#/\INDEX_J#=J#/\VALUE#=V#<=>S,  
element_matrix_signature(Ms,INDEX_I,INDEX_J,VALUE,Ss).
B.131 element_product

◊ **Meta-Data:**

```prolog
ctr_date(element_product, ['20051229', '20060808']).
```

```prolog
ctr_origin(
    element_product,
    \cite{OttossonThorsteinsson00}, []).
```

```prolog
ctr_synonyms(element_product, [element]).
```

```prolog
ctr_arguments(
    element_product,
    ['Y'-dvar, 'TABLE'-collection(value-int), 'X'-dvar, 'Z'-dvar]).
```

```prolog
ctr_restrictions(
    element_product,
    ['Y'>=1,
    'Y'=<size('TABLE'),
    'X'>=0,
    'Z'>=0,
    required('TABLE', value),
    'TABLE'\value>=0]).
```

```prolog
ctr_example(
    element_product,
    element_product(
        3,
        [[value-6], [value-9], [value-2], [value-9]],
        5,
        10)).
```

```prolog
ctr_typical(
    element_product,
    ['X'>0,
    'Z'>0,
    size('TABLE')>1,
    range('TABLE'\value)>1,
    'TABLE'\value>0]).
```

```prolog
ctr_derived_collections(
    element_product,
    [col('ITEM'-collection(y-dvar, x-dvar, z-dvar),
    [item(y-'Y', x-'X', z-'Z')]]).
```
ctr_graph(  
element_product,
  ['ITEM','TABLE'],
  2,
  ['PRODUCT'>>collection(item,table)],
  [item^y=table^key,item^z=item^x*table^value],
  ['NARC'=1],
  []).

ctr_eval(element_product, [reformulation(element_product_r)]).

ctr_pure_functional_dependency(element_product, []).

ctr_functional_dependency(element_product, 4, [1, 2, 3]).

ctr_extensible(element_product, [], 'TABLE', suffix).

element_product_r(Y,TABLE,X,Z) :-
  check_type(dvar,Y),
  collection(TABLE,[int_gteq(0)]),
  check_type(dvar,X),
  check_type(dvar,Z),
  length(TABLE,N),
  Y#>=1,
  Y#=<N,
  X#>=0,
  Z#>=0,
  get_attr1(TABLE,VALUES),
  element(Y,VALUES,VAL),
  Z#=VAL*X.
B.132 element_sparse

◊ Meta-Data:

ctr_date(element_sparse, ['20030820', '20040530', '20060808']).

ctr_origin(element_sparse, '\\index{CHIP|indexuse}CHIP', []).

ctr_usual_name(element_sparse, element).

ctr_arguments(element_sparse,
    ['ITEM'-collection(index-dvar,value-dvar),
     'TABLE'-collection(index-int,value-int),
     'DEFAULT'-int]).

ctr_restrictions(element_sparse,
    [required('ITEM',[index,value]),
     'ITEM'\index{\textsuperscript{\text{-index}}>=1},
     size('ITEM')=1,
     size('TABLE')>0,
     required('TABLE',[index,value]),
     'TABLE'\index{\textsuperscript{\text{-index}}>=1},
     distinct('TABLE',index)]).

ctr_example(element_sparse,
    element_sparse(  
        [[index-2,value-5]],
        [[index-1,value-6],
         [index-2,value-5],
         [index-4,value-2],
         [index-8,value-9]],
        5)).

ctr_typical(element_sparse,
    [size('TABLE')>1,range('TABLE'\index{\textsuperscript{\text{-value}}})>1]).

ctr_exchangeable(element_sparse,
    [items('TABLE',all),
     vals(['ITEM'\index{\textsuperscript{\text{-value}}}, 'TABLE'\index{\textsuperscript{\text{-value}}}, 'DEFAULT'],
           int,]
=\=,
all,
dontcare))).

ctr_derived_collections(
element_sparse,
  [col('DEF'-collection(index-int, value-int),
     [item(index-0, value-'DEFAULT')]),
   col('TABLE_DEF'-collection(index-dvar, value-dvar),
     [item(index-'TABLE'\index, value-'TABLE'\value),
      item(index-'DEF'\index, value-'DEF'\value)])]
).

ctr_graph(
element_sparse,
  ['ITEM','TABLE_DEF'],
  2,
  ['PRODUCT'\collection(item,table_def)],
  [item\value=table_def\value,
   item\index=table_def\index#/table_def\index=0],
  ['NARC'\>=1],
  []).

ctr_eval(
element_sparse,
  [reformulation(element_sparse_r),
   automaton(element_sparse_a)])
).

element_sparse_r(ITEM,TABLE,DEFAULT) :-
  length(ITEM,1),
  length(TABLE,N),
  N>0,
  collection(ITEM,[dvar_gteq(1),dvar]),
  collection(TABLE,[int_gteq(1),dvar]),
  check_type(int,DEFAULT),
  get_attr1(ITEM,[I]),
  get_attr2(ITEM,[V]),
  get_attr1(TABLE,INDEXES),
  get_attr2(TABLE,VALUES),
  all_different(INDEXES),
  element_sparse1(INDEXES,VALUES,I,V,DEFAULT,Term1,Term2),
  call(Term1#/Term2).

element_sparse1([],[],_37831,V,DEFAULT,0,V#=DEFAULT).

element_sparse1( [IND|R],
[VAL|S],
I,
V,
DEFAULT,
I#=IND#/\V#=VAL#/\T,
I\#\=IND#/\U) :-

element_sparse_a(FLAG,ITEM,TABLE,DEFAULT) :-
length(ITEM,1),
length(TABLE,N),
N>0,
collection(ITEM,[dvar_gteq(1),dvar]),
collection(TABLE,[int_gteq(1),dvar]),
check_type(int,DEFAULT),
get_attr1(TABLE,INDEXES),
all_different(INDEXES),
ITEM=[[index-ITEM_INDEX,value-ITEM_VALUE]],
element_sparse_signature(
  TABLE,
  SIGNATURE,
  ITEM_INDEX,
  ITEM_VALUE,
  DEFAULT),
AUTOMATON=
automaton(
  SIGNATURE,
  _40987,
  SIGNATURE,
  [source(s),sink(d),sink(t)],
  [arc(s,0,s),
   arc(s,1,t),
   arc(s,2,d),
   arc(d,1,t),
   arc(d,2,d),
   arc(t,0,t),
   arc(t,1,t),
   arc(t,2,t)],
  [],
  [],
  []),
automaton_bool(FLAG,[0,1,2],AUTOMATON).
element_sparse_signature([],[],_37831,_37832,_37833).
element_sparse_signature(
[[index-TABLE_INDEX,value-TABLE_VALUE]|TABLEs],
[S|Ss],
ITEM_INDEX,
ITEM_VALUE,
DEFAULT) :-
  S in 0..2,
  ITEM_INDEX\TABLE_INDEX/ITEM_VALUE\DEFAULT\=S#=0,
  ITEM_INDEX\=TABLE_INDEX/ITEM_VALUE\=TABLE_VALUE\=S#=1,
  ITEM_INDEX\=TABLE_INDEX/ITEM_VALUE\=DEFAULT\=S#=2,
  element_sparse_signature(
    TABLEs,
    Ss,
    ITEM_INDEX,
    ITEM_VALUE,
    DEFAULT).
B.133 elementn

◊ Meta-Data:

ctr_date(elementn, ['20061004']).

ctr_origin(elementn, 'P. Flener', []).

ctr_arguments(elementn, ['INDEX'-dvar, 'TABLE'-collection(value-int), 'ENTRIES'-collection(entry-dvar)]).

ctr_restrictions(elementn, ['INDEX'>=1, 'INDEX'=<size('TABLE')-size('ENTRIES')+1, size('TABLE')>0, size('ENTRIES')>0, size('TABLE')>=size('ENTRIES'), required('TABLE',value), required('ENTRIES',entry)]).

ctr_example(elementn, elementn(3, [[value-6],[value-9],[value-2],[value-9]], [[entry-2],[entry-9]])).

ctr_typical(elementn, [size('TABLE')>1,range('TABLE'\^value)>1,size('ENTRIES')>1]).

ctr_exchangeable(elementn, [vals([\'TABLE'\^value,'ENTRIES'\^entry], int, =\=, all, dontcare)]).

ctr_eval(elementn,
[reformulation(elementn_r), automaton(elementn_a)].

ctr_extensible(elementn, [], 'TABLE', suffix).

elementn_r(INDEX, TABLE, ENTRIES) :-
    length(TABLE, N),
    length(ENTRIES, M),
    N > 0,
    M > 0,
    N >= M,
    NM is N - M + 1,
    check_type(dvar(1, NM), INDEX),
    collection(TABLE, [int]),
    collection(ENTRIES, [dvar]),
    get_attr1(TABLE, TAB),
    get_attr1(ENTRIES, VALS),
    elementn1(VALS, 0, INDEX, TAB).

elementn1([], 17950, 17951, 17952).

elementn1([V|R], K, INDEX, TAB) :-
    IND# = INDEX + K,
    element(IND, TAB, V),
    K1 is K + 1,
    elementn1(R, K1, INDEX, TAB).

elementn_a(FLAG, INDEX, TABLE, ENTRIES) :-
    length(TABLE, T),
    length(ENTRIES, E),
    T > 0,
    E > 0,
    T >= E,
    TE is T - E + 1,
    check_type(dvar(1, TE), INDEX),
    collection(TABLE, [int]),
    collection(ENTRIES, [dvar]),
    elementn_get_para(TABLE, Table),
    elementn_get_para(ENTRIES, Entries),
    elementn_gen_val(1, TE, LV),
    elementn_gen_arc(1, TE, E, LV, Table, Arcs),
    append([INDEX], Entries, SIGNATURE),
    AUTOMATON =
    automaton(
        SIGNATURE,
        _21210,
        SIGNATURE,
[source(s), sink(t)],
Arct,
[],
[],
[],
[]),
union_dom_list_int(SIGNATURE, ALPHABET),
automaton Bool(FLAG, ALPHABET, AUTOMATON).

elementn_get_para([], []).

elementn_get_para([[?_17959-P] | R], [P | S]) :-
    elementn_get_para(R, S).

elementn_gen_val(I, I, [I]) :-
    !.

elementn_gen_val(I, J, [I | R]) :-
    I < J,
    I1 is I + 1,
    elementn_gen_val(I1, J, R).

elementn_gen_arc(I, J, _, [], [], Arcs) :-
    I =< J,
    K = 1 + E * (I - 1),
    A0 = [arc(s, I, K)],
    elementn_gen_arc1(I, E, K, [F | T], A1),
    I1 is I + 1,
    elementn_gen_arc(I1, J, E, S, T, A),
    append(A0, A1, A2),
    append(A2, A, Arcs).

elementn_gen_arc1(J, E, K, [F | T], [arc(K, F, K1) | R]) :-
    J < E,
    !,
    K1 is K + 1,
    J1 is J + 1,
    elementn_gen_arc1(J1, E, K1, T, R).

elementn_gen_arc(E, E, K, [F | _17959], [arc(K, F, t)]).
B.134  elements

◊ **META-DATA:**

```prolog
ctr_date(elements,['20030820','20060808']).

ctr_origin(elements,'Derived from %c.',[element]).

ctr_arguments(elements,
['ITEMS'-collection(index-dvar,value-dvar),
'TABLE'-collection(index-int,value-dvar)]).

ctr_restrictions(elements,
  [required('ITEMS',[index,value]),
   'ITEMS'^(index)>1,
   'ITEMS'^(index)<size('TABLE'),
   required('TABLE',[index,value]),
   'TABLE'^(index)>1,
   'TABLE'^(index)<size('TABLE'),
   distinct('TABLE',index)]).

ctr_example(elements,
  elements([[index-4,value-9],[index-1,value-6]],
            [[index-1,value-6],
             [index-2,value-9],
             [index-3,value-2],
             [index-4,value-9]])).

ctr_typical(elements,
  [size('ITEMS')>1,
   range('ITEMS'^(index)>1,
   size('TABLE')>1,
   range('TABLE'^(value)>1)]).

ctr_exchangeable(elements,
  [items('ITEMS',all),
   items('TABLE',all),
   vals(['ITEMS'^(value,'TABLE'^(value),int,\=,all,dontcare)])).
```

ctr_graph
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

elements,
[‘ITEMS’,’TABLE’],
2,
[‘PRODUCT’>>collection(items,table)],
[items`index=table`index,items`value=table`value],
[‘NARC’=size(‘ITEMS’)],
[]).

ctr_eval(elements,[reformulation(elements_r)]).

ctr_pure_functional_dependency(elements,[]).

ctr_functional_dependency(elements,1-2,[1-1,2]).

elements_r(ITEMS,TABLE) :-
  length(TABLE,N),
  collection(ITEMS,[dvar(1,N),dvar]),
  collection(TABLE,[int(1,N),dvar]),
  get_attr1(TABLE,INDEXES),
  all_different(INDEXES),
  sort(TABLE,STAB),
  get_attr2(STAB,VALUES),
  get_attr1(ITEMS,INDS),
  get_attr2(ITEMS,VALS),
  elements1(INDS,VALS,VALUES).

elements1([],[],_32251).

elements1([IND|R],[VAL|S],VALUES) :-
  element (IND,VALUES,VAL),
  elements1(R,S,VALUES).
B.135 elements_alldifferent

◊ **META-DATA:**

```prolog
ctr_date(elements_alldifferent,['20030820','20060809']).
```

```prolog
ctr_origin(
    elements_alldifferent,
    Derived from %c and %c.,
    [elements_alldifferent]).
```

```prolog
ctr_synonyms(
    elements_alldifferent,
    [elements_alldiff,elements_alldistinct]).
```

```prolog
ctr_arguments(
    elements_alldifferent,
    ['ITEMS'-collection(index-dvar,value-dvar),
    'TABLE'-collection(index-int,value-dvar)]).
```

```prolog
ctr_restrictions(
    elements_alldifferent,
    [required('ITEMS',[index,value]),
    'ITEMS'\^index>=1,
    'ITEMS'\^index=<size('TABLE'),
    size('ITEMS')=size('TABLE'),
    required('TABLE',[index,value]),
    'TABLE'\^index>=1,
    'TABLE'\^index=<size('TABLE'),
    distinct('TABLE',index)]).
```

```prolog
ctr_example(
    elements_alldifferent,
    elements_alldifferent(
        [[index-2,value-9],
        [index-1,value-6],
        [index-4,value-9],
        [index-3,value-2]],
        [[index-1,value-6],
        [index-2,value-9],
        [index-3,value-2],
        [index-4,value-9]]).
```

```prolog
ctr_typical(
    elements_alldifferent,
    [size('ITEMS')>1,
```
range('ITEMS'\textasciitilde value)>1, 
size('TABLE')>1, 
range('TABLE'\textasciitilde value)>1). 

\texttt{ctr\_exchangeable(}
\texttt{ elements\_alldifferent,}
\texttt{ [args([['ITEMS','TABLE']])],}
\texttt{ items('ITEMS',all),}
\texttt{ items('TABLE',all),}
\texttt{ vals(['ITEMS'\textasciitilde value,'TABLE'\textasciitilde value],int,\textasciitilde,all,dontcare)].}

\texttt{ctr\_graph(}
\texttt{ elements\_alldifferent,}
\texttt{ ['ITEMS','TABLE'],}
\texttt{ 2,}
\texttt{ ['PRODUCT'\textasciitilde collection(items,table)],}
\texttt{ [items\textasciitilde index=table\textasciitilde index,items\textasciitilde value=table\textasciitilde value],}
\texttt{ ['NVERTEX'=size('ITEMS')+size('TABLE')],}
\texttt{ []).}

\texttt{ctr\_eval(}
\texttt{ elements\_alldifferent,}
\texttt{ [reformulation(elements\_alldifferent\_r)])].}

\texttt{ctr\_functional\_dependency(elements\_alldifferent,1-2,[1-1,2]).}

\texttt{elements\_alldifferent\_r(ITEMS,TABLE) :-}
\texttt{ length(TABLE,N),}
\texttt{ collection(ITEMS,[dvar(1,N),dvar]),}
\texttt{ collection(TABLE,[int(1,N),dvar]),}
\texttt{ get\_attr1(TABLE,INDEXES),}
\texttt{ all\_different(INDEXES),}
\texttt{ sort(TABLE,STAB),}
\texttt{ get\_attr2(STAB,VALUES),}
\texttt{ get\_attr1(ITEMS,INDS),}
\texttt{ get\_attr2(ITEMS,VALS),}
\texttt{ all\_different(INDS),}
\texttt{ elements\_alldifferent\_1(INDS,VALS,VALUES).}

\texttt{elements\_alldifferent\_1([],[],\textasciitilde38083).}

\texttt{elements\_alldifferent\_1([IND|R],[VAL|S],VALUES) :-}
\texttt{ element(IND,VALUES,VAL),}
\texttt{ elements\_alldifferent\_1(R,S,VALUES).}
B.136 elements_sparse

◊ META-DATA:

ctr_date(elements_sparse, [‘20030820’, ‘20060809’]).

ctr_origin(elements_sparse, ‘Derived from %c.’, [element_sparse]).

ctr_arguments(
    elements_sparse,
    [‘ITEMS’-collection(index-dvar, value-dvar),
     ‘TABLE’-collection(index-int, value-int),
     ‘DEFAULT’-int]).

ctr_restrictions(
    elements_sparse,
    [required(‘ITEMS’, [index, value]),
     ‘ITEMS’^index>=1,
     required(‘TABLE’, [index, value]),
     ‘TABLE’^index>=1,
     distinct(‘TABLE’, index)]).

ctr_example(
    elements_sparse,
    elements_sparse(
        [[index-8, value-9],
         [index-3, value-5],
         [index-2, value-5]],
        [[index-1, value-6],
         [index-2, value-5],
         [index-4, value-2],
         [index-8, value-9]],
        5)).

ctr_typical(
    elements_sparse,
    [size(‘ITEMS’)>1,
     range(‘ITEMS’^value)>1,
     size(‘TABLE’)>1,
     range(‘TABLE’^value)>1]).

ctr_exchangeable(
    elements_sparse,
    [items(‘ITEMS’, all),
     items(‘TABLE’, all),
     vals(}
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

['ITEMS' value, 'TABLE' value, 'DEFAULT'],
  int,
  =\|=,
  all,
  dontcare)).

ctr_derived_collections(
collection
  elements_sparse,
  [col('DEF'-collection(index-int, value-int),
    item(index-0, value-DEFAULT)),
   col('TABLE_DEF'-collection(index-dvar, value-dvar),
    item(index-'TABLE' index, value-'TABLE' index),
    item(index-'DEF' index, value-'DEF' value))]).

ctr_graph(
collection
  elements_sparse,
  ['ITEMS', 'TABLE_DEF'],
  2,
  ['PRODUCT' >> collection(items, table_def)],
  [items\ value=table_def\ value,
    items\ index=table_def\ index#/table_def\ index=0],
  ['NSOURCE' = size('ITEMS')],
  []).

ctr_eval(elements_sparse, [reformulation(elements_sparse_r)]).

elements_sparse_r(ITEMS, TABLE, DEFAULT) :-
collection(ITEMS, [dvar_gteq(1), dvar]),
collection(TABLE, [int_gteq(1), dvar]),
check_type(int, DEFAULT),
get_attr1(ITEMS, IS),
get_attr2(ITEMS, VS),
get_attr1(TABLE, INDEXES),
get_attr2(TABLE, VALUES),
all_different(INDEXES),
elements_sparse1(IS, VS, INDEXES, VALUES, DEFAULT).

elements_sparse1([], [], _38157, _38158, _38159).

elements_sparse1([I|R], [V|S], INDEXES, VALUES, DEFAULT) :-
elements_sparse2(INDEXES,
  VALUES,
  I,
  V,
  DEFAULT,
\begin{verbatim}
Term1,
Term2),
call(Term1\//Term2),
elements_sparse1(R,S,INDEXES,VALUES,DEFAULT).
elements_sparse2([],[],38157,V,DEFAULT,0,V#=DEFAULT).
elements_sparse2([IND|R], [VAL|S], I, V, DEFAULT, I#=IND#/V#=VAL#/T, I\=IND#/U) :-
\end{verbatim}
B.137  eq

◇ Meta-Data:

ctr_predefined(eq).

ctr_date(eq,['20070821']).

ctr_origin(eq,'Arithmetic.',[]).

ctr_synonyms(eq,[xeqy]).

ctr_arguments(eq,['VAR1'-dvar,'VAR2'-dvar]).

ctr_restrictions(eq,[]).

ctr_example(eq,eq(8,8)).

ctr_exchangeable(eq,
    [args([[VAR1','VAR2']],
        vals(['VAR1','VAR2'],int,=\=,all,dontcare)]).

ctr_eval(eq,[builtin(eq_b)]).

ctr_pure_functional_dependency(eq,[]).

ctr_functional_dependency(eq,2,[1]).

ctr_functional_dependency(eq,1,[2]).

eq_b(VAR1,VAR2) :-
    check_type(dvar,VAR1),
    check_type(dvar,VAR2),
    VAR1#=VAR2.
B.138  eq_cst

◊ **Meta-Data:**

ctr_predefined(eq_cst).

ctr_date(eq_cst,['20090923']).

ctr_origin(eq_cst,'Arithmetic.',[]).

ctr_arguments(eq_cst,['VAR1'-dvar,'VAR2'-dvar,'CST2'-int]).

ctr_example(eq_cst,eq_cst(8,2,6)).

ctr_typical(eq_cst,['CST2'=\=0]).

ctr_exchangeable(eq_cst,

    [args([['VAR1'], ['VAR2', 'CST2']]),
     translate([['VAR1', 'VAR2']],
     translate([['VAR1', 'CST2']])).

    ctr_eval(eq_cst,[builtin(eq_cst_b)]).

ctr_pure_functional_dependency(eq_cst,[]).

ctr_functional_dependency(eq_cst,1,[2,3]).

ctr_functional_dependency(eq_cst,2,[1,3]).

ctr_functional_dependency(eq_cst,3,[1,2]).

    eq_cst_b(VAR1,VAR2,CST2) :-
        check_type(dvar,VAR1),
        check_type(dvar,VAR2),
        check_type(int,CST2),
        VAR1#\=VAR2+CST2.


B.139 eq_set

◊ **META-DATA:**

ctr_predefined(eq_set).

ctr_date(eq_set,[‘20030820’]).

ctr_origin(
    eq_set,
    Used for defining %c.,
    [alldifferent_between_sets]).

ctr_arguments(eq_set,[‘SET1’-svar,’SET2’-svar]).

ctr_example(eq_set,eq_set({3,5},{3,5})).

ctr_exchangeable(
    eq_set,
    [args([‘SET1’,’SET2’]),
     vals([‘SET1’,’SET2’],int,\=,all,dontcare)]).
B.140  equal_sboxes

◊ META-DATA:

ctr_date(equal_sboxes,['20070622','20090725']).

ctr_origin(
    equal_sboxes,
    Geometry, derived from \cite{RandellCuiCohn92},
    []).

ctr_synonyms(equal_sboxes,[equal]).

ctr_types(
    equal_sboxes,
    ['VARIABLES'-collection(v-dvar),
     'INTEGERS'-collection(v-int),
     'POSITIVES'-collection(v-int)]).

ctr_arguments(
    equal_sboxes,
    ['K'-int,
     'DIMS'-sint,
     'OBJECTS'-collection(oid-int,sid-int,x-'VARIABLES'),
     'SBOXES'-collection(sid-int,t-'INTEGERS',l-'POSITIVES'))).

ctr_restrictions(
    equal_sboxes,
    [size('VARIABLES')>=1,
     size('INTEGERS')>=1,
     size('POSITIVES')>=1,
     required('VARIABLES',v),
     size('VARIABLES')='K',
     required('INTEGERS',v),
     size('INTEGERS')='K',
     required('POSITIVES',v),
     size('POSITIVES')='K',
     'POSITIVES'\^v>0,
     'K'>0,
     'DIMS'>=0,
     'DIMS'<'K',
     increasing_seq('OBJECTS',[oid]),
     required('OBJECTS',[oid,sid,x]),
     'OBJECTS'\^oid>=1,
     'OBJECTS'\^oid=<size('OBJECTS'),
     'OBJECTS'\^sid>=1,}
'OBJECTS'\text{\textasciitilde}sid=<size('SBOXES'),
size('SBOXES')\geq1,
required('SBOXES',[sid,t,l]),
'SBOXES'\text{\textasciitilde}sid\geq1,
'SBOXES'\text{\textasciitilde}sid\leq<size('SBOXES'),
do\_not\_overlap('SBOXES')).

ctr\_example(
  equal\_sboxes,
  equal\_sboxes(2,
    [0,1],
    [[oid-1,sid-2,x-[[v-4],[v-1]]],
     [oid-2,sid-2,x-[[v-4],[v-1]]],
     [oid-3,sid-2,x-[[v-4],[v-1]]],
     [sid-1,t-[[v-0],[v-0]],l-[[v-1],[v-2]]],
     [sid-2,t-[[v-0],[v-0]],l-[[v-1],[v-1]]],
     [sid-2,t-[[v-1],[v-0]],l-[[v-1],[v-3]]],
     [sid-2,t-[[v-0],[v-2]],l-[[v-1],[v-1]]],
     [sid-3,t-[[v-0],[v-0]],l-[[v-3],[v-1]]],
     [sid-3,t-[[v-0],[v-1]],l-[[v-1],[v-1]]],
     [sid-3,t-[[v-2],[v-1]],l-[[v-1],[v-1]]],
     [sid-4,t-[[v-0],[v-0]],l-[[v-1],[v-1]]])).

ctr\_typical(equal\_sboxes,[size('OBJECTS')\geq1]).

ctr\_exchangeable(
  equal\_sboxes,
  [items('OBJECTS',all),
   items('SBOXES',all),
   items\_sync('OBJECTS'\text{\textasciitilde}x,'SBOXES'\text{\textasciitilde}t,'SBOXES'\text{\textasciitilde}l,all))).

ctr\_eval(equal\_sboxes,[logic(equal\_sboxes\_g)]).

ctr\_logic(
  equal\_sboxes,
  [DIMENSIONS,OIDs],
  [(origin(O1,S1,D)\rightarrow O1\text{\textasciitilde}x(D)+S1\_t(D)),
   (end(O1,S1,D)\rightarrow O1\text{\textasciitilde}x(D)+S1\_t(D)+S1\_l(D)),
   (equal\_sboxes(Dims,O1,S1,O2,S2))\rightarrow
    \forall D,
    \forall Dims,
    \text{origin}(O1,S1,D)=\text{origin}(O2,S2,D)\\downarrow
    \text{end}(O1,S1,D)=\text{end}(O2,S2,D)),
  (equal\_objects(Dims,O1,O2))\rightarrow
forall(
    S1,
    sboxes([01^sid]),
    exists(
        S2,
        sboxes([02^sid]),
        equal_sboxes(Dims,01,S1,02,S2))))},
(all_equal(Dims,OIDS)--->
forall(
    01,
    objects(OIDS),
    forall(
        02,
        objects(OIDS),
        01^oid#=02^oid-1#=>equal_objects(Dims,01,02))),
all_equal(DIMENSIONS,OIDS))).

ctr_contractible(equal_sboxes,[],'OBJECTS',suffix).

equal_sboxes_g(K,_30671,[],_30673) :-
    !,
    check_type(int_gteq(1),K).

equal_sboxes_g(K,_DIMS,OBJECTS,SBOXES) :-
    length(OBJECTS,O),
    length(SBOXES,S),
    O>0,
    S>0,
    check_type(int_gteq(1),K),
    collection(OBJECTS,[int(1,O),dvar(1,S),col(K,[dvar])]),
    collection(SBOXES,
        [int(1,S),col(K,[int]),col(K,[int_gteq(1)]))]),
    get_attr1(OBJECTS,OIDS),
    get_attr2(OBJECTS,SIDS),
    get_col_attr3(OBJECTS,1,XS),
    get_attr1(SBOXES,SIDES),
    get_col_attr2(SBOXES,1,TS),
    get_col_attr3(SBOXES,1,TL),
    collection_increasing_seq(OBJECTS,[1]),
    geost1(OIDS,SIDS,XS,Objects),
    geost2(SIDES,TS,TL,Sboxes),
    geost_dims(1,K,DIMENSIONS),
    ctr_logic(equal_sboxes,[DIMENSIONS,OIDS],Rules),
    geost(Objects,Sboxes,[overlap(true)],Rules).
B.141  equivalent

◊ Meta-Data:

ctr_date(equivalent, [‘20051226’]).

ctr_origin(equivalent, ‘Logic’, []).

ctr_synonyms(equivalent, [eq]).

ctr_arguments(equivalent, [VAR \rightarrow dvar, VARIABLES \rightarrow collection(var-dvar)]).

ctr_restrictions(equivalent, [VAR \geq 0, VAR \leq 1, size(VARIABLES) = 2, required(VARIABLES, var), VARIABLES^var \geq 0, VARIABLES^var \leq 1]).

ctr_example(equivalent, [equivalent(1, [[var-0], [var-0]]), equivalent(0, [[var-0], [var-1]]), equivalent(0, [[var-1], [var-0]]), equivalent(1, [[var-1], [var-1]])].

ctr_exchangeable(equivalent, [items(VARIABLES, all), vals([‘VAR’, ‘VARIABLES’^var], int(0 in 1), <, all, dontcare)]).

ctr_eval(equivalent, [reformulation(equivalent_r), automaton(equivalent_a)]).

ctr_pure_functional_dependency(equivalent, []).

ctr_functional_dependency(equivalent, 1, [2]).

equivalent_r(VAR, VARIABLES) :-
    check_type(dvar(0, 1), VAR),
    collection(VARIABLES, [dvar(0, 1)]),
length(VARIABLES, 2),
get_attr1(VARIABLES, [VAR1, VAR2]),
VAR#<=>(VAR1#<=>VAR2).

equivalent_a(FLAG, VAR, VARIABLES) :-
check_type(dvar(0,1),VAR),
collection(VARIABLES,[dvar(0,1)]),
length(VARIABLES,2),
get_attr1(VARIABLES, LIST),
append([VAR],LIST,LIST_VARIABLES),
AUTOMATON=
automaton(
    LIST_VARIABLES,
    _20298,
    LIST_VARIABLES,
    [source(s), sink(t)],
    [arc(s,0,i),
     arc(s,1,j),
     arc(i,0,l),
     arc(i,1,k),
     arc(j,0,k),
     arc(j,1,l),
     arc(k,0,t),
     arc(l,1,t)],
    [],
    [],
    []),
automaton_bool(FLAG,[0,1],AUTOMATON).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.142 exactly

◊ **META-DATA:**

ctr_date(exactly, ['20040807', '20060809']).

ctr_origin(exactly, 'Derived from %c and %c.', [atleast, atmost]).

ctr_synonyms(exactly, [count]).

ctr_arguments(exactly, ['N'-int, 'VARIABLES'-collection(var-dvar), 'VALUE'-int]).

ctr_restrictions(exactly, ['N'>=0, 'N'=<size('VARIABLES'), required('VARIABLES', var)]).

ctr_example(exactly, exactly(2, [[var-4], [var-2], [var-4], [var-5]], 4)).

ctr_typical(exactly, ['N'>0, 'N'<size('VARIABLES'), size('VARIABLES')>1]).

ctr_exchangeable(exactly, items('VARIABLES', all),
vals([ ['VARIABLES'\^var],
   int(=\=(\"VALUE\")),
   =\=,
   dontcare,
   dontcare])).

ctr_graph(exactly, ['VARIABLES'], 1,
['SELF'=>collection(variables)],
[variables\^var='VALUE'],
['NARC'='N'],
[]).

ctr_eval
exactly,  
[reformulation(exactly_r), automaton(exactly_a)].

ctr_pure_functional_dependency(exactly, []).  
ctr_functional_dependency(exactly, 1, [2, 3]).

ctr_aggregate(exactly, [], [+], union, id).  

exactly_r(N, VARIABLES, VALUE) :-
collection(VARIABLES, [dvar]),
length(VARIABLES, NVAR),
check_type(int(0, NVAR), N),
integer(VALUE),
get_attr1(VARIABLES, VARS),
get_minimum(VARS, MINVARS),
MIN is min(MINVARS, VALUE),
MAX is max(MAXVARS, VALUE),
complete_card(MIN, MAX, NVAR, [VALUE], [N], VN),
global_cardinality(VARS, VN).

exactly_a(FLAG, N, VARIABLES, VALUE) :-
collection(VARIABLES, [dvar]),
length(VARIABLES, NVAR),
check_type(int(0, NVAR), N),
integer(VALUE),
exactly_signature(VARIABLES, SIGNATURE, VALUE),
automaton( 
  SIGNATURE,
  _33126,
  SIGNATURE,
  [source(s), sink(s)],
  [arc(s, 0, s), arc(s, 1, s, [C+1])],
  [C],
  [0],
  [COUNT]),
COUNT#=N#<=>FLAG.

exactly_signature([], [], _31677).

exactly_signature([ [var-VAR] | VARS ], [S | Ss], VALUE) :-
  VAR#=VALUE#<=>S,
exactly_signature(VARS, Ss, VALUE).
B.143  gcd

◊ **Meta-Data:**

```prolog
ctr_predefined(gcd).
ctr_date(gcd, ['20070930']).
ctr_origin(gcd, '\cite{DenmatGotliebDucasse07}', []).
ctr_arguments(gcd, ['X'-dvar, 'Y'-dvar, 'Z'-dvar]).
ctr_restrictions(gcd, ['X'>0, 'Y'>0, 'Z'>0]).
ctr_example(gcd, gcd(24, 60, 12)).
ctr_typical(gcd, ['X'>1, 'Y'>1]).
ctr_exchangeable(gcd, [args([[/X/, [Y]], [Z]])]).
ctr_eval(gcd, [checker(gcd_c)]).
ctr_pure_functional_dependency(gcd, []).
ctr_functional_dependency(gcd, 1, [2, 3]).
gcd_c(X, Y, Z) :-
    check_type(int_gteq(1), X),
    check_type(int_gteq(1), Y),
    check_type(int_gteq(1), Z),
    Z is gcd(X, Y).
```
B.144 geost

◊ **META-DATA:**

\[\text{ctr\_predefined(geost).}\]

\[\text{ctr\_date(geost, ['20060919', '20080609', '20090116', '20090725']).}\]

\[\text{ctr\_origin(geost,'Generalisation of %c.',[diffn]).}\]

\[\text{ctr\_types(}\]
\[\quad \text{geost,}\]
\[\quad \quad \text{['VARIABLES'-collection(v-dvar),}\]
\[\quad \quad \quad \text{'INTEGERS'-collection(v-int),}\]
\[\quad \quad \quad \text{'POSITIVES'-collection(v-int)].}\]
\[\text{ctr\_arguments(}\]
\[\quad \text{geost,}\]
\[\quad \quad \text{['K'-int,}\]
\[\quad \quad \quad \text{'OBJECTS'-collection(oid-int,sid-dvar,x-'VARIABLES'),}\]
\[\quad \quad \quad \text{'SBOXES'-collection(sid-int,t-'INTEGERS',l-'POSITIVES')].}\]
\[\text{ctr\_restrictions(}\]
\[\quad \text{geost,}\]
\[\quad \quad \text{[size('VARIABLES')>=1,}\]
\[\quad \quad \quad \text{size('INTEGERS')>=1,}\]
\[\quad \quad \quad \text{size('POSITIVES')>=1,}\]
\[\quad \quad \quad \text{required('VARIABLES',v),}\]
\[\quad \quad \quad \text{size('VARIABLES')='K',}\]
\[\quad \quad \quad \text{required('INTEGERS',v),}\]
\[\quad \quad \quad \text{size('INTEGERS')='K',}\]
\[\quad \quad \quad \text{required('POSITIVES',v),}\]
\[\quad \quad \quad \text{size('POSITIVES')='K',}\]
\[\quad \quad \quad \text{'POSITIVES'\^{}v>0,}\]
\[\quad \quad \quad \text{'K'>0,}\]
\[\quad \quad \quad \text{required('OBJECTS',[oid,sid,x])},\]
\[\quad \quad \quad \text{distinct('OBJECTS',oid),}\]
\[\quad \quad \quad \text{'OBJECTS'\^{}oid=1,}\]
\[\quad \quad \quad \text{'OBJECTS'\^{}oid=<size('OBJECTS'),}\]
\[\quad \quad \quad \text{'OBJECTS'\^{}sid=1,}\]
\[\quad \quad \quad \text{'OBJECTS'\^{}sid=<size('SBOXES'),}\]
\[\quad \quad \quad \text{size('SBOXES')>=1,}\]
\[\quad \quad \quad \text{required('SBOXES',[sid,t,l])},\]
\[\quad \quad \quad \text{'SBOXES'\^{}sid=1,}\]
\[\quad \quad \quad \text{'SBOXES'\^{}sid=<size('SBOXES'),}\]
\[\quad \quad \quad \text{do\_not\_overlap('SBOXES').}\]
CTR example:

\[
\text{geost,}
\text{geost (}
\text{2,}
\text{[oid-1, sid-1, x-[[v-1], [v-2]]],}
\text{[oid-2, sid-5, x-[[v-2], [v-1]]],}
\text{[oid-3, sid-8, x-[[v-4], [v-1]]]],}
\text{[sid-1, t-[[v-0], [v-0]], l-[[v-2], [v-1]]],}
\text{[sid-1, t-[[v-0], [v-1]], l-[[v-1], [v-2]]],}
\text{[sid-1, t-[[v-1], [v-2]], l-[[v-3], [v-1]]],}
\text{[sid-2, t-[[v-0], [v-0]], l-[[v-3], [v-1]]],}
\text{[sid-2, t-[[v-0], [v-1]], l-[[v-1], [v-3]]],}
\text{[sid-2, t-[[v-2], [v-1]], l-[[v-1], [v-1]]],}
\text{[sid-3, t-[[v-0], [v-0]], l-[[v-2], [v-1]]],}
\text{[sid-3, t-[[v-1], [v-1]], l-[[v-1], [v-2]]],}
\text{[sid-3, t-[[v-2], [v-2]], l-[[v-3], [v-1]]],}
\text{[sid-4, t-[[v-0], [v-0]], l-[[v-3], [v-1]]],}
\text{[sid-4, t-[[v-0], [v-1]], l-[[v-1], [v-1]]],}
\text{[sid-4, t-[[v-2], [v-1]], l-[[v-1], [v-3]]],}
\text{[sid-5, t-[[v-0], [v-0]], l-[[v-2], [v-1]]],}
\text{[sid-5, t-[[v-1], [v-1]], l-[[v-1], [v-1]]],}
\text{[sid-5, t-[[v-0], [v-2]], l-[[v-2], [v-1]]],}
\text{[sid-6, t-[[v-0], [v-0]], l-[[v-3], [v-1]]],}
\text{[sid-6, t-[[v-0], [v-1]], l-[[v-1], [v-1]]],}
\text{[sid-6, t-[[v-2], [v-1]], l-[[v-1], [v-1]]],}
\text{[sid-7, t-[[v-0], [v-0]], l-[[v-3], [v-2]]],}
\text{[sid-8, t-[[v-0], [v-0]], l-[[v-2], [v-3]]]])}.
\]

CTR typical(geost, [size('OBJECTS') > 1]).

CTR exchangeable(geost, [items('OBJECTS', all),
items('SBOXES', all),
items_sync('OBJECTS' \ x, 'SBOXES' \ t, 'SBOXES' \ l, all),
vals(['SBOXES' \ l \ v], int(>=(1)), >, dontcare, dontcare)]).

CTR eval(geost, [builtin(geost_b)]).

geost_b(K, [], _44598) :-
!,
check_type(int_gteq(1), K).

geost_b(K, OBJECTS, SBOXES) :-
length(OBJECTS, 0),
length(SBOXES,S),
O>0,
S>0,
check_type(int_gteq(1),K),
collection(OBJECTS,[int(1,O),dvar(1,S),col(K,[dvar])]),
collection(  
    SBOXES,  
    [int(1,S),col(K,[int]),col(K,[int_gteq(1)])]),
get_attr1(OBJECTS,OIDS),
get_attr2(OBJECTS,SIDS),
get_col_attr3(OBJECTS,1,XS),
get_attr1(SBOXES,SIDES),
get_col_attr2(SBOXES,1,TS),
get_col_attr3(SBOXES,1,TL),
geost1(OIDS,SIDS,XS,Objects),
geost2(SIDES,TS,TL,Sboxes),
catch(geost(Objects,Sboxes),_Flag,fail).
B.145 geost_time

◊ Meta-Data:

ctr_predefined(geost_time).

ctr_date(geost_time, [\'20060919\']).

ctr_origin(geost_time, 'Generalisation of %c.', [diffn]).

ctr_types(
    geost_time,
    ['VARIABLES'-collection(v-dvar),
     'INTEGERS'-collection(v-int),
     'POSITIVES'-collection(v-int)]).

ctr_arguments(
    geost_time,
    ['K'-int,
     'DIMS'-sint,
     OBJECTS-
      collection(
        oid-int,
        sid-dvar,
        x-'VARIABLES',
        start-dvar,
        duration-dvar,
        end-dvar),
     'SBOXES'-collection(sid-int, t-'INTEGERS', l-'POSITIVES')]).

ctr_restrictions(
    geost_time,
    [size('VARIABLES')>=1,
     size('INTEGERS')>=1,
     size('POSITIVES')>=1,
     required('VARIABLES', v),
     size('VARIABLES')='K',
     required('INTEGERS', v),
     size('INTEGERS')='K',
     required('POSITIVES', v),
     size('POSITIVES')='K',
     'POSITIVES'~v>0,
     'K'>=0,
     'DIMS'>=0,
     'DIMS'<'K',
     distinct('OBJECTS', oid),
)
required('OBJECTS', [oid, sid, x]),
require_at_least(2, 'OBJECTS', [start, duration, end]),
'OBJECTS'\^oid>=1,
'OBJECTS'\^oid=<size('OBJECTS'),
'OBJECTS'\^sid>=1,
'OBJECTS'\^sid=<size('SBOXES'),
'OBJECTS'\^duration>=0,
size('SBOXES')>=1,
required('SBOXES', [sid, t, l]),
'SBOXES'\^sid>=1,
'SBOXES'\^sid=<size('SBOXES'),
do_not_overlap('SBOXES')).

ctr_example(
  geost_time,
  geost_time(2,
    {0,1},
    [[oid-1, sid-1, x-[[v-1], [v-2]],
      start-0, duration-1, end-1],
    [oid-2, sid-5, x-[[v-2], [v-1]],
      start-0, duration-1, end-1],
    [oid-3, sid-8, x-[[v-4], [v-1]],
      start-0, duration-1, end-1]],
  [[sid-1,t-[[v-0],[v-0]],l-[[v-2],[v-1]]],
   [sid-1,t-[[v-0],[v-1]],l-[[v-1],[v-2]]],
   [sid-1,t-[[v-1],[v-2]],l-[[v-3],[v-1]]],
   [sid-2,t-[[v-0],[v-0]],l-[[v-3],[v-1]]],
   [sid-2,t-[[v-0],[v-1]],l-[[v-1],[v-3]]],
   [sid-2,t-[[v-2],[v-1]],l-[[v-1],[v-1]]],
   [sid-3,t-[[v-0],[v-0]],l-[[v-2],[v-1]]],
   [sid-3,t-[[v-1],[v-1]],l-[[v-1],[v-2]]],
   [sid-3,t-[[v-2],[v-2]],l-[[v-3],[v-1]]],
   [sid-4,t-[[v-0],[v-0]],l-[[v-3],[v-1]]],
   [...]}
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

\[
\text{ctr\_typical}(\text{geost\_time}, [\text{size('OBJECTS')}>1]).
\]

\[
\text{ctr\_exchangeable}(\text{geost\_time},
\begin{align*}
\text{items('OBJECTS', all),} \\
\text{items('SBOXES', all),} \\
\text{items\_sync('OBJECTS'\^x,'SBOXES'\^t,'SBOXES'\^l, all),} \\
\text{vals(['SBOXES'\^l\^v], int(>=(1)), >, dont\_care, dont\_care),} \\
\text{translate(['OBJECTS'\^start,'OBJECTS'\^end])}).
\end{align*}
\]
B.146  \texttt{geq}

\begin{itemize}
\item \textbf{Meta-Data:}
\end{itemize}

\begin{verbatim}
ctr_predefined(geq).
ctr_date(geq, [’20070821’]).
ctr_origin(geq, ’Arithmetic.’, []).
ctr_synonyms(geq, [rel, xgteqy]).
ctr_arguments(geq, [’VAR1’-dvar, ’VAR2’-dvar]).
ctr_example(geq, geq(8,1)).
ctr_typical(geq, [’VAR1’>’VAR2’]).
ctr_exchangeable(
geq,
    [vals([’VAR1’], int(>=’VAR2’), =\=, all, dontcare),
     vals([’VAR2’], int(<=’VAR1’), =\=, all, dontcare)]).
ctr_eval(geq, [builtin(geq_b)]).

geq_b(VAR1, VAR2) :-
    check_type(dvar, VAR1),
    check_type(dvar, VAR2),
    VAR1#>=VAR2.
\end{verbatim}
B.147  \texttt{geq\_cst}

\textbf{Meta-Data:}

\begin{verbatim}
ctr_predefined(geq_cst).
ctr_date(geq_cst, [’20090912’]).
ctr_origin(geq_cst, ’Arithmetic.’, []).
ctr_arguments(geq_cst, [’VAR1’-dvar, ’VAR2’-dvar, ’CST2’-int]).
ctr_example(geq_cst, geq_cst(8, 1, 7)).
ctr_typical(geq_cst, [’CST2’ =\= 0, ’VAR1’ >’VAR2’ +’CST2’]).
ctr_exchangeable(geq_cst, [args([’VAR1’],[’VAR2’,’CST2’]),
                       vals([’VAR1’],int(>=’VAR2’+’CST2’),=\=,all,dontcare),
                       vals([’VAR2’],int(=\=’VAR1’-’CST2’),=\=,all,dontcare),
                       vals([’CST2’],int(=\=’VAR1’-’VAR2’),=\=,all,dontcare)]).
ctr_eval(geq_cst, [builtin(geq_cst_b)]).
\end{verbatim}

\texttt{geq\_cst\_b(VAR1,VAR2,CST2) :-
  check_type(dvar,VAR1),
  check_type(dvar,VAR2),
  check_type(int,CST2),
  VAR1#>=VAR2+CST2.}
B.148 global_cardinality

◊ Meta-Data:

ctr_date(
  global_cardinality,
  ['20030820','20040530','20060809','20091218']).

ctr_origin(
  global_cardinality,
  \index{CHARME|indexuse}CHARME \cite{OplobeduMarcovitchTourbier89}, [])).

ctr_synonyms(
  global_cardinality,
  [count, 
distribute, 
distribution, 
gcc, 
card_var_gcc, 
egcc, 
extended_global_cardinality]).

ctr_arguments(
  global_cardinality,
  ['VARIABLES'-collection(var-dvar), 
   'VALUES'-collection(val-int,noccurrence-dvar)]).

ctr_restrictions(
  global_cardinality,
  [required('VARIABLES',var), 
   required('VALUES',[val,noccurrence]), 
   distinct('VALUES',val), 
   'VALUES'ˆnoccurrence>=0, 
   'VALUES'ˆnoccurrence=<size('VARIABLES'))].

ctr_example(
  global_cardinality,
  global_cardinality(
    [[var-3],[var-3],[var-8],[var-6]], 
    [[val-3,noccurrence-2], 
      [val-5,noccurrence-0], 
      [val-6,noccurrence-1]])).

ctr_typical(
  global_cardinality,
[size('VARIABLES') > 1,
range('VARIABLES' \^ var) > 1,
size('VALUES') > 1,
size('VARIABLES') >= size('VALUES'),
in_attr('VARIABLES', var,'VALUES', val)].

ctr_exchangeable(
  global_cardinality,
  [items('VARIABLES', all),
   items('VALUES', all),
   vals(
     ['VARIABLES' \^ var],
     all(notin('VALUES' \^ val)),
     =,
     dontcare,
     dontcare),
   vals(
     ['VARIABLES' \^ var,'VALUES' \^ val],
     int,
     =\=,
     all,
     dontcare])].

ctr_graph(
  global_cardinality,
  ['VARIABLES'],
  1,
  foreach('VALUES', ['SELF'>>collection(variables)]),
  [variables \^ var='VALUES' \^ val],
  ['NVERTEX=' 'VALUES' \^ noccurrence],
  []).

ctr_eval(global_cardinality, [builtin(global_cardinality_b)]).

ctr_pure_functional_dependency(global_cardinality, []).

ctr_functional_dependency(global_cardinality, 2-2, [1, 2-1]).

ctr_contractible(global_cardinality, [], 'VALUES', any).

global_cardinality_b(VARIABLES, VALUES) :-
  length(VARIABLES, N),
  collection(VARIABLES, [dvar]),
  collection(VALUES, [int, dvar(0, N)]),
  get_attr1(VARIABLES, VARS),
  get_attr1(VALUES, VALS),
get_attr2(VALUES, NOCCS),
all_different(VALS),
get_minimum(VARS, MINVARS),
get_maximum(VARS, MAXVARS),
get_minimum(VALS, MINVALS),
get_maximum(VALS, MAXVALS),
MIN is min(MINVARS, MINVALS),
MAX is max(MAXVARS, MAXVALS),
complete_card(MIN, MAX, N, VALS, NOCCS, VN),
global_cardinality(VARS, VN).
B.149  global_cardinality_low_up

◊ Meta-Data:

ctr_date(
  global_cardinality_low_up,
  ['20031008','20040530','20060809','20090521']).

ctr_origin(
  global_cardinality_low_up,
  Used for defining %c.,
  [sliding_distribution]).

ctr_synonyms(global_cardinality_low_up,[gcc_low_up,gcc]).

ctr_arguments(
  global_cardinality_low_up,
  ['VARIABLES'-collection(var-dvar),
   'VALUES'-collection(val-int,omin-int,omax-int)]).

ctr_restrictions(
  global_cardinality_low_up,
  [required('VARIABLES',var),
   size('VALUES')>0,
   required('VALUES',[val,omin,omax]),
   distinct('VALUES',val),
   'VALUES'~omin>=0,
   'VALUES'~omax=<size('VARIABLES'),
   'VALUES'~omin=<'VALUES'~omax]).

ctr_example(
  global_cardinality_low_up,
  global_cardinality_low_up(
    [[var-3],[var-3],[var-8],[var-6]],
    [[val-3,omin-2,omax-3],
     [val-5,omin-0,omax-1],
     [val-6,omin-1,omax-2]]).

ctr_typical(
  global_cardinality_low_up,
  [size('VARIABLES')>1,
   range('VARIABLES'~var)>1,
   size('VALUES')>1,
   'VALUES'~omin=<size('VARIABLES'),
   'VALUES'~omax>0,
   'VALUES'~omax<size('VARIABLES'),
   ...]}
size('VARIABLES') > size('VALUES'),
in_attr('VARIABLES', var, 'VALUES', val)).

ctr_exchangeable(
global_cardinality_low_up,
[items('VARIABLES', all),
vals(
  ['VARIABLES'ˆvar],
  all(notin('VALUES'ˆval)),
  =,
  dontcare,
  dontcare),
items('VALUES', all),
vals([`VALUES'ˆomin], int(>=0), >, dontcare, dontcare),
vals(
  ['VALUES'ˆomax],
  int(=<(size('VARIABLES'))),
  <,
  dontcare,
  dontcare),
vals(
  ['VARIABLES'ˆvar,'VALUES'ˆval],
  int,
  =\=,
  all,
  dontcare)).

ctr_graph(
  global_cardinality_low_up,
  ['VARIABLES'],
  1,
  foreach('VALUES', ['SELF'>>collection(variables)]),
  [variablesˆvar='VALUES'ˆval],
  ['NVERTEX'>='VALUES'ˆomin,'NVERTEX'=<'VALUES'ˆomax],
  []).

ctr_eval(
  global_cardinality_low_up,
  [reformulation(global_cardinality_low_up_r)]).

ctr_contractible(global_cardinality_low_up, [], 'VALUES', any).

global_cardinality_low_up_r(VARIABLES, VALUES) :-
  length(VARIABLES, N),
  collection(VARIABLES, [dvar]),
  collection(VALUES, [int, int(0, N), int(0, N)]),
  ...
length(VALUES,M),
M>0,
get_attr1(VARIABLES,VARS),
get_attr1(VALUES,VALS),
get_attr2(VALUES,OMINS),
get_attr3(VALUES,OMAXS),
all_different(VALS),
get_minimum(VARS,MINVARS),
get_maximum(VARS,MAXVARS),
get_minimum(VALUES,MINVALS),
get_maximum(VALUES,MAXVALS),
MIN is min(MINVARS,MINVALS),
MAX is max(MAXVARS,MAXVALS),
complete_card_low_up(MIN,MAX,N,VALS,OMINS,OMAXS,VN),
global_cardinality(VARS,VN).
B.150  \texttt{global\_cardinality\_low\_up\_no\_loop}

\section*{Meta-Data:}

\begin{verbatim}
ctr_date(
    global_cardinality_low_up_no_loop,
    ['20051218','20060809']).

ctr_origin(
    global_cardinality_low_up_no_loop,
    Derived from \%c and \%c.,
    [global_cardinality_low_up,tree]).

ctr_synonyms(
    global_cardinality_low_up_no_loop,
    [gcc_low_up_no_loop]).

ctr_arguments(
    global_cardinality_low_up_no_loop,
    ['MINLOOP'-int,
     'MAXLOOP'-int,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int,omin-int,omax-int)]).

ctr_restrictions(
    global_cardinality_low_up_no_loop,
    ['MINLOOP']\geq0,
    ['MINLOOP']\leq['MAXLOOP',
    'MAXLOOP']\leq\size('VARIABLES'),
    required('VARIABLES',var),
    \size('VALUES')\geq0,
    required('VALUES',[val,omin,omax]),
    distinct('VALUES',val),
    'VALUES' `omin\geq0,
    'VALUES' `omax\leq\size('VARIABLES'),
    'VALUES' `omin\leq('VALUES' `omax)).

ctr_example(
    global_cardinality_low_up_no_loop,
    global_cardinality_low_up_no_loop(
      1,
      1,
      [[\text{var-1},\text{var-1},\text{var-8},\text{var-6}]],
      [[\text{val-1},\text{omin-1},\text{omax-1}],
      [[\text{val-5},\text{omin-0},\text{omax-0}],
      [\text{val-6},\text{omin-1},\text{omax-2}]]))).
\end{verbatim}
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

\begin{verbatim}
ctr_typical(
    global_cardinality_low_up_no_loop,
    [size('VARIABLES')>1,
     range('VARIABLES'\^var)>1,
     size('VALUES')>1,
     'VALUES'\^omin=<size('VARIABLES'),
     'VALUES'\^omax>0,
     'VALUES'\^omax<size('VARIABLES'),
     size('VARIABLES')>size('VALUES')]).

ctr_exchangeable(
    global_cardinality_low_up_no_loop,
    [items('VALUES',all),
     vals(['VALUES'\^omin],int(>(0)),>,dontcare,dontcare),
     vals(   ['VALUES'\^omax],
              int(=<(size('VARIABLES'))),
              <,dontcare,dontcare))).

ctr_graph(
    global_cardinality_low_up_no_loop,
    ['VARIABLES'],
    1,
    foreach('VALUES',['SELF'\>collection(variables)]),
    [variables\^var='VALUES'\^val,variables\^key='VALUES'\^val],
    ['NVERTEX'\>='VALUES'\^omin,'NVERTEX'\<='VALUES'\^omax],
    []).

ctr_graph(
    global_cardinality_low_up_no_loop,
    ['VARIABLES'],
    1,
    ['SELF'\>collection(variables)],
    [variables\^var=variables\^key],
    ['NARC'\>='MINLOOP','NARC'\<='MAXLOOP'],
    []).

ctr_eval(
    global_cardinality_low_up_no_loop,
    [reformulation(global_cardinality_low_up_no_loop_r)]).

global_cardinality_low_up_no_loop_r(
    MINLOOP,
   petsc_2hpetc)
\end{verbatim}
variables, values) :-
  check_type(int_gteq(0),MINLOOP),
  check_type(int_gteq(MINLOOP),MAXLOOP),
  collection(VARIABLES,[dvar]),
  length(VARIABLES,N),
  collection(VALUES,[int,int(0,N),int(0,N)]),
  length(VALUES,M),
  M>0,
  get_attr1(VARIABLES,VARS),
  get_attr1(VALUES,VALS),
  get_attr2(VALUES,OMINS),
  get_attr3(VALUES,OMAXS),
  all_different(VALS),
  gcc_no_loop1(VARS,1,SUMLOOP),
  call(SUMLOOP#>=MINLOOP),
  call(SUMLOOP#=<MAXLOOP),
  global_cardinality_low_up_no_loop1(1,M,N,VALS,OMINS,OMAXS,VARS).

global_cardinality_low_up_no_loop1(I,M,_,37127,[],[],[],_37131) :-
  I>M,
  !.

global_cardinality_low_up_no_loop1(I,M,RVAL,OMIN,OMAX,VARS) :-
  I=<M,
  gcc_no_loop2(1,N,I,VARS,VAL,OMIN),
  call(VAL#>=OMIN),
  call(VAL#=<OMAX),
  I1 is I+1,
  global_cardinality_low_up_no_loop1(I1,
M,
N,
RVAL,
ROMIN,
ROMAX,
VARS).
B.151  global_cardinality_no_loop

◊ Meta-Data:

ctr_date(global_cardinality_no_loop,['20051104','20060809']).

ctr_origin(
    global_cardinality_no_loop,
    Derived from %c and %c.,
    [global_cardinality,tree]).

ctr_synonyms(global_cardinality_no_loop,[gcc_no_loop]).

ctr_arguments(
    global_cardinality_no_loop,
    ['NLOOP'-dvar,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int,noccurrence-dvar)]).

ctr_restrictions(
    global_cardinality_no_loop,
    ['NLOOP']>=0,
    'NLOOP'=<size('VARIABLES'),
    required('VARIABLES',var),
    size('VALUES')>=0,
    required('VALUES',[val,noccurrence]),
    distinct('VALUES',val),
    'VALUES'\noccurrence>0,
    'VALUES'\noccurrence=<size('VARIABLES')).

ctr_example(
    global_cardinality_no_loop,
    global_cardinality_no_loop(1,
        [[var-1],[var-1],[var-8],[var-6]],
        [[val-1,noccurrence-1],
         [val-5,noccurrence-0],
         [val-6,noccurrence-1]]).

ctr_typical(
    global_cardinality_no_loop,
    [size('VARIABLES')>1,
     range('VARIABLES'\var)>1,
     size('VALUES')>1,
     size('VARIABLES')>size('VALUES')).
CTR_EXCHANGEABLE

CTR_GRAPH

CTR_GRAPH

CTR_EVAL

CTRPURE_FUNCTIONAL_DEPENDENCY

CTR_FUNCTIONAL_DEPENDENCY

CTR_FUNCTIONAL_DEPENDENCY

GLOBAL_CARDINALITY_NO_LOOP_R(NLOOP, VARIABLES, VALUES) :-
    check_type(dvar_gteq(0), NLOOP),
    collection(VARIABLES, [dvar]),
    length(VARIABLES, N),
    NLOOP#=<N,
    collection(VARIABLES, [int, dvar(0, N)]),
    length(VARIABLES, M),
    M>0,
    getAttr1(VARIABLES, VARS),
    getAttr1(VARIABLES, VALS),
    getAttr2(VARIABLES, NO henditions),
all_different(VALS),
gcc_no_loop1(VARS,1,SUMLOOP),
call(SUMLOOP#=NLOOP),
global_cardinality_no_loop1(
   1,
   M,
   N,
   VALS,
   NOCCURRENCES,
   VARS).

global_cardinality_no_loop1(I,M,_36389,[],[],_36392) :-
   I>M,
   !.

global_cardinality_no_loop1(
   I,
   M,
   N,
   [VAL|RVAL],
   [NOCCURRENCE|RNOCCURRENCE],
   VARS) :-
   I=<M,
   gcc_no_loop2(1,N,I,VARS,VAL,SUMI),
call(SUMI#=NOCCURRENCE),
   I1 is I+1,
   global_cardinality_no_loop1(
      I1,
      M,
      N,
      RVAL,
      RNOCCURRENCE,
      VARS).
B.152  global_cardinality_with_costs

◊ Meta-Data:

ctr_date(
  global_cardinality_with_costs,
  ['20030820','20040530','20060809','20090425']).

ctr_origin(global_cardinality_with_costs, '\cite{Regin99a}', []).

ctr_synonyms(global_cardinality_with_costs, [gcc, cost_gcc]).

ctr_arguments(
  global_cardinality_with_costs,
  ['VARIABLES'-collection(var-dvar),
   'VALUES'-collection(val-int,noccurrence-dvar),
   'MATRIX'-collection(i-int,j-int,c-int),
   'COST'-dvar]).

ctr_restrictions(
  global_cardinality_with_costs,
  [required('VARIABLES', var),
   size('VALUES')>0,
   required('VALUES', [val,noccurrence]),
   distinct('VALUES',val),
   'VALUES'~noccurrence>=0,
   'VALUES'~noccurrence=<size('VARIABLES'),
   required('MATRIX', [i,j,c]),
   increasing_seq('MATRIX', [i,j]),
   'MATRIX'~i=1,
   'MATRIX'~i=size('VARIABLES'),
   'MATRIX'~j=1,
   'MATRIX'~j=size('VALUES'),
   size('MATRIX')=size('VARIABLES')*size('VALUES')).

ctr_example(
  global_cardinality_with_costs,
  global_cardinality_with_costs(
    [[[var-3],[var-3],[var-3],[var-6]],
     [[var-3,noccurrence-3],
      [val-5,noccurrence-0],
      [val-6,noccurrence-1]],
     [[i-1,j-1,c-4],
      [i-1,j-2,c-1],
      [i-1,j-3,c-7],
      [i-2,j-1,c-1],
      [...]]).
[(i-2,j-2,c-0),
 (i-2,j-3,c-8),
 (i-3,j-1,c-3),
 (i-3,j-2,c-2),
 (i-3,j-3,c-1),
 (i-4,j-1,c-0),
 (i-4,j-2,c-0),
 (i-4,j-3,c-6),
 14)].

ctr_typical(
global_cardinality_with_costs,
 [size('VARIABLES')>1,
  range('VARIABLES'\^var)>1,
  size('VALUES')>1,
  range('VALUES'\^noccurrence)>1,
  range('MATRIX'\^c)>1,
  size('VARIABLES')>size('VALUES')]).

ctr_graph(
global_cardinality_with_costs,
 ['VARIABLES'],
 1,
  foreach('VALUES', ['SELF']>>collection(variables)),
  [variables\^var=values\^val],
  ['NVERTEX'=values\^noccurrence],
  []).

ctr_graph(
  global_cardinality_with_costs,
  ['VARIABLES','VALUES'],
  2,
  ['PRODUCT']>>collection(variables,values),
  [variables\^var=values\^val],
  ['SUM_WEIGHT_ARC'
   MATRIX@
   ((variables\^key-1)*size('VALUES')+values\^key)^c=
   COST],
  []).

ctr_eval(
global_cardinality_with_costs,
 [reformulation(global_cardinality_with_costs_r)]).

ctr_pure_functional_dependency(}
global_cardinality_with_costs([], []). 

ctr_functional_dependency(
  global_cardinality_with_costs,
  2-2,
  [1]). 

ctr_functional_dependency(
  global_cardinality_with_costs,
  4,
  [1,2,3]).

global_cardinality_with_costs_r(VARIABLES,VALUES,MATRIX,COST) :-
  collection(VARIABLES,[dvar]),
  length(VARIABLES,N),
  collection(VALUES,[int,dvar(0,N)]),
  length(VALUES,M),
  M>0,
  get_attr1(VARIABLES,VARS),
  get_attr1(VALUES,VALS),
  all_different(VALS),
  collection(MATRIX,[int(1,N),int(1,M),int]),
  collection_increasing_seq(MATRIX,[1,2]),
  eval(global_cardinality(VARIABLES,VALUES)),
  get_attr3(MATRIX,CS),
  global_cardinality_with_costs1(VARS,VALS,M,CS,TERM),
  call(TERM#=COST).

global_cardinality_with_costs1([],_47767,_47768,_47769,0).

global_cardinality_with_costs1([VAR|R],VALS,M,CMAT,C+S) :-
  global_cardinality_with_costs2(M,
    CMAT,
    ELEMTABLE,
    RESTCMAT),
  element(IVAL,VALS,VAR),
  element(IVAL,ELEMTABLE,C),
  global_cardinality_with_costs1(R,VALS,M,RESTCMAT,S).

global_cardinality_with_costs2(0,CMAT,[],CMAT) :- !.

global_cardinality_with_costs2(I,[C|R],[C|S],T) :-
  I>0,
II is I-1,
global_cardinality_with_costs2(I1,R,S,T).
B.153  global_contiguity

◊ Meta-Data:

ctr_date(global_contiguity, ['20030820', '20040530', '20060809']).

ctr_origin(global_contiguity, '\cite{Maher02}', []).

ctr_synonyms(global_contiguity, [contiguity]).

ctr_arguments(
  global_contiguity,
  ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
  global_contiguity,
  [required('VARIABLES', var),
   'VARIABLES' \textasciitilde var \geq 0,
   'VARIABLES' \textasciitilde var \leq 1]).

ctr_example(
  global_contiguity,
  global_contiguity([[var-0], [var-1], [var-1], [var-0]])).

ctr_typical(
  global_contiguity,
  [size('VARIABLES') \geq 2, range('VARIABLES' \textasciitilde var \geq 1)]).

ctr_exchangeable(
  global_contiguity,
  [items('VARIABLES', reverse)]).

ctr_graph(
  global_contiguity,
  ['VARIABLES'],
  2,
  ['PATH' \textasciitilde collection(variables1, variables2),
   'LOOP' \textasciitilde collection(variables1, variables2)],
  [variables1 \textasciitilde var \textasciitilde variables2 \textasciitilde var,
   variables1 \textasciitilde var = 1],
  ['NCC' = 1],
  []).

ctr_eval(
  global_contiguity,
  [checker(global_contiguity_c),
   automaton(global_contiguity_a)]).
ctr_contractible(global_contiguity,[],'VARIABLES',any).

global_contiguity_c([]) :- !.

global_contiguity_c(VARIABLES) :-
    collection(VARIABLES,[int(0,1)]),
    get_attr1(VARIABLES,VARS),
    global_contiguity_c1(VARS).

global_contiguity_c1([]) :- !.

global_contiguity_c1([0|R]) :- !, global_contiguity_c1(R).

global_contiguity_c1([1|R]) :-
    global_contiguity_c2(R).

global_contiguity_c2([]) :- !.

global_contiguity_c2([1|R]) :- !, global_contiguity_c2(R).

global_contiguity_c2([0|R]) :-
    global_contiguity_c3(R).

global_contiguity_c3([]) :- !.

global_contiguity_c3([0|R]) :-
    global_contiguity_c3(R).

global_contiguity_a(1,[]) :- !.

global_contiguity_a(0,[]) :- !,
    fail.

global_contiguity_a(FLAG,VARIABLES) :-
    collection(VARIABLES,[dvar(0,1)]),
get_attr1(VARIABLES, LIST_VARIABLES),
AUTOMATON=
automaton(
    LIST_VARIABLES, _32003,
    LIST_VARIABLES, [source(s), sink(m), sink(z), sink(s)],
    [arc(s,0,s), arc(s,1,m), arc(m,0,z), arc(m,1,m), arc(z,0,z)], [ ], [ ], []),
automaton_bool(FLAG, [0,1], AUTOMATON).
B.154 golomb

◊ **META-DATA:**

```prolog
ctr_date(golomb,['20000128','20030820','20040530','20060809']).

ctr_origin(golomb,'Inspired by \cite{Golomb72}.',[]).

ctr_arguments(golomb,['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
  golomb,
  [required('VARIABLES',var),
   'VARIABLES'\^var>=0,
   strictly_increasing('VARIABLES')].

ctr_example(golomb,golomb([[var-0],[var-1],[var-4],[var-6]])).

ctr_typical(golomb,[size('VARIABLES')>2]).

ctr_exchangeable(golomb,[translate(['VARIABLES'\^var])]).

ctr_derived_collections(
  golomb,
  [col('PAIRS'-collection(x-dvar,y-dvar),
    [> -item(x-'VARIABLES'\^var,y-'VARIABLES'\^var)])].

ctr_graph(
  golomb,
  ['PAIRS'],
  2,
  ['CLIQUE'\>>collection(pairs1,pairs2)],
  [pairs1\^y-pairs1\^x=pairs2\^y-pairs2\^x],
  ['MAX_NSCC'=<1],
  []).

ctr_eval(golomb,[checker(golomb_c),reformulation(golomb_r)]).

ctr_contractible(golomb,[],'VARIABLES',any).

golomb_c([]) :-
  !.

golomb_c(VARIABLES) :-
  collection(VARIABLES,[dvar_gteq(0)]),
  collection_increasing_seq(VARIABLES,[1]),
  collection_increasing_seq(VARIABLES,[1]),
get_attr1(VARIABLES,VARS),
golomb3(VARS,D),
sort(D,SD),
length(D,N),
length(SD,N).

golomb_r([]) :-
    !.

golomb_r(VARIABLES) :-
collection(VARIABLES,[dvar_gteq(0)]),
collection_increasing_seq(VARIABLES,[1]),
get_attr1(VARIABLES,VARS),
golomb1(VARS,D),
all_different(D).

golomb1([_33420],[]) :-
    !.

golomb1([U,V|R],Diffs) :-
golomb2([V|R],U,D),
golomb1([V|R],Diff),
append(D,Diff,Diffs).

golomb2([],_33416,[]).

golomb2([Vi|R],Vj,[D|S]) :-
    D#=Vi-Vj,
golomb2(R,Vj,S).

golomb3([_33420],[]) :-
    !.

golomb3([U,V|R],Diffs) :-
golomb4([V|R],U,D),
golomb3([V|R],Diff),
append(D,Diff,Diffs).

golomb4([],_33416,[]).

golomb4([Vi|R],Vj,[D|S]) :-
    D is Vi-Vj,
golomb4(R,Vj,S).
B.155 graph_crossing

◊ **META-DATA:**

```prolog
ctr_date(
  graph_crossing,
  ['20000128','20030820','20040530','20060809']).

ctr_origin(graph_crossing,'N.˘Beldiceanu',[]).

ctr_synonyms(graph_crossing,[crossing,ncross]).

ctr_arguments(
  graph_crossing,
  ['NCROSS'-dvar,'NODES'-collection(succ-dvar,x-int,y-int)]).

ctr_restrictions(
  graph_crossing,
  ['NCROSS'>=0,
   required('NODES',[succ,x,y]),
   'NODES'\succ>=1,
   'NODES'\succ=<size('NODES'))).

ctr_example(
  graph_crossing,
  graph_crossing(2,
    [[succ-1,x-4,y-7],
     [succ-1,x-2,y-5],
     [succ-1,x-7,y-6],
     [succ-2,x-1,y-2],
     [succ-3,x-2,y-2],
     [succ-2,x-5,y-3],
     [succ-3,x-8,y-2],
     [succ-9,x-6,y-2],
     [succ-10,x-10,y-6],
     [succ-8,x-10,y-1]]).

ctr_typical(
  graph_crossing,
  [size('NODES')>1,
   range('NODES'\succ)>1,
   range('NODES'\^x)>1,
   range('NODES'\^y)>1]).

ctr_exchangeable(_:0).
```
graph_crossing,
    [attrs_sync('NODES',[[succ],[x,y]]),
    translate([['NODES'~x]),
    translate([['NODES'~y])]).

ctr_graph(
    graph_crossing,
    ['NODES'],
    2,
    ['CLIQUE'(<)>>collection(n1,n2)],
    [max(n1~x,'NODES'@(n1~succ)~x)>=
    min(n2~x,'NODES'@(n2~succ)~x),
    max(n2~x,'NODES'@(n2~succ)~x)>=
    min(n1~x,'NODES'@(n1~succ)~x),
    max(n1~y,'NODES'@(n1~succ)~y)>=
    min(n2~y,'NODES'@(n2~succ)~y),
    max(n2~y,'NODES'@(n2~succ)~y)>=
    min(n1~y,'NODES'@(n1~succ)~y),
    (n2~x-'NODES'@(n1~succ)~x)*
    ('NODES'@(n1~succ)~y-n1~y)=\= 0,
    (n2~y-n1~y)=
    (n2~x-n1~x)*('NODES'@(n2~succ)~y-'NODES'@(n1~succ)~y)=\= 0,
    sign(
    (n2~x-'NODES'@(n1~succ)~x)*
    ('NODES'@(n1~succ)~y-n1~y)-
    ('NODES'@(n1~succ)~x-n1~x)*
    (n2~y-'NODES'@(n1~succ)~y))=\=
    sign(
    ('NODES'@(n2~succ)~x-'NODES'@(n1~succ)~x)*
    (n2~y-n1~y)-
    (n2~x-n1~x)*('NODES'@(n2~succ)~y-'NODES'@(n1~succ)~y))],
    ['NARC'='NCROSS'],
    []).

ctr_pure_functional_dependency(graph_crossing,[]).

ctr_functional_dependency(graph_crossing,1,[2]).
B.156 graph_isomorphism

◊ META-DATA:

ctr_predefined(graph_isomorphism).

ctr_date(graph_isomorphism, ['20090822']).

ctr_origin(graph_isomorphism, '\cite{Gregor79}', []).

ctr_arguments(
    graph_isomorphism,
    ['NODES_PATTERN' -collection(index-int, succ-sint),
     'NODES_TARGET' -collection(index-int, succ-sint),
     'FUNCTION' -collection(image-dvar)]).

ctr_restrictions(
    graph_isomorphism,
    [required('NODES_PATTERN', [index, succ]),
     'NODES_PATTERN'~index>=1,
     'NODES_PATTERN'~index=<size('NODES_PATTERN'),
     distinct('NODES_PATTERN', index),
     'NODES_PATTERN'~succ>=1,
     'NODES_PATTERN'~succ=<size('NODES_PATTERN'),
     required('NODES_TARGET', [index, succ]),
     'NODES_TARGET'~index>=1,
     'NODES_TARGET'~index=<size('NODES_TARGET'),
     distinct('NODES_TARGET', index),
     'NODES_TARGET'~succ>=1,
     'NODES_TARGET'~succ=<size('NODES_TARGET'),
     size('NODES_TARGET')=size('NODES_PATTERN'),
     required('FUNCTION', [image]),
     'FUNCTION'~image>=1,
     'FUNCTION'~image=<size('NODES_TARGET'),
     distinct('FUNCTION', image),
     size('FUNCTION')=size('NODES_PATTERN')].

ctr_example(
    graph_isomorphism,
    graph_isomorphism(
        [[index-1, succ-{2,4}],
         [index-2, succ-{1,3,4}],
         [index-3, succ-{}],
         [index-4, succ-{}]],
        [[index-1, succ-{}],
         [index-2, succ-{1,3,4}]],
        [[index-1, succ-{2,4}],
         [index-2, succ-{1,3,4}],
         [index-3, succ-{}],
         [index-4, succ-{}]]).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

\[
\begin{align*}
&[\text{index-3}, \text{succ-\{\}}], \\
&[\text{index-4}, \text{succ-\{1,2\}}], \\
&[[\text{image-4}], [\text{image-2}], [\text{image-3}], [\text{image-1}]]).
\end{align*}
\]

\text{ctr\_typical}(\text{graph\_isomorphism}, [\text{size(’NODES\_PATTERN’)>1}]).

\text{ctr\_exchangeable(}
  \begin{align*}
  &\text{graph\_isomorphism}, \\
  &[\text{items(’NODES\_PATTERN’, all)}, \text{items(’NODES\_TARGET’, all)}].
  \end{align*}
\)
B.157 group

◊ Meta-Data:

```lisp
ctr_date(group, ['20000128', '20030820', '20040530', '20060809']).

ctr_origin(group, \\index{CHIP|indexuse}CHIP', []).

ctr_arguments(
  group,
  ['NGROUP'-dvar,
   'MIN_SIZE'-dvar,
   'MAX_SIZE'-dvar,
   'MIN_DIST'-dvar,
   'MAX_DIST'-dvar,
   'NVAL'-dvar,
   'VARIABLES'-collection(var-dvar),
   'VALUES'-collection(val-int)]).

ctr_restrictions(
  group,
  ['NGROUP'>=0,
   'MIN_SIZE'>=0,
   'MAX_SIZE'>='MIN_SIZE',
   'MIN_DIST'>=0,
   'MAX_DIST'>='MIN_DIST',
   'MAX_DIST'=<size('VARIABLES'),
   'NVAL'>='MAX_SIZE',
   'NVAL'>='NGROUP',
   'NVAL'=<size('VARIABLES'),
   required('VARIABLES',var),
   required('VALUES',val),
   distinct('VALUES',val))].

ctr_example(
  group,
  group(2,
    1,
    2,
    2,
    4,
    3,
    [[var-2], [var-8], [var-1],
    [var-3], [var-7], [var-1]],
    [[var-4], [var-5], [var-6]]).
```
```prolog
ctr_typical(
    group,
    ['NGROUP'>0,
    'MIN_SIZE'>0,
    'MAX_SIZE'>='MIN_SIZE',
    'MIN_DIST'>0,
    'MAX_DIST'>='MIN_DIST',
    'MAX_DIST'<size('VARIABLES'),
    'NVAL'>='MAX_SIZE',
    'NVAL'>='NGROUP',
    'NVAL'<size('VARIABLES'),
    size('VARIABLES')>1,
    range('VARIABLES'ˆvar)>1,
    size('VALUES')>0,
    size('VARIABLES')>size('VALUES'))) .

ctr_exchangeable(
    group,
    [items('VARIABLES',reverse),
     items('VALUES',all),
     vals(
        ['VARIABLES'ˆvar],
        comp('VALUES'ˆval),
        =,
        dontcare,
        dontcare))) .

ctr_graph(
    group,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2),
     'LOOP'>>collection(variables1,variables2)],
    [variables1ˆvar in 'VALUES',variables2ˆvar in 'VALUES'],
    ['NCC'='NGROUP',
     'MIN_NCC'='MIN_SIZE',
     'MAX_NCC'='MAX_SIZE',
     'NVERTEX'='NVAL']) .
```
ctr_graph(
    group,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2),
     'LOOP'>>collection(variables1,variables2)],
    [not_in(variables1^var,'VALUES'),
     not_in(variables2^var,'VALUES')],
    ['MIN_NCC'='MIN_DIST','MAX_NCC'='MAX_DIST'],
    []).

ctr_eval(group,[automata(group_a)]).

ctr_functional_dependency(group,1,[7,8]).

ctr_functional_dependency(group,2,[7,8]).

ctr_functional_dependency(group,3,[7,8]).

ctr_functional_dependency(group,4,[7,8]).

ctr_functional_dependency(group,5,[7,8]).

ctr_functional_dependency(group,6,[7,8]).

group_a(
    NGROUP,
    MIN_SIZE,
    MAX_SIZE,
    MIN_DIST,
    MAX_DIST,
    NVAL,
    VARIABLES,
    VALUES) :-
    check_type(dvar,NGROUP),
    check_type(dvar,MIN_SIZE),
    check_type(dvar,MAX_SIZE),
    check_type(dvar,MIN_DIST),
    check_type(dvar,MAX_DIST),
    check_type(dvar,NVAL),
    collection(VARIABLES,[dvar]),
    collection(VALUES,[int]),
    length(VARIABLES,N),
    get_attr1(VALUES,VALS),
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

NGROUP#>=0,
MIN_SIZE#>=0,
MAX_SIZE#>=MIN_SIZE,
MIN_DIST#>=0,
MAX_DIST#>=MIN_DIST,
MAX_DIST#=<N,
NVAL#>=MAX_SIZE,
NVAL#>=NGROUP,
NVAL#=<N,
all_different(VALS),

group_ngroup(NGROUP,VARIABLES,VALUES),
group_min_size(MIN_SIZE,VARIABLES,VALUES),
group_max_size(MAX_SIZE,VARIABLES,VALUES),
group_min_dist(MIN_DIST,VARIABLES,VALUES),
group_max_dist(MAX_DIST,VARIABLES,VALUES),
group_nval(NVAL,VARIABLES,VALUES).

group_ngroup(NGROUP,VARIABLES,VALUES) :-
get_attr1(VALUES,LIST_VALUES),
list_to_fdset(LIST_VALUES,SET_OF_VALUES),
group_signature_in(VARIABLES,SIGNATURE,SET_OF_VALUES),
automaton(
    SIGNATURE,
    _52392,
    SIGNATURE,
    [source(s),sink(i),sink(s)],
    [arc(s,0,s),
     arc(s,1,i,[C+1]),
     arc(i,1,i),
     arc(i,0,s)],
    [C],
    [0],
    [NGROUP]).

group_min_size(MIN_SIZE,VARIABLES,VALUES) :-
length(VARIABLES,NVAR),
get_attr1(VALUES,LIST_VALUES),
list_to_fdset(LIST_VALUES,SET_OF_VALUES),
group_signature_in(VARIABLES,SIGNATURE,SET_OF_VALUES),
MIN_SIZE#=min(C1,D1),
automaton(
    SIGNATURE,
    _52882,
    SIGNATURE,
    [source(s),sink(j),sink(k),sink(s)],
    [arc(s,0,s),
     arc(s,1,j),
     arc(s,2,k),
     arc(s,3,s)],
    [j,k],
    [0],
    [NGROUP],
    [MIN_SIZE]).
arc(s, 1, j, [NVAR, D]),
arc(j, 1, j, [C, D+1]),
arc(j, 0, k, [min(C, D), D]),
arc(k, 0, k),
arc(k, 1, j, [C, 1]),
[C, D],
[0, 1],
[C1, D1]).

group_max_size(MAX_SIZE, VARIABLES, VALUES) :-
get_attr1(VALUES, LIST_VALUES),
list_to_fdset(LIST_VALUES, SET_OF_VALUES),
group_signature_in(VARIABLES, SIGNATURE, SET_OF_VALUES),
MAX_SIZE# = max(C1, D1),
automaton(
  SIGNATURE,
  _52600,
  SIGNATURE,
  [source(s), sink(s)],
  [arc(s, 1, s, [C, D+1]), arc(s, 0, s, [max(C, D), 0])],
  [C, D],
  [0, 0],
  [C1, D1]).

group_min_dist(MIN_DIST, VARIABLES, VALUES) :-
length(VARIABLES, NVAR),
get_attr1(VALUES, LIST_VALUES),
list_to_fdset(LIST_VALUES, SET_OF_VALUES),
group_signature_not_in(VARIABLES, SIGNATURE, SET_OF_VALUES),
MIN_DIST# = min(C1, D1),
automaton(
  SIGNATURE,
  _53141,
  SIGNATURE,
  [source(s), sink(j), sink(k), sink(s)],
  [arc(s, 0, s),
   arc(s, 1, j, [NVAR, D]),
   arc(j, 1, j, [C, D+1]),
   arc(j, 0, k, [min(C, D), D]),
   arc(k, 0, k),
   arc(k, 1, j, [C, 1])],
  [C, D],
  [0, 1],
  [C1, D1]).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

[group_max_dist(MAX_DIST,VARIABLES,VALUES) :-
  get_attr1(VALUES,LIST_VALUES),
  list_to_fdset(LIST_VALUES,SET_OF_VALUES),
  group_signature_not_in(
      VARIABLES,
      SIGNATURE,
      SET_OF_VALUES),
  MAX_DIST#=max(C1,D1),
  automaton(
      SIGNATURE,
      _52859,
      SIGNATURE,
      [source(s),sink(s)],
      [arc(s,1,s,[C,D+1]),arc(s,0,s,[max(C,D),0])],
      [C,D],
      [0,0],
      [C1,D1]).

[group_nval(NVAL,VARIABLES,VALUES) :-
  get_attr1(VALUES,LIST_VALUES),
  list_to_fdset(LIST_VALUES,SET_OF_VALUES),
  group_signature_in(VARIABLES,SIGNATURE,SET_OF_VALUES),
  automaton(
      SIGNATURE,
      _52350,
      SIGNATURE,
      [source(s),sink(s)],
      [arc(s,0,s),arc(s,1,s,[C+1])],
      [C],
      [0],
      [NVAL]).

  group_signature_in([],[],_51245).
  group_signature_in([[[var-VAR]|VARs],[S|Ss],SET_OF_VALUES) :-
    VAR in_set SET_OF_VALUES#<=S,
    group_signature_in(VARs,Ss,SET_OF_VALUES).

  group_signature_not_in([],[],_51245).
  group_signature_not_in([[[var-VAR]|VARs],[S|Ss],SET_OF_VALUES) :-
    VAR in_set SET_OF_VALUES#=> #\S,
    group_signature_not_in(VARs,Ss,SET_OF_VALUES).
B.158 group_skip_isolated_item

◊ **META-DATA:**

```prolog
ctr_date(
    group_skip_isolated_item,
    ['20000128','20030820','20040530','20060809']).
```

```prolog
ctr_origin(group_skip_isolated_item,'Derived from %c. ',[group]).
```

```prolog
ctr_arguments(
    group_skip_isolated_item,
    ['NGROUP'-dvar,
     'MIN_SIZE'-dvar,
     'MAX_SIZE'-dvar,
     'NVAL'-dvar,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int)]).
```

```prolog
ctr_restrictions(
    group_skip_isolated_item,
    ['NGROUP'>=0,
     'MIN_SIZE'>=0,
     'MAX_SIZE'>='MIN_SIZE',
     'NVAL'>='MAX_SIZE',
     'NVAL'>='NGROUP',
     'NVAL'<size('VARIABLES'),
     required('VARIABLES',var),
     required('VALUES',val),
     distinct('VALUES',val))].
```

```prolog
ctr_example(
    group_skip_isolated_item,
    group_skip_isolated_item(1, 2, 2, 3, [[var-2], [var-8], [var-1], [var-7], [var-4], [var-5], [var-1], [var-1]]),
```


APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

[\text{var-1}],
[[\text{val-0}],[\text{val-2}],[\text{val-4}],[\text{val-6}],[\text{val-8}])]\).

\text{ctr\_typical}(
\begin{array}
\text{group\_skip\_isolated\_item}, \\
[\{
\text{NGROUP}\}>0, \\
\text{MIN\_SIZE}\}>0, \\
\text{NVAL}\rangle\text{MAX\_SIZE}', \\
\text{NVAL}\rangle\text{NGROUP}', \\
\text{NVAL}\langle\text{size('VARIABLES')}, \\
\text{size('VARIABLES')}\rangle1, \\
\text{range('VARIABLES'\text{\_var})}\rangle1, \\
\text{size('VALUES')}\rangle0, \\
\text{size('VARIABLES')}\rangle\text{size('VALUES')}]\).
\end{array}

\text{ctr\_exchangeable}(
\begin{array}
\text{group\_skip\_isolated\_item}, \\
[\text{items('VARIABLES',\text{reverse})}, \\
\text{items('VALUES',\text{all})}, \\
\text{vals}(
[\text{\text{'VARIABLES'\_var}}, \\
\text{\text{comp('VALUES'\_val)}, \\
=, \\
\text{dont\_care, \\
\text{dont\_care}])}. \\
\end{array}

\text{ctr\_graph}(
\begin{array}
\text{group\_skip\_isolated\_item}, \\
[\text{\text{VARIABLES'}}, \\
2, \\
[\text{\text{CHAIN'}}\rangle\text{\text{collection('VARIABLES1',\text{variables2})}}, \\
[\text{\text{'VARIABLES1'\_var}}\text{\text{in 'VALUES'},\text{\text{variables2\_var}}\text{\text{in 'VALUES'}}, \\
[\text{\text{NSCC'}}\text{\text{='NGROUP'}}, \\
\text{\text{MIN\_NSCC'}}\text{\text{='MIN\_SIZE'}}, \\
\text{\text{MAX\_NSCC'}}\text{\text{='MAX\_SIZE'}}, \\
\text{\text{\text{N\_VERTEX'}}}\text{\text{='NVAL'}}, \\
[]]. \\
\end{array}

\text{ctr\_eval}(
\begin{array}
\text{group\_skip\_isolated\_item}, \\
[\text{\text{\text{automata('group\_skip\_isolated\_item\_a'))}}]. \\
\end{array}

\text{ctr\_functional\_dependency}(\text{group\_skip\_isolated\_item},1,[5,6]).

\text{ctr\_functional\_dependency}(\text{group\_skip\_isolated\_item},2,[5,6]).
ctr_functional_dependency(group_skip_isolated_item, 3, [5, 6]).
ctr_functional_dependency(group_skip_isolated_item, 4, [5, 6]).
group_skip_isolated_item_a(NGROUP, MIN_SIZE, MAX_SIZE, NVAL, VARIABLES, VALUES) :-
  check_type(dvar,NGROUP),
  check_type(dvar,MIN_SIZE),
  check_type(dvar,MAX_SIZE),
  check_type(dvar,NVAL),
  collection(VARIABLES,[dvar]),
  collection(VALUES,[int]),
  length(VARIABLES,N),
  get_attr1(VALUES,VALS),
  NGROUP#>=0,
  MIN_SIZE#>=0,
  MAX_SIZE#>=MIN_SIZE,
  NVAL#>=MAX_SIZE,
  NVAL#>=NGROUP,
  NVAL#=<N,
  all_different(VALS),
  group_skip_isolated_item_ngroup(NGROUP, VARIABLES, VALUES),
  group_skip_isolated_item_min_size(MIN_SIZE, VARIABLES, VALUES),
  group_skip_isolated_item_max_size(MAX_SIZE, VARIABLES, VALUES),
  group_skip_isolated_item_nval(NVAL, VARIABLES, VALUES).

group_skip_isolated_item_ngroup(NGROUP, VARIABLES, VALUES) :-
  get_attr1(VALUES,LIST_VALUES),
  list_to_fdset(LIST_VALUES,SET_OF_VALUES),
  group_skip_isolated_item_signature(VARIABLES, VALUES, SIGNATURE,
set_of_values),
automaton(
  signature,
_40963,
signature,
[source(s),sink(i),sink(j),sink(s)],
[arc(s,0,s),
arcs,1,i),
arcs,0,s),
arcs,1,j,[C+1]),
arcs(j,1,j),
arcs(j,0,s]),
[C],
[0],
[ngroup]).
group_skip_isolated_item_min_size(MIN_SIZE,VARIABLES,VALUES) :-
  length(VARIABLES,NVAR),
  get_attr1(VALUES,LIST_VALUES),
  list_to_fdset(LIST_VALUES,SET_OF_VALUES),
  group_skip_isolated_item_signature(
    VARIABLES,
    signature,
    set_of_values),
  MIN_SIZE#=min(C1,D1),
automaton(
  signature,
_41495,
signature,
[source(s),
sink(j),
sink(k),
sink(l),
sink(m),
sink(s)],
[arc(s,0,s),
arcs,1,j),
arcs,0,s),
arcs(j,1,k,[NVAR,D]),
arcs(k,1,k,[C,D+1]),
arcs(k,0,l,[min(C,D),D]),
arcs(l,0,l),
arcs(l,1,m),
arcs(m,0,l),
arcs(m,1,k,[C,2]),
[C,D],
group_skip_isolated_item_max_size(MAX_SIZE,VARIABLES,VALUES) :-
  get_attr1(VALUES,LIST_VALUES),
  list_to_fdset(LIST_VALUES,SET_OF_VALUES),
  group_skip_isolated_item_signature( VARIABLES, SIGNATURE, SET_OF_VALUES),
  MAX_SIZE#=max(C1,D1),
  automaton( SIGNATURE, _41188, SIGNATURE, [source(s),sink(i),sink(s)],
             [arc(s,0,s),
              arc(s,1,i,[C,1]),
              arc(i,0,s,[max(C,D),D]),
              arc(i,1,i,[C,D+1])],
             [C,D],
             [0,0],
             [C1,D1]).

group_skip_isolated_item_nval(NVAL,VARIABLES,VALUES) :-
  get_attr1(VALUES,LIST_VALUES),
  list_to_fdset(LIST_VALUES,SET_OF_VALUES),
  group_skip_isolated_item_signature( VARIABLES, SIGNATURE, SET_OF_VALUES),
  automaton( SIGNATURE, _40879, SIGNATURE, [source(s),sink(s)],
             [arc(s,0,s),arc(s,1,s,[C+1])],
             [C],
             [0],
             [NVAL]).

group_skip_isolated_item_signature([],[],_39515).

group_skip_isolated_item_signature( [[var-VAR]|VARs],
                                       [S|Ss],
SET_OF_VALUES) :-
    VAR in_set SET_OF_VALUES#<=>S,
    group_skip_isolated_item_signature(
        VARs,
        Ss,
        SET_OF_VALUES).
B.159  gt

◇ Meta-Data:

ctr_predefined(gt).

ctr_date(gt, [’20070821’]).

ctr_origin(gt, ’Arithmetic.’, []).

ctr_synonyms(gt, [rel, xgty]).

ctr_arguments(gt, [’VAR1’-dvar, ’VAR2’-dvar]).

ctr_example(gt, gt(8, 1)).

ctr_exchangeable(
  gt,
  [vals([’VAR1’], int(>’VAR2’)), =\=, all, dontcare),
  vals([’VAR2’], int(<’VAR1’)), =\=, all, dontcare)]).

ctr_eval(gt, [builtin(gt_b)]).

gt_b(VAR1, VAR2) :-
  check_type(dvar, VAR1),
  check_type(dvar, VAR2),
  VAR1#>VAR2.
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.160 highest_peak

◊ Meta-Data:

ctr_date(highest_peak,[’20040530’]).

ctr_origin(highest_peak,’Derived from %c.’,[peak]).

ctr_arguments(
  highest_peak,
  [’HEIGHT’-dvar,’VARIABLES’-collection(var-dvar)]).

ctr_restrictions(
  highest_peak,
  [’HEIGHT’>=0,’VARIABLES’^var>=0,required(’VARIABLES’,var)]).

ctr_example(
  highest_peak,
  highest_peak(8,
    [[var-1],
     [var-1],
     [var-4],
     [var-8],
     [var-6],
     [var-2],
     [var-7],
     [var-1]])).

ctr_typical(
  highest_peak,
  [’HEIGHT’>0,size(’VARIABLES’)>2,range(’VARIABLES’^var)>1]).

ctr_exchangeable(highest_peak,[items(’VARIABLES’,reverse)]).

ctr_eval(highest_peak,[automaton(highest_peak_a)]).

highest_peak_a(FLAG,HEIGHT,VARIABLES) :-
  check_type(dvar_gteq(0),HEIGHT),
  collection(VARIABLES,[dvar_gteq(0)]),
  highest_peak_signature(VARIABLES,SIGNALATURE,PAIRS),
  automaton(
    PAIRS,
    VAR1_-VAR2,
    SIGNALATURE,
    [source(s),sink(u),sink(s)].,
highest_peak_signature([],[],[]).

highest_peak_signature([_13976],[],[]) :- !.

highest_peak_signature([VAR1,VAR2|VARS],S,Ss,PAIRS) :-
  S in 0..2,
  VAR1#>VAR2#<=>S#=0,
  VAR1#=VAR2#<=>S#=1,
  VAR1<VAR2#<=>S#=2,
  highest_peak_signature([[VAR2]|VARS],Ss,PAIRS).
B.161  imply

◊ Meta-Data:

ctr_date(imply, ['20051226', '20091016']).

ctr_origin(imply, 'Logic', []).

ctr_synonyms(imply, [rel, ifthen]).

ctr_arguments(
  imply,
  ['VAR'-dvar, 'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
  imply,
  ['VAR']>=0,
  'VAR'=<1,
  size('VARIABLES')=2,
  required('VARIABLES', var),
  'VARIABLES'~var>=0,
  'VARIABLES'~var=<1).

ctr_example(
  imply,
  [imply(1, [[var-0], [var-0]]),
   imply(1, [[var-0], [var-1]]),
   imply(0, [[var-1], [var-0]]),
   imply(1, [[var-1], [var-1]])].

ctr_exchangeable(
  imply,
  [vals([VAR', 'VARIABLES'~var], int(0 in 1), <, all, dontcare)]).

ctr_eval(imply, [reformulation(imply_r), automaton(imply_a)]).

ctr_pure_functional_dependency(imply, []).

ctr_functional_dependency(imply, 1, [2]).

imply_r(VAR, VARIABLES) :-
  check_type(dvar(0, 1), VAR),
  collection(VARIABLES, [dvar(0, 1)]),
  length(VARIABLES, 2),
  get_attr1(VARIABLES, VARS),
  VARS=[VAR1, VAR2],
VAR1\Rightarrow VAR2.

\texttt{ imply\_a(FLAG,VAR,VARIABLES) :-}
\texttt{ check\_type(dvar(0,1),VAR),}
\texttt{ collection(VARIABLES,[dvar(0,1)]),}
\texttt{ length(VARIABLES,2),}
\texttt{ get\_attr1(VARIABLES,LIST),}
\texttt{ append([VAR],LIST,LIST\_VARIABLES),}
\texttt{ AUTOMATON=
    automaton(
        LIST\_VARIABLES,
        _20420,
        LIST\_VARIABLES,
        [source(s),sink(t)],
        [arc(s,0,i),
         arc(s,1,j),
         arc(i,1,k),
         arc(j,0,t),
         arc(j,1,l),
         arc(k,0,t),
         arc(l,1,t),
         arc(t,0,t),
         arc(t,1,t)],
        [],
        [],
        []),
    automaton\_bool(FLAG,[0,1],AUTOMATON).}
B.162 in

◊ **Meta-Data:**

```prolog
ctr_date(in,['20030820','20040530','20060810']).
ctr_origin(in,'Domain definition.',[]).
ctr_synonyms(in,[dom,in_set,member]).
ctr_arguments(in,['VAR'-dvar,'VALUES'-collection(val-int)]).
ctr_restrictions(
in,
[size('VALUES')>0,
 required('VALUES',val),
 distinct('VALUES',val)]).
ctr_example(in,3 in[[val-1],[val-3]]).
ctr_typical(in,[size('VALUES')>1]).
ctr_exchangeable(
in,
[items('VALUES',all),
 vals(['VAR'],int(['VAR','VALUES'\^val]),=\=,all,dontcare),
 translate(['VAR','VALUES'\^val])]).
ctr_derived_collections(
in,
[col('VARIABLES'-collection(var-dvar),[item(var-'VAR')]])).
ctr_graph(
in,
['VARIABLES','VALUES'],
2,
['PRODUCT'>>collection(variables,values)],
[variables\^var=values\^val],
['NARC'=1],
[]).
ctr_eval(in,[reformulation(in_r),automaton(in_a)]).
ctr_extensible(in,[],'VALUES',any).
in_r(VAR,VALUES) :-
```
check_type(dvar, VAR),
collection(VALUES, [int]),
length(VALUES, L),
L > 0,
get_attr1(VALUES, VALS),
all_different(VALS),
build_or_var_in_values(VALS, VAR, TERM),
call(TERM).

in_a(FLAG, VAR, VALUES) :-
  check_type(dvar, VAR),
collection(VALUES, [int]),
length(VALUES, L),
L > 0,
get_attr1(VALUES, VALS),
all_different(VALS),
in_signature(VALUES, SIGNATURE, VAR),
AUTOMATON =
  automaton(
    SIGNATURE,
    _38467,
    SIGNATURE,
    [source(s), sink(t)],
    [arc(s, 0, s), arc(s, 1, t), arc(t, 0, t)],
    [],
    [],
    []),
  automaton_bool(FLAG, [0, 1], AUTOMATON).

in_signature([], [], _36450).

in_signature([[val-VAL] | VALs], [S | Ss], VAR) :-
  VAR# = VAL# <==> S,
in_signature(VALs, Ss, VAR).
B.163 in_interval

◊ Meta-Data:

\[
\begin{align*}
\text{ctr\_date} & \text{(in\_interval, } \{\text{'20060317', '20060810'}\}\}. \\
\text{ctr\_origin} & \text{(in\_interval, 'Domain definition.', [])}. \\
\text{ctr\_synonyms} & \text{(in\_interval, [\text{dom, in}]).} \\
\text{ctr\_arguments} & \text{(in\_interval, ['VAR'-dvar,'LOW'-int,'UP'-int]).} \\
\text{ctr\_restrictions} & \text{(in\_interval, ['LOW'=<'UP'].)} \\
\text{ctr\_example} & \text{(in\_interval, in\_interval(3,2,5)).} \\
\text{ctr\_typical} & \text{(in\_interval, ['LOW'<'UP','VAR'>'LOW','VAR'<'UP'].)} \\
\text{ctr\_exchangeable} & \text{(in\_interval,} \\
& \text{vals([\text{'LOW'},\text{int},>,\text{dontcare},\text{dontcare}],} \\
& \text{vals([\text{'UP'},\text{int},<,\text{dontcare},\text{dontcare}],} \\
& \text{vals([\text{'VAR'},\text{int}('LOW' in 'UP'),=\text{,dontcare},\text{dontcare},} \\
& \text{translate([\text{'VAR','LOW','UP'].}])}).} \\
\text{ctr\_derived\_collections} & \text{(in\_interval,} \\
& \text{col('VARIABLE'-collection(var\_dvar),[item(var-'VAR'].}])}, \\
& \text{col('INTERVAL'-collection(low\_int,up\_int),} \\
& \text{[item(low-'LOW',up-'UP].}])}).} \\
\text{ctr\_graph} & \text{(in\_interval,} \\
& \text{['VARIABLE','INTERVAL'],} \\
& 2,} \\
& \text{['PRODUCT'>>collection(variable,interval)]}, \\
& \text{[variable\_var>=interval\_low,variable\_var=<interval\_up],} \\
& \text{['NARC'=1],} \\
& []).} \\
\text{ctr\_eval} & \text{(in\_interval,} \\
& \text{[reformulation(in\_interval\_r),automaton(in\_interval\_a)].} \\
\text{in\_interval\_r} & \text{(VAR,LOW,UP) :-} \\
& \text{check\_type(fdvar,VAR),}
\end{align*}
\]

\[
\begin{align*}
& \text{in\_interval\_r(VAR,LOW,UP) :-} \\
& \text{check\_type(fdvar,VAR),}
\end{align*}
\]
check_type(int,LOW),
check_type(int,UP),
LOW=<UP,
VAR#>=LOW,
VAR#=<UP.

\textbf{in\_interval\_a}(\textbf{FLAG},\textbf{VAR},\textbf{LOW},\textbf{UP}) :-
check_type(fdvar,\textbf{VAR}),
check_type(int,\textbf{LOW}),
check_type(int,\textbf{UP}),
LOW=<UP,
VAR#>=LOW#/\VAR#=<UP#<=>S,
AUTOMATON=
automaton(
[S],
_33636,
[S],
[source(s),sink(t)],
[arc(s,1,t)],
[],
[]),
automaton\_bool(\textbf{FLAG},[0,1],\textbf{AUTOMATON}).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.164 in_interval_reified

◊ Meta-Data:

ctr_predefined(in_interval_reified).

ctr_date(in_interval_reified, ['20100916']).

ctr_origin(
    in_interval_reified,
    Reified version of %c.,
    [in_interval]).

ctr_synonyms(in_interval_reified, [dom_reified, in_reified]).

ctr_arguments(
    in_interval_reified,
    ['VAR'-dvar,'LOW'-int,'UP'-int,'B'-dvar]).

ctr_restrictions(
    in_interval_reified,
    ['LOW'=<'UP','B'>=0,'B'=<1]).

ctr_example(in_interval_reified, in_interval_reified(3,2,5,1)).

ctr_typical(
    in_interval_reified,
    ['VAR'=\='LOW','VAR'=\='UP','LOW'<'UP']).

ctr_exchangeable(
    in_interval_reified,
    [vals(['VAR'],comp('LOW' in 'UP'),=,dontcare,dontcare),
     translate(['VAR','LOW','UP'])].

ctr_eval(
    in_interval_reified,
    [reformulation(in_interval_reified_r)]).

in_interval_reified_r(VAR,LOW,UP,B) :-
    check_type(dvar,VAR),
    check_type(int,LOW),
    check_type(int,UP),
    check_type(dvar(0,1),B),
    LOW=<UP,
    VAR in LOW..UP#=>B.
B.165 in_intervals

◊ META-DATA:

ctr_predefined(in_intervals).

ctr_date(in_intervals, ’20080610’).

ctr_origin(in_intervals, ’Domain definition.’, []).

ctr_synonyms(in_intervals, [in]).

ctr_arguments( in_intervals, ['VAR'-dvar, 'INTERVALS'-collection(low-int, up-int)]).

ctr_restrictions( in_intervals, [required('INTERVALS', [low, up]),
’INTERVALS’-low=<’INTERVALS’-up, size('INTERVALS')>0]).

ctr_example( in_intervals, in_intervals(5, [[low-1, up-1], [low-3, up-5], [low-8, up-8]])).

ctr_typical(in_intervals, [size('INTERVALS')>1]).

ctr_exchangeable( in_intervals, [items('INTERVALS', all),
vals(['INTERVALS'-'low', int, >, dontcare, dontcare),
vals(['INTERVALS'-'up', int, <, dontcare, dontcare),
translate(['VAR', 'INTERVALS'-'low', 'INTERVALS'-'up'])].

ctr_eval(in_intervals, [reformulation(in_intervals_r)]).

ctr_extensible(in_intervals, [], 'INTERVALS', any).

in_intervals_r(VAR, INTERVALS) :-
    check_type(dvar, VAR),
collection(INTERVALS, [int, int]),
length(INTERVALS, L),
L>0,
get_attr1(INTERVALS, LOWS),
get_attr2(INTERVALS, UPS),
check_leq(LOWS, UPS),
in_intervals1(LOWS, UPS, VAR, TERM),
call(TERM).

in_intervals1([], [], _16549, 0).

in_intervals1([LOW | RLOW], [UP | RUP], VAR, VAR# >= LOW# / \VAR# <= UP# / R) :- in_intervals1(RLOW, RUP, VAR, R).
B.166  in_relation

◊ **META-DATA:**

\[
\text{ctr\_date(in\_relation,[}’20030820’,’20040530’,’20060810’)\text{).}
\]

\[
\text{ctr\_origin(}
\text{in\_relation,}
\text{Constraint explicitly defined by tuples of values.,}
\text{[]).}
\]

\[
\text{ctr\_synonyms(}
\text{in\_relation,}
\text{[case,}
\text{extension,}
\text{extensional,}
\text{extensional\_support,}
\text{extensional\_supportva,}
\text{extensional\_supportmdd,}
\text{extensional\_supportstr,}
\text{feastupleac,}
\text{table]).}
\]

\[
\text{ctr\_types(}
\text{in\_relation,}
\text{[’TUPLE\_OF\_VARS’-collection(var-dvar),}
\text{’TUPLE\_OF\_VALS’-collection(val-int)]).}
\]

\[
\text{ctr\_arguments(}
\text{in\_relation,}
\text{[’VARIABLES’-’TUPLE\_OF\_VARS’,}
\text{’TUPLES\_OF\_VALS’-collection(tuple-’TUPLE\_OF\_VALS’)]).}
\]

\[
\text{ctr\_restrictions(}
\text{in\_relation,}
\text{[required(’TUPLE\_OF\_VARS’,var),}
\text{size(’TUPLE\_OF\_VARS’)>=1,}
\text{size(’TUPLE\_OF\_VALS’)>=1,}
\text{size(’TUPLE\_OF\_VALS’)=size(’VARIABLES’),}
\text{required(’TUPLE\_OF\_VALS’,val),}
\text{required(’TUPLES\_OF\_VALS’,tuple)]).}
\]

\[
\text{ctr\_example(}
\text{in\_relation,}
\text{in\_relation(}
\text{[[var-5],[var-3],[var-3]],}
\text{}}
\]
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

[[tuple-[[val-5],[val-2],[val-3]]],
[tuple-[[val-5],[val-2],[val-6]]],
[tuple-[[val-5],[val-3],[val-3]]]]).

ctr_typical(in_relation,[size(‘TUPLE_OF_VARS’)>1]).

ctr_exchangeable(
  in_relation,
  [items(‘TUPLES_OF_VALS’,all),
   items_sync(‘VARIABLES’,”TUPLES_OF_VALS”-tuple,all),
   vals(‘VARIABLES’,”TUPLES_OF_VALS”-tuple),
   int,
   =\=,
   all,
   dontcare))).

ctr_derived_collections(
  in_relation,
  [col(‘TUPLES_OF_VARS’-collection(vec-‘TUPLE_OF_VARS’),
    [item(vec-‘VARIABLES’)]))).

ctr_graph(
  in_relation,
  [‘TUPLES_OF_VARS’,”TUPLES_OF_VARS’],
  2,
  [’PRODUCT’>>collection(tuples_of_vars,tuples_of_vals)],
  [vec_eq_tuple(tuples_of_vars-vec,tuples_of_vals-tuple)],
  [’NARC’>=1],
  []).

ctr_eval(in_relation,[reformulation(in_relation_r)]).

ctr_extensible(in_relation,[],’TUPLES_OF_VALS’,any).

in_relation_r(VARIABLES,TUPLES_OF_VALS) :-
collection(VARIABLES,[dvar]),
length(VARIABLES,N),
collection(TUPLES_OF_VALS,[col(N,[int])]),
get_attr1(VARIABLES,VARS),
get_col_attr1(TUPLES_OF_VALS,1,TUPLES),
table([VARS],TUPLES).
B.167 in_same_partition

◊ META-DATA:

ctr_date(in_same_partition,['20030820','20040530','20060810']).

ctr_origin(  
in_same_partition,  
Used for defining several entries of this catalog.,  
[]).

ctr_types(in_same_partition,['VALUES'-collection(val-int)]).

ctr_arguments(  
in_same_partition,  
['VAR1'-dvar,  
'VAR2'-dvar,  
'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(  
in_same_partition,  
[size('VALUES')>=1,  
required('VALUES',val),  
distinct('VALUES',val),  
required('PARTITIONS',p),  
size('PARTITIONS')>=2]).

ctr_example(  
in_same_partition,  
in_same_partition(  
6,  
2,  
[[p-[[val-1],[val-3]]],  
[p-[[val-4]]],  
[p-[[val-2],[val-6]]]]).

ctr_typical(in_same_partition,['VAR1'='VAR2']).

ctr_exchangeable(  
in_same_partition,  
[<args([['VAR1','VAR2'],['PARTITIONS']]),  
items('PARTITIONS',all),  
items('PARTITIONS'\p,all)]).
\[\text{col('VARIABLES'\text{-}collection(var\text{-}dvar),}
\text{[item(var\text{-}‘VAR1’),item(var\text{-}‘VAR2’)])]}\].

\(\text{ctr\_graph(}
\text{in\_same\_partition,}
\text{[‘VARIABLES’,'PARTITIONS’],}
\text{2,}
\text{[‘PRODUCT’\text{\textgreater\textgreater}collection:variables:partitions],}
\text{[variables\text{\_}var in partitions\_p],}
\text{[‘NSOURCE’=2,’NSINK’=1],}
\text{[]].}\)

\(\text{ctr\_eval(}
\text{in\_same\_partition,}
\text{[reformulation(in\_same\_partition\_r),}
\text{automaton(in\_same\_partition\_a)]}.}\)

\(\text{ctr\_extensible(in\_same\_partition,[],’PARTITIONS’,\text{any}).}\)

\(\text{in\_same\_partition\_r(VAR1,VAR2,PARTITIONS) :-}
\text{check\_type(dvar,VAR1),}
\text{check\_type(dvar,VAR2),}
\text{collection(PARTITIONS,[col\_len\_gteq(1,[\text{int}]])],}
\text{length(PARTITIONS,P),}
\text{P>1,}
\text{collection\_distinct(PARTITIONS,1),}
\text{get\_col\_attr1(PARTITIONS,1,PVALS),}
\text{in\_same\_partition1(PVALS,VAR1,VAR2,TERM),}
\text{call(TERM).}\)

\(\text{in\_same\_partition1([],_35297,_35298,0).}\)

\(\text{in\_same\_partition1([VALS[R]},VAR1,VAR2,TERM1#/\text{\textbackslash}TERM2#/\text{\textbackslash}\text{TERM}) :-}
\text{build\_or\_var\_in\_values(VALS,VAR1,TERM1),}
\text{build\_or\_var\_in\_values(VALS,VAR2,TERM2),}
\text{in\_same\_partition1(R,VAR1,VAR2,TERM).}\)

\(\text{in\_same\_partition\_a(FLAG,VAR1,VAR2,PARTITIONS) :-}
\text{check\_type(dvar,VAR1),}
\text{check\_type(dvar,VAR2),}
\text{collection(PARTITIONS,[col\_len\_gteq(1,[\text{int}]])],}
\text{length(PARTITIONS,P),}
\text{P>1,}
\text{collection\_distinct(PARTITIONS,1),}
\text{in\_same\_partition\_signature(}
\text{PARTITIONS,}\)
SIGNATURE, 
VAR1, 
VAR2), 
AUTOMATON= 
automaton( 
  SIGNATURE, 
  _37686, 
  SIGNATURE, 
  [source(s),sink(t)], 
  [arc(s,0,s),arc(s,1,t),arc(t,0,t),arc(t,1,t)], 
  [], 
  [], 
  []), 
automaton_bool(FLAG,[0,1],AUTOMATON).

in_same_partition_signature([],[],_35298,_35299).

in_same_partition_signature( 
  [[p-VALUES]|PARTITIONs], 
  [S|Ss], 
  VAR1, 
  VAR2) :- 
  get_attr1(VALUES,LIST_VALUES), 
  list_to_fdset(LIST_VALUES,SET_OF_VALUES), 
  VAR1 in_set SET_OF_VALUES\/
  VAR2 in_set SET_OF_VALUES\<=>
  S, 
in_same_partition_signature(PARTITIONs,Ss,VAR1,VAR2).
B.168  in_set

◊ **Meta-Data:**

```prolog
ctr_predefined(in_set).
ctr_date(in_set,['20030820']).
ctr_origin(
    in_set,
    Used for defining constraints with set variables.,
    []).
ctr_synonyms(in_set,[dom,member]).
ctr_arguments(in_set,['VAL'-dvar,'SET'-svar]).
ctr_example(in_set,3 in_set{1,3}).
```
B.169 incomparable

◊ **META-DATA:**

ctr_predefined(incomparable).

ctr_date(incomparable, ['20120202']).

ctr_origin(
    incomparable,
    Inspired by incomparable rectangles.,
    []).

ctr_synonyms(incomparable, [incomparables]).

ctr_arguments(
    incomparable,
    ['VECTOR1'-collection(var-dvar),
    'VECTOR2'-collection(var-dvar)]).

ctr_restrictions(
    incomparable,
    [required('VECTOR1', var),
    required('VECTOR2', var),
    size('VECTOR1')>=1,
    size('VECTOR2')>=1,
    size('VECTOR1')=size('VECTOR2')]).

ctr_example(
    incomparable,
    [incomparable([[var-16],[var-2]],[[var-4],[var-11]])]).

ctr_typical(incomparable, [size('VECTOR1')>1]).

ctr_eval(incomparable, [reformulation(incomparable_r)]).

incomparable_r(VECTOR1,VECTOR2) :-
    collection(VECTOR1,[dvar]),
    collection(VECTOR2,[dvar]),
    length(VECTOR1,L),
    length(VECTOR2,L),
    get_attr1(VECTOR1,VECT1),
    get_attr1(VECTOR2,VECT2),
    incomparable(VECT1,VECT2).

incomparable([[U1,U2],[V1,V2]] :-
incomparable(U,V) :-
  length(U,N),
  length(V,N),
  N>1,
  length(PU,N),
  length(PV,N),
  domain(PU,1,N),
  domain(PV,1,N),
  get_minimum(U,MinU),
  get_maximum(U,MaxU),
  get_minimum(V,MinV),
  get_maximum(V,MaxV),
  length(SU,N),
  length(SV,N),
  domain(SU,MinU,MaxU),
  domain(SV,MinV,MaxV),
  sorting(U,PU,SU),
  sorting(V,PV,SV),
  incomparable(SU,SV,Or1),
  call(Or1),
  incomparable(SV,SU,Or2),
  call(Or2).

incomparable([],[],0).

incomparable([U|R],[V|S],U>V\$/T) :-
incomparable(R,S,T).
B.170 increasing

◊ **META-DATA:**

ctr_date(increasing, ['20040814', '20060810', '20091105']).

ctr_origin(increasing, 'KOALOG', []).

ctr_arguments(increasing, ['VARIABLES' - collection(var-dvar)]).

ctr_restrictions(increasing, [required('VARIABLES', var)]).

ctr_example(increasing, increasing([[var-1], [var-1], [var-4], [var-8]])).

ctr_typical(increasing, [size('VARIABLES') > 1, range('VARIABLES' ^ var) > 1]).

ctr_exchangeable(increasing, translate(['VARIABLES' ^ var])).

ctr_graph(increasing, ['VARIABLES'], 2, ['PATH' >> collection(variables1, variables2)], [variables1 ^ var =< variables2 ^ var], ['NARC' = size('VARIABLES') - 1], []).

ctr_eval(increasing, [checker(increasing_c), reformulation(increasing_r), automaton(increasing_a)]).

ctr_contractible(increasing, [], 'VARIABLES', any).

increasing_c([], 'VARIABLES', any).

increasing_c(VARIABLES) :- collection(VARIABLES, [int]), get_attr1(VARIABLES, VARS), increasing_c1(VARS).
increasing_c1([]) :- !.

increasing_c1([_30966]) :- !.

increasing_c1([X,Y|R]) :-
    X=<Y,
    increasing_c1([Y|R]).

increasing_r([]) :- !.

increasing_r(VARIABLES) :-
    collection(VARIABLES,[dvar]),
    get_attr1(VARIABLES,VARS),
    increasing1(VARS).

increasing1([_30966]) :- !.

increasing1([V1,V2|R]) :-
    V1#=<V2,
    increasing1([V2|R]).

increasing_a(1,[]) :- !.

increasing_a(0,[]) :- !, fail.

increasing_a(FLAG,VARIABLES) :-
    collection(VARIABLES,[dvar]),
    increasing_signature(VARIABLES,SIGNATURE),
    AUTOMATON=automaton(
        SIGNATURE,
        _32097,
        SIGNATURE,
        [source(s),sink(s)],
        [arc(s,1,s)],
        [],
        [],
        [],
        []).
automaton_bool(FLAG,[0,1],AUTOMATON).

increasing_signature([_30967],[[]) :- !.

increasing_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss]) :-
  S in 0..1,
  VAR1#=<VAR2#<=S,
  increasing_signature([[var-VAR2]|VARs],Ss).
B.171 increasing_global_cardinality

◊ Meta-Data:

ctr_date(increasing_global_cardinality, ['20091015']).

ctr_origin(
    increasing_global_cardinality,
    Conjoin %c and %c.,
    [global_cardinality_low_up, increasing]).

ctr_synonyms(
    increasing_global_cardinality,
    [increasing_global_cardinality_low_up,
     increasing_gcc,
     increasing_gcc_low_up]).

ctr_arguments(
    increasing_global_cardinality,
    ['VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int, omin-int, omax-int)]).

ctr_restrictions(
    increasing_global_cardinality,
    [required('VARIABLES', var),
     increasing('VARIABLES'),
     size('VALUES') > 0,
     required('VALUES', [val, omin, omax]),
     distinct('VALUES', val),
     'VALUES'^omin >= 0,
     'VALUES'^omax <= size('VARIABLES'),
     'VALUES'^omin <= 'VALUES'^omax]).

ctr_example(
    increasing_global_cardinality,
    increasing_global_cardinality(
        [[var-3],[var-3],[var-6],[var-8]],
        [[val-3,omin-2,omax-3],
         [val-5,omin-0,omax-1],
         [val-6,omin-1,omax-2]]).

ctr_typical(
    increasing_global_cardinality,
    [size('VARIABLES') > 1,
     range('VARIABLES'^var) > 1,
     size('VALUES') > 1,
'VALUES'\_omin=size('VARIABLES'),
'VALUES'\_omax>0,
'VALUES'\_omax=size('VARIABLES'),
size('VARIABLES')>size('VALUES')).

\text{ctr\_exchangeable}(
  increasing\_global\_cardinality,
  [items('VALUES',all)]).

\text{ctr\_graph}(
  increasing\_global\_cardinality,
  ['VARIABLES'],
  1,
  foreach('VALUES',['SELF'>>collection(variables)]),
  [variables\_var='VALUES'\_val],
  ['NVERTEX'='VALUES'\_omin,'NVERTEX'='VALUES'\_omax],
  []).

\text{ctr\_eval}(
  increasing\_global\_cardinality,
  [reformulation(increasing\_global\_cardinality\_r),
   automaton(increasing\_global\_cardinality\_a)]).

\text{ctr\_functional\_dependency}(increasing\_nvalue,1,[2]).

\text{increasing\_global\_cardinality\_r}(VARIABLES,VALUES) :-
  eval(increasing(VARIABLES)),
  eval(global\_cardinality\_low\_up(VARIABLES,VALUES)).

\text{increasing\_global\_cardinality\_a}(FLAG,VARIABLES,VALUES) :-
  increasing\_global\_cardinality\_get\_a(
    VARIABLES,
    VALUES,
    AUTOMATON,
    ALPHABET),
  automaton\_bool(FLAG,ALPHABET,AUTOMATON).

\text{increasing\_global\_cardinality\_get\_a}(
  VARIABLES,
  VALUES,
  AUTOMATON,
  ALPHABET) :-
  length(VARIABLES,N),
  collection(VARIABLES,[dvar]),
  collection(VALUES,[int,int(0,N),int(0,N)]),
  length(VALUES,M),
\[ M > 0, \]
\[
\text{sort}\_\text{collection}(\text{VALUES}, \text{val}, \text{SVALUES}),
\]
\[
\text{get}\_\text{attr1}(\text{VARIABLES}, \text{VARS}),
\]
\[
\text{get}\_\text{attr1}(\text{SVALUES}, \text{VALS}),
\]
\[
\text{get}\_\text{attr2}(\text{SVALUES}, \text{OMINS}),
\]
\[
\text{get}\_\text{attr3}(\text{SVALUES}, \text{OMAXS}),
\]
\[
\text{all}\_\text{different}(\text{VALS}),
\]
\[
\text{check}\_\text{lesseq}(\text{OMINS}, \text{OMAXS}),
\]
\[
\text{get}\_\text{minimum}(\text{VARS}, \text{MINVARS}),
\]
\[
\text{get}\_\text{maximum}(\text{VARS}, \text{MAXVARS}),
\]
\[
\text{get}\_\text{minimum}(\text{VALS}, \text{MINVALS}),
\]
\[
\text{get}\_\text{maximum}(\text{VALS}, \text{MAXVALS}),
\]
\[
\text{MIN is min}(\text{MINVARS}, \text{MINVALS}),
\]
\[
\text{MAX is max}(\text{MAXVARS}, \text{MAXVALS}),
\]
\[
\text{get}\_\text{sum}(\text{OMINS}, \text{SUM}\_\text{OMINS}),
\]
\[
\text{REST is N-SUM}\_\text{OMINS},
\]
\[
\text{REST} \geq 0,
\]
\[
\text{increasing}\_\text{global}\_\text{cardinality}\_\text{complete}\_\text{values}(\text{MIN}, \text{MAX}, \text{SVALUES}, \text{REST}, \text{CVVALUES}, \text{SUM}\_\text{OMAXS}),
\]
\[
\text{reverse}(\text{CVVALUES}, \text{RVALUES}),
\]
\[
\text{increasing}\_\text{global}\_\text{cardinality}\_\text{term}\_\text{states}(\text{RVALUES}, \text{SUM}\_\text{OMAXS}, \text{TERMINALS}),
\]
\[
\text{append}([\text{source}(0)], \text{TERMINALS}, \text{STATES}),
\]
\[
\text{increasing}\_\text{global}\_\text{cardinality}\_\text{source}\_\text{trans}(\text{CVVALUES}, 1, \text{TRANSITIONS}\_\text{FROM}\_\text{SOURCE}),
\]
\[
\text{increasing}\_\text{global}\_\text{cardinality}\_\text{horiz}\_\text{trans}(\text{CVVALUES}, 1, \text{TRANSITIONS}\_\text{HORIZONTAL}),
\]
\[
\text{increasing}\_\text{global}\_\text{cardinality}\_\text{vert}\_\text{trans}(\text{CVVALUES}, 1, \text{TRANSITIONS}\_\text{VERTICAL}),
\]
\[
\text{append}([-\text{TRANSITIONS}\_\text{FROM}\_\text{SOURCE}, \text{TRANSITIONS}\_\text{HORIZONTAL}, \text{T1})]}
append(T1,TRANSITIONS_VERTICAL,ALL_TRANSITIONS),
AUTOMATON=
automaton(
    VARS,
    _45074,
    VARS,
    STATES,
    ALL_TRANSITIONS,
    [],
    [],
    []),
append(VARS,VALS,ALL),
union_dom_list_int(ALL,ALPHABET).

increasing_global_cardinality_complete_values(
    MIN,
    MAX,
    VALUES,
    _REST,
    VALUES, 
    0) :-
    MIN>MAX,
    ( VALUES=[] ->
        true
        ; write(problem),
        nl,
        abort
    ),
    !.

increasing_global_cardinality_complete_values(
    MIN,
    MAX,
    [],
    REST,
    [[constrained-CTR,val-MIN,omin-0,omax-OOMAX]|S],
    SUM) :-
    MIN=<MAX,
    !,
    ( REST>1 ->
        CTR=0,
        OOMAX=1
        ; CTR=1,
        OOMAX is max(1,REST)
    ),
    MIN1 is MIN+1,
increasing_global_cardinality_complete_values(
  MIN1, 
  MAX, 
  [], 
  REST, 
  S, 
  TSUM), 
SUM is TSUM+OOMAX.

increasing_global_cardinality_complete_values(
  MIN, 
  MAX, 
  [[val-VAL,omin-OMIN,omax-OOMAX]|R], 
  REST, 
  [[constrained-CTR,val-VAL,omin-OMIN,omax-OOMAX]|S], 
  SUM) :-
  MIN=<MAX, 
  MIN=VAL, 
  !,
  ( OMAX>1, 
  OMAX>=REST+OMIN -> 
  CTR=0, 
  OOMAX is max(1,OMIN) ; 
  CTR=1, 
  OOMAX is max(1,OMAX) ),
  MIN1 is MIN+1, 
  increasing_global_cardinality_complete_values(
    MIN1, 
    MAX, 
    R, 
    REST, 
    S, 
    TSUM), 
  SUM is TSUM+OOMAX.

increasing_global_cardinality_complete_values(
  MIN, 
  MAX, 
  [[val-VAL,omin-OMIN,omax-OOMAX]|R], 
  REST, 
  [[constrained-CTR,val-MIN,omin-0,omax-OOMAX]|S], 
  SUM) :-
  MIN=<MAX, 
  MIN<VAL, 
  ( REST>1 ->
increasing_global_cardinality_term_states([],_37153,[]).

increasing_global_cardinality_term_states([\[[\text{constrained}_{-37164}\text{,val}_{-VAL},\text{omin}_{-OMIN},\text{omax}_{-OMAX}]\mid R\],\text{LAST\_STATE\_ID},\text{RES}\} :-
  I is \text{LAST\_STATE\_ID\_OMAX}+\text{max}(1,\text{OMIN}),
  increasing_global_cardinality_term_states1(I,\text{LAST\_STATE\_ID},\text{TERMS}),
  \text{LAST\_STATE\_ID1} is \text{LAST\_STATE\_ID\_OMAX},
  (\text{OMIN}=0 ->
   increasing_global_cardinality_term_states([R,\text{LAST\_STATE\_ID1},\text{S}]),
   append(\text{S},\text{TERMS},\text{RES})
  ; \text{RES}=\text{TERMS}
  ).

increasing_global_cardinality_term_states1(I,MAX,[]) :-
  I>MAX,
  !.

increasing_global_cardinality_term_states1(I,MAX,[\text{sink}(I)\mid R]) :-
  I<=MAX,
  I1 is I+1,
  increasing_global_cardinality_term_states1(I1,MAX,R).

increasing_global_cardinality_source_trans([],_37153,[]).
increasing_global_cardinality_source_trans(
    [[\text{constrained}--\text{CTR}, \text{val}--\text{VAL}, \text{omin}--\text{OMIN}, \text{omax}--\text{OMAX}]\mid R],
    \text{CUR}_{\text{ID}},
    [\text{arc}(0,\text{VAL}, \text{CUR}_{\text{ID}})\mid S]) :-
    \text{CUR}_{\text{ID}1} \text{ is } \text{CUR}_{\text{ID}}+\text{OMAX},
    \begin{cases}
        \text{OMIN}=0 \rightarrow
        \text{increasing_global_cardinality_source_trans}(R, \text{CUR}_{\text{ID}1}, S) \\
        \text{S}=[]
    \end{cases}
).

increasing_global_cardinality_horiz_trans([],_37153,[]).

increasing_global_cardinality_horiz_trans(
    [[\text{constrained}--\text{CTR}, \text{val}--\text{VAL}, \text{omin}--\text{OMIN}, \text{omax}--\text{OMAX}]\mid R],
    \text{CUR}_{\text{ID}},
    \text{RESULT}) :-
    increasing_global_cardinality_horiz_trans1(I, \text{OMAX}, \text{CTR}, \text{VAL}, \text{CUR}_{\text{ID}}, \text{TR}),
    \text{CUR}_{\text{ID}1} \text{ is } \text{CUR}_{\text{ID}}+\text{OMAX},
    \text{increasing_global_cardinality_horiz_trans}(R, \text{CUR}_{\text{ID}1}, \text{S}),
    \text{append}(\text{TR}, \text{S}, \text{RESULT}).

increasing_global_cardinality_horiz_trans1(I, \text{OMAX}, 1, _37540, _37586, []):-
    I>=\text{OMAX},
    !.

increasing_global_cardinality_horiz_trans1(I, \text{OMAX}, 0, \text{VAL},
ID, [arc(ID,VAL,ID))] :- I>=OMAX, !.

increasing_global_cardinality_horiz_trans1(I, OMAX, CTR, VAL, ID, [arc(ID,VAL,ID1)|R]) :- I<OMAX, ID1 is ID+1, I1 is I+1, increasing_global_cardinality_horiz_trans1(I1, OMAX, CTR, VAL, ID1, R).

increasing_global_cardinality_vert_trans([_37158],_37156,[]) :- !.

increasing_global_cardinality_vert_trans([_[constrained-_37164,val-_VAL,omin-OMIN,omax-OMAX]|R], CUR_ID, RESULT) :- I is CUR_ID+max(0,OMIN-1), CUR_ID1 is CUR_ID+OMAX, increasing_global_cardinality_vert_trans1(R, CUR_ID1, I, CUR_ID1, S), increasing_global_cardinality_vert_trans(R,CUR_ID1,T), append(S,T,RESULT).

increasing_global_cardinality_vert_trans1([], _37428, _37474, _37520,
increasing_global_cardinality_vert_trans1( 
   [[constrained-_37166,val-VAL,omin-OMIN,omax-OMAX]|R],
   CUR_ID, I, MAX, RESULT) :-
   increasing_global_cardinality_vert_trans2( 
       I, MAX, CUR_ID, VAL, RES1),
   CUR_ID1 is CUR_ID+OMAX,
   ( OMIN=0 ->
     increasing_global_cardinality_vert_trans1( 
       R, CUR_ID1, I, MAX, RES2),
     append(RES1,RES2,RESULT) ; RESULT=RES1
   ).

increasing_global_cardinality_vert_trans2( 
   MAX, MAX, _37485, _37531, [[]] :-
   !.

increasing_global_cardinality_vert_trans2( 
   I, MAX, CUR_ID, VAL, [arc(I,VAL,CUR_ID)|R]) :-
   I<MAX, II is I+1,
   increasing_global_cardinality_vert_trans2( II, MAX, CUR_ID, 

VAL,
R) .
B.172  increasing_nvalue

◊ Meta-Data:

ctr_date(increasing_nvalue, ['20091104']).

ctr_origin(
    increasing_nvalue,
    Conjoin %c and %c.,
    [nvalue,increasing]).

ctr_arguments(
    increasing_nvalue,
    ['NVAL'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    increasing_nvalue,
    ['NVAL'>=min(1,size('VARIABLES')),
     'NVAL'=<size('VARIABLES'),
     required('VARIABLES',var),
     increasing('VARIABLES')]).

ctr_example(
    increasing_nvalue,
    increasing_nvalue(2,
    [[var-6],[var-6],[var-8],[var-8],[var-8]])).

ctr_typical(
    increasing_nvalue,
    [size('VARIABLES')>1,range('VARIABLES'\^var)>1]).

ctr_exchangeable(
    increasing_nvalue,
    [translate(['VARIABLES'\^var])].

ctr_graph(
    increasing_nvalue,
    ['VARIABLES'],
    2,
    ['CLIQUE'\>>collection(variables1,variables2)],
    [variables1\^var=variables2\^var],
    ['NSCC'='NVAL'],
    ['EQUIVALENCE'])..

ctr_eval(
increasing_nvalue,  
[reformulation(increasing_nvalue_r),  
automata(increasing_nvalue_a)).

increasing_nvalue_r(0,[]) :-  
!.

increasing_nvalue_r(NVAL,VARIABLES) :-  
eval(increasing(VARIABLES)),  
eval(nvalue(NVAL,VARIABLES)).

increasing_nvalue_a(0,[]) :-  
!.

increasing_nvalue_a(NVAL,VARIABLES) :-  
check_type(dvar,NVAL),  
collection(VARIABLES,[dvar]),  
get_attr1(VARIABLES,VARS),  
length(VARIABLES,N),  
NVAL#>=min(1,N),  
NVAL#=N,  
get_minimum(VARS,MINVARS),  
get_maximum(VARS,MAXVARS),  
SIZE is MAXVARS-MINVARS+1,  
fd_min(NVAL,MINNVAL),  
fd_max(NVAL,MAXNVAL),  
D is min(N,min(SIZE,MAXNVAL)),  
fd_set(NVAL,SVAL),  
fdset_to_list(SVAL,VALUES),  
increasing_nvalue_states(VALUES,SIZE,MINNVAL,STATES),  
gen_automaton_state(s,0,0,S_00),  
increasing_nvalue_class1(  
1,  
SIZE,  
MINNVAL,  
MINVARS,  
S_00,  
TRANS1),  
increasing_nvalue_class2(  
1,  
D,  
SIZE,  
MINNVAL,  
MINVARS,  
TRANS2),  
increasing_nvalue_class3(
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

1,
D,
SIZE,
MINNVAL,
MINVARS,
TRANS3),
append(TRANS1,TRANS2,TRANS12),
append(TRANS12,TRANS3,ALL_TRANSITIONS),
automaton(
  VARS,
  _45260,
  VARS,
  STATES,
  ALL_TRANSITIONS,
  [],
  [],
  []),
eval(nvalue(NVAL,VARIABLES)).

increasing_nvalue_states([],_39743,_39744,[source(S_00)]) :-
gen_automaton_state(s,0,0,S_00).

increasing_nvalue_states([V|R],SIZE,MINNVAL,STATES) :-
  increasing_nvalue_states1(V,SIZE,V,MINNVAL,STATES1),
  increasing_nvalue_states(R,SIZE,MINNVAL,STATES2),
  append(STATES1,STATES2,STATES).

increasing_nvalue_states1(J,SIZE,_39744,_39745,[]) :-
  J>SIZE,
  !.

increasing_nvalue_states1(J,SIZE,I,MINNVAL,[sink(S_IJ)|STATES]) :-
  J=<SIZE,
  I_SIZE_J is I+SIZE-J,
  I_SIZE_J>=MINNVAL,
  !,
  gen_automaton_state(s,I,J,S_IJ),
  J1 is J+1,
  increasing_nvalue_states1(J1,SIZE,I,MINNVAL,STATES).

increasing_nvalue_states1(J,SIZE,I,MINNVAL,STATES) :-
  J=<SIZE,
  J1 is J+1,
  increasing_nvalue_states1(J1,SIZE,I,MINNVAL,STATES).

increasing_nvalue_class1(J,SIZE,_39744,_39745,_39746,[]) :-
increasing_nvalue_class1(J, SIZE, MINNVAL, MINVARS, S_00, [arc(S_00, LABEL, S_1J) | TRANS]) :-
    J=<SIZE,
    I_SIZE_J is 1+SIZE-J,
    I_SIZE_J>=MINNVAL,
    !,
    gen_automatoni_state(s,1,J,S_1J),
    LABEL is MINVARS+J-1,
    J1 is J+1,
    increasing_nvalue_class1(J1, SIZE, MINNVAL, MINVARS, S_00, TRANS).

declare_clause(increasing_nvalue_class1(J,SIZE,MINNVAL,MINVARS,S_00,TRANS)) :-
    J=<SIZE,
    J1 is J+1,
    increasing_nvalue_class1(J1,SIZE,MINNVAL,MINVARS,S_00,TRANS).

increasing_nvalue_class2(I,D,_SIZE,_MINNVAL,_MINVARS,[]) :-
    I>D,
    !.

declare_clause(increasing_nvalue_class2(I,D,SIZE,MINNVAL,MINVARS,TRANS)) :-
    I=<D,
    increasing_nvalue_class2(I,SIZE,MINNVAL,MINVARS,TRANS).
increasing_nvalue_class2(J, SIZE, I, MINNVAL, MINVARS, TRANS) :-
    J < SIZE,
    I1 is I + SIZE - J,
    I1 >= MINNVAL,
    gen_automaton_state(s, I, J, S_IJ),
    LABEL is MINVARS + J - 1,
    J1 is J + 1,
    increasing_nvalue_class2(J1, SIZE, I, MINNVAL, MINVARS, TRANS).

increasing_nvalue_class2(J, SIZE, I, MINNVAL, MINVARS, TRANS) :-
    J < SIZE,
    J1 is J + 1,
    increasing_nvalue_class2(J1, SIZE, I, MINNVAL, MINVARS, TRANS).
increasing\_nvalue\_class3(I,D,MINNVAL,MINVARS,TRANS).

\[
\text{increasing\_nvalue\_class3(I,D,\_39744,\_39745,\_39746,[])} :\text{I>=D,} \\
\text{!}.
\]

\[
\text{increasing\_nvalue\_class3(I,D,SIZE,MINNVAL,MINVARS,TRANS)} :\text{I<D,} \\
\text{increasing\_nvalue\_class31(} \\
\text{I, SIZE,} \\
\text{I,} \\
\text{MINNVAL,} \\
\text{MINVARS,} \\
\text{TRANS1),} \\
\text{I1 is I+1,} \\
\text{increasing\_nvalue\_class3(} \\
\text{I1, D,} \\
\text{SIZE,} \\
\text{MINNVAL,} \\
\text{MINVARS,} \\
\text{TRANS2),} \\
\text{append(TRANS1,TRANS2,TRANS).}
\]

\[
\text{increasing\_nvalue\_class31(J,SIZE,\_39744,\_39745,\_39746,[])} :\text{J>SIZE,} \\
\text{!}.
\]

\[
\text{increasing\_nvalue\_class31(J,SIZE,I,MINNVAL,MINVARS,TRANS)} :\text{J=<SIZE,} \\
\text{I\_SIZE\_J is I\_SIZE\_J,} \\
\text{I\_SIZE\_J>=MINNVAL,} \\
\text{!,} \\
\text{gen\_automaton\_state(s,I,J,S\_IJ),} \\
\text{J1 is J+1,} \\
\text{increasing\_nvalue\_class32(} \\
\text{J1,} \\
\text{SIZE,} \\
\text{I,} \\
\text{J,} \\
\text{S\_IJ,} \\
\text{MINNVAL,} \\
\text{MINVARS,}
\]
TRANS1),
increasing_nvalue_class31(  
   J1,  
   SIZE,  
   I,  
   MINNVAL,  
   MINVARS,  
   TRANS2),  
append(TRANS1,TRANS2,TRANS).

increasing_nvalue_class31(J,SIZE,I,MINNVAL,MINVARS,TRANS) :-  
   J=<SIZE,  
   J1 is J+1,  
   increasing_nvalue_class31(  
      J1,  
      SIZE,  
      I,  
      MINNVAL,  
      MINVARS,  
      TRANS).

increasing_nvalue_class32(  
   K,  
   SIZE,  
   _40089,  
   _40135,  
   _40181,  
   _40227,  
   _40273,  
   []) :-  
   K>SIZE,  
!.

increasing_nvalue_class32(  
   K,  
   SIZE,  
   I,  
   J,  
   S_IJ,  
   MINNVAL,  
   MINVARS,  
   [arc(S_IJ,LABEL,S_I1K) | TRANS}) :-  
   K=<SIZE,  
   I1 is I+1,  
   I1_SIZE_K is I1+SIZE-K,  
   I1_SIZE_K>=MINNVAL,
!,
gen_automaton_state(s,I1,K,S_I1K),
LABEL is MINVARS+K-1,
K1 is K+1,
increasing_nvalue_class32(
  K1,
  SIZE,
  I,
  J,
  S_IJ,
  MINNVAL,
  MINVARS,
  TRANS).

increasing_nvalue_class32(
  K,
  SIZE,
  I,
  J,
  S_IJ,
  MINNVAL,
  MINVARS,
  TRANS) :-
  K=<SIZE,
  K1 is K+1,
increasing_nvalue_class32(
  K1,
  SIZE,
  I,
  J,
  S_IJ,
  MINNVAL,
  MINVARS,
  TRANS).
B.173  increasing_nvalue_chain

♦ Meta-Data:

ctr_date(increasing_nvalue_chain, [‘20091118’]).

ctr_origin(
    increasing_nvalue_chain,
    Derived from %c.,
    [increasing_nvalue]).

ctr_arguments(
    increasing_nvalue_chain,
    [‘NVAL’-dvar,’VARIABLES’-collection(b-dvar,var-dvar)]).

ctr_restrictions(
    increasing_nvalue_chain,
    [‘NVAL’>=min(1,size(‘VARIABLES’)),
     ‘NVAL’<=size(‘VARIABLES’),
     required(‘VARIABLES’,[b,var]),
     ‘VARIABLES’^b>=0,
     ‘VARIABLES’^b=<1]).

ctr_example(
    increasing_nvalue_chain,
    increasing_nvalue_chain(6,
        [[b-0,var-2],
        [b-1,var-4],
        [b-1,var-4],
        [b-1,var-4],
        [b-0,var-4],
        [b-1,var-8],
        [b-0,var-1],
        [b-0,var-7],
        [b-1,var-7]]).

ctr_typical(
    increasing_nvalue_chain,
    [size(‘VARIABLES’)>1,
     range(‘VARIABLES’^b)>1,
     range(‘VARIABLES’^var)>1]).

ctr_graph(
    increasing_nvalue_chain,
    [‘VARIABLES’],
2,
[‘PATH’>>collection(variables1,variables2)],
[variables2\b=0\variables1\var=variables2\var],
[‘NARC’=\size{‘VARIABLES’}-1],
[]).

ctr_graph(
  increasing_nvalue_chain,
  [‘VARIABLES’],
  2,
  [‘PATH’>>collection(variables1,variables2)],
  [variables2\b=0\variables1\var<variables2\var],
  [‘NARC’=\NVAL’-1],
  []).

ctr_eval(
  increasing_nvalue_chain,
  [reformulation(increasing_nvalue_chain_r)]).

increasing_nvalue_chain_r(_32696,_32697).
B.174  increasing_sum

◊ Meta-Data:

CTR_Predefined(increasing_sum).

CTR_Date(increasing_sum,['20110617']).

CTR_Origin(
  increasing_sum,
  Conjoin %c and %c.,
  [increasing,sum_ctr]).

CTR_Synonyms(
  increasing_sum,
  [increasing_sum_ctr,increasing_sum_eq]).

CTR_Arguments(
  increasing_sum,
  ['VARIABLES'-collection(var-dvar),'S'-dvar]).

CTR_Restrictions(
  increasing_sum,
  [required('VARIABLES',var),increasing('VARIABLES')]).

CTR_Example(
  increasing_sum,
  increasing_sum([[var-3],[var-3],[var-6],[var-8]],20)).

CTR_Typical(
  increasing_sum,
  [size('VARIABLES')>1,range('VARIABLES'-'var'>1)].

CTR_Eval(increasing_sum,[reformulation(increasing_sum_r)]).

increasing_sum_r(VARIABLES,S) :-
  eval(increasing(VARIABLES)),
  eval(sum_ctr(VARIABLES,=,S)).
B.175  indexed_sum

◊ Meta-Data:

ctr_date(indexed_sum,[‘20040814’,‘20060810’,‘20090422’]).

ctr_origin(indexed_sum,’N.˜Beldiceanu’,[{}]).

ctr_arguments(
    indexed_sum,
    [‘ITEMS’-collection(index-dvar,weight-dvar),
     ‘TABLE’-collection(index-int,summation-dvar)]).

ctr_restrictions(
    indexed_sum,
    [size(‘ITEMS’)>0,
     size(‘TABLE’)>0,
     required(‘ITEMS’,[index,weight]),
     ‘ITEMS’^index>=1,
     ‘ITEMS’^index=<size(‘TABLE’),
     required(‘TABLE’,[index,summation]),
     ‘TABLE’^index>=1,
     ‘TABLE’^index=<size(‘TABLE’),
     increasing_seq(‘TABLE’,index)]).

ctr_example(
    indexed_sum,
    indexed_sum(
        [[index-3,weight- -4],
         [index-1,weight-6],
         [index-3,weight-1]],
        [[index-1,summation-6],
         [index-2,summation-0],
         [index-3,summation- -3]]).

ctr_typical(
    indexed_sum,
    [size(‘ITEMS’)>1,
     range(‘ITEMS’^index)>1,
     size(‘TABLE’)>1,
     range(‘TABLE’^summation)>1]).

ctr_exchangeable(
    indexed_sum,
    [items(‘ITEMS’,all),items(‘TABLE’,all)]).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

ctr_graph(
    indexed_sum,
    ['ITEMS','TABLE'] ,
    2,
    foreach('TABLE',[PRODUCT>>collection(items,table)]),
    [items^index=table^index],
    [],
    [],
    [SUCC>>
      [source,
       variables-
       col('VARIABLES'-collection(var-dvar),
         [item(var-'ITEMS'`weight)])],
     [sum_ctr(variables,='TABLE'`summation)])).

ctr_eval(indexed_sum,[reformulation(indexed_sum_r)]).

indexed_sum_r(ITEMS,TABLE) :-
    length(ITEMS,I),
    length(TABLE,T),
    I>0,
    T>0,
    collection(ITEMS,[dvar(1,T),dvar]),
    collection(TABLE,[int(1,T),dvar]),
    collection_increasing_seq(TABLE,[1]),
    get_attr1(ITEMS,ITEMS_INDEXES),
    get_attr2(ITEMS,ITEMS_WEIGHTS),
    get_attr2(TABLE,TABLE_TSUMS),
    indexed_sum1(  
      1,  
      T,  
      TABLE_TSUMS,  
      ITEMS_INDEXES,  
      ITEMS_WEIGHTS).

indexed_sum1(I,T,[],_32611,_32612) :-
    I>T,
    !.

indexed_sum1(I,T,[SUM|R],ITEMS_INDEXES,ITEMS_WEIGHTS) :-
    indexed_sum2(ITEMS_INDEXES,ITEMS_WEIGHTS,I,TERM),
    call(SUM#=TERM),
    I1 is I+1,
    indexed_sum1(I1,T,R,ITEMS_INDEXES,ITEMS_WEIGHTS).

indexed_sum2([],[],_32607,0).
indexed_sum2([J|R],[W|S],I,W*B+T) :-
    B#<=>J#=I,
    indexed_sum2(R,S,I,T).
B.176 inflexion

◊ **Meta-Data:**

```prolog
ctr_date(inflexion,['20000128','20030820','20040530']).

ctr_origin(inflexion,'N.˘Beldiceanu',[]).

ctr_arguments(inflexion,[
    [\'N\'-dvar,'VARIABLES\'-collection(var-dvar)]].

ctr_restrictions(inflexion,[
    [\'N\'>=0,
     \'N\'<\max(0,size('VARIABLES')-2),
     required('VARIABLES',var)]].

ctr_example(inflexion,
    inflexion(3,
        [[[\text{var-1}]],
         [[\text{var-1}]],
         [[\text{var-4}]],
         [[\text{var-8}]],
         [[\text{var-8}]],
         [[\text{var-2}]],
         [[\text{var-7}]],
         [[\text{var-1}]]]).

ctr_typical(inflexion,[\'N\'>0,size('VARIABLES')>2,\text{range('VARIABLES'~\text{var})>1}].

ctr_exchangeable(inflexion,[\text{items('VARIABLES',reverse)},\text{translate(['VARIABLES'~\text{var}])}].

ctr_eval(inflexion,[\text{automaton(inflexion_a)}]).

inflexion_a(\text{FLAG,N,VARIABLES}) :-
    collection(VARIABLES,[dvar]),
    length(VARIABLES,L),
    MAX is \max(0,L-2),
    check_type(dvar(0,MAX),N),
    
    flag(\text{FLAG}),
    N = \text{N},
    VARIABLES = \text{VARIABLES},
    
    \text{update}(
        \text{dvar}(0,MAX),N,
        \text{VARIABLES})
```

---

**APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE**

2452
inflexion_signature(VARIABLES, SIGNATURE),
automaton(
    SIGNATURE,
    _17580,
    SIGNATURE,
    [source(s), sink(i), sink(j), sink(s)],
    [arc(s,1,s),
     arc(s,2,i),
     arc(s,0,j),
     arc(i,1,i),
     arc(i,2,i),
     arc(i,0,j,[C+1]),
     arc(j,1,j),
     arc(j,0,j),
     arc(j,2,i,[C+1])],
    [C],
    [0],
    [COUNT]),
COUNT#=N#<=>FLAG.

inflexion_signature([],[]).

inflexion_signature([_15918],[]) :- !.

inflexion_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss]) :-
    S in 0..2,
    VAR1#>VAR2#<=>S#=0,
    VAR1#=VAR2#<=>S#=1,
    VAR1#<VAR2#<=>S#=2,
    inflexion_signature([[var-VAR2]|VARs],Ss).
B.177 inside_sboxes

◊ Meta-Data:

ctr_date(inside_sboxes,['20070622','20090725']).

ctr_origin(inside_sboxes,
  Geometry, derived from \cite{RandellCuiCohn92},[]).

ctr_synonyms(inside_sboxes,[inside]).

ctr_types(inside_sboxes,
  ['VARIABLES'-collection(v-dvar),
   'INTEGERS'-collection(v-int),
   'POSITIVES'-collection(v-int)]).

ctr_arguments(inside_sboxes,
  ['K'-int,
   'DIMS'-sint, 
   'OBJECTS'-collection(oid-int,sid-int,x-'VARIABLES'),
   'SBOXES'-collection(sid-int,t-'INTEGERS',l-'POSITIVES')]).

ctr_restrictions(inside_sboxes,
  [size('VARIABLES')>=1,
   size('INTEGERS')>=1,
   size('POSITIVES')>=1,
   required('VARIABLES',v),
   size('VARIABLES')='K',
   required('INTEGERS',v),
   size('INTEGERS')='K',
   required('POSITIVES',v),
   size('POSITIVES')='K',
   'POSITIVES' hat v>0,
   'K'>0,
   'DIMS'>=0,
   'DIMS'<='K',
   increasing_seq('OBJECTS',[oid]),
   required('OBJECTS',[oid,sid,x]),
   'OBJECTS'^oid=1,
   'OBJECTS'^oid=<size('OBJECTS'),
   'OBJECTS'^sid=1,
'OBJECTS' \^ sid=<size('SBOXES'),
size('SBOXES')>=1,
required('SBOXES', [sid, t, l]),
'SBOXES' \^ sid=1,
'SBOXES' \^ sid=<size('SBOXES'),
do_not_overlap('SBOXES')).

ctr_example(
  inside_sboxes,
  inside_sboxes(2,
    [0,1],
    [[oid-1, sid-1, x-[[v-3], [v-3]]],
     [oid-2, sid-2, x-[[v-2], [v-2]]],
     [oid-3, sid-3, x-[[v-1], [v-1]]],
     [[sid-1, t-[[v-0], [v-0]], l-[[v-1], [v-1]]],
     [sid-2, t-[[v-0], [v-0]], l-[[v-3], [v-3]]],
     [sid-3, t-[[v-0], [v-0]], l-[[v-5], [v-5]]])).

ctr_typical(inside_sboxes, [size('OBJECTS') > 1]).

ctr_exchangeable(
  inside_sboxes,
  [items('SBOXES', all),
   items_sync('OBJECTS' \^ x, 'SBOXES' \^ t, 'SBOXES' \^ l, all)].

ctr_eval(inside_sboxes, [logic(inside_sboxes_g)]).

ctr_logic(
  inside_sboxes,
  [DIMENSIONS, OIDS],
  [(origin(O1, S1, D) \rightarrow O1 \^ x(D) + S1 \^ t(D)),
   (end(O1, S1, D) \rightarrow O1 \^ x(D) + S1 \^ t(D) + S1 \^ l(D)),
   (inside_sboxes(Dims, O1, S1, O2, S2) \rightarrow
    forall(D, Dims,
      origin(O2, S2, D) \#<origin(O1, S1, D)\#/\end(O1, S1, D) \#<end(O2, S2, D))))],
  (inside_objects(Dims, O1, O2) \rightarrow
   forall(S1, sboxes([O1\^sid]),
     exists(S2, sboxes([O2\^sid]),
       ...)]}).
inside_sboxes(Dims,O1,S1,O2,S2))))),
(all_inside(Dims,OIDS)--->
forall(
  O1,
  objects(OIDS),
  forall(
    O2,
    objects(OIDS),
    O1ˆoid#<O2ˆoid#=>inside_objects(Dims,O1,O2))
  ),
all_inside(DIMENSIONS,OIDS))).

ctr_contractible(inside_sboxes,[],’OBJECTS’,suffix).

inside_sboxes_g(K,_28723,[],_28725) :-
  !,
  check_type(int_gteq(1),K).

inside_sboxes_g(K,_DIMS,OBJECTS,SBOXES) :-
  length(OBJECTS,O),
  length(SBOXES,S),
  O>0,
  S>0,
  check_type(int_gteq(1),K),
  collection(OBJECTS,[int(1,O),dvar(1,S),col(K,[dvar]))],
  collection(
    SBOXES,
    [int(1,S),col(K,[int]),col(K,[int_gteq(1)]))]),
  get_attr1(OBJECTS,OIDS),
  get_attr2(OBJECTS,SIDS),
  get_col_attr3(OBJECTS,1,XS),
  get_attr1(SBOXES,SIDES),
  get_col_attr2(SBOXES,1,TS),
  get_col_attr3(SBOXES,1,TL),
  collection_increasing_seq(OBJECTS,[1]),
  geost1(OIDS,SIDS,XS,Objects),
  geost2(SIDES,TS,TL,Sboxes),
  geost_dims(1,K,DIMENSIONS),
  ctr_logic(inside_sboxes,[DIMENSIONS,OIDS],Rules),
  geost(Objects,Sboxes,[overlap(true)],Rules).
B.178  int_value_precede

\textbf{META-DATA:}

\begin{verbatim}
ctr_date(int_value_precede, ['20041003']).

ctr_origin(int_value_precede,'\cite{YatChiuLawJimmyLee04}',[]).

ctr_synonyms(  
  int_value_precede,  
  [precede, precedence, value_precede]).

ctr_arguments(  
  int_value_precede,  
  ['S'-int,'T'-int,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(  
  int_value_precede,  
  ['S'='T',required('VARIABLES',var)]).

ctr_example(  
  int_value_precede,  
  int_value_precede(  
    0,  
    1,  
    [[var-4],[var-0],[var-6],[var-1],[var-0]])).

ctr_typical(  
  int_value_precede,  
  ['S'<'T',  
    size('VARIABLES')>1,  
    atleast(1,'VARIABLES','S'),  
    atleast(1,'VARIABLES','T')]).

ctr_exchangeable(  
  int_value_precede,  
  [vals(  
    ['VARIABLES'~var],  
    int(notin(['S','T'])),  
    =\=,  
    dontcare,  
    dontcare),  
  vals(  
    ['S','T','VARIABLES'~var],  
    int(['S','T'])),  
    =\=,  
  ])
\end{verbatim}
ctr_eval(int_value_precede, [automaton(int_value_precede_a)]).

ctr_contractible(int_value_precede, [], 'VARIABLES', suffix).

ctr_aggregate(int_value_precede, [], [id, id, union]).

int_value_precede_a(1, S, T, []) :-
  !,
  check_type(int, S),
  check_type(int, T),
  S =\= T.

int_value_precede_a(0, _S, _T, []) :-
  !,
  fail.

int_value_precede_a(FLAG, S, T, VARIABLES) :-
  check_type(int, S),
  check_type(int, T),
  S =\= T,
  collection(VARIABLES, [dvar]),
  int_value_precede_signature(VARIABLES, SIGNATURE, S, T),
  AUTOMATON =
  automaton(  
    SIGNATURE, _22909, 
    [source(s), sink(s), sink(t)],
    [arc(s,3,s),
     arc(s,1,t),
     arc(t,1,t),
     arc(t,2,t),
     arc(t,3,t)],
    [], [], []),
  automaton_bool(FLAG, [1,2,3], AUTOMATON).

int_value_precede_signature([], [], _21127, _21128).

int_value_precede_signature([[var-VAR]|VARs], [SI|SIs], S, T) :-
  SI in 1..3,
  VAR#=S#<=>SI#=1,
VAR# = T# <= SI# = 2, 
VAR# = S# / VAR# = T# <= SI# = 3, 
int_value_precede_signature(VARs, SIs, S, T).
B.179 int_value_precede_chain

◊ **Meta-Data:**

```prolog
ctr_date(
    int_value_precede_chain,
    ['20041003','20090728','20090822']).

ctr_origin(
    int_value_precede_chain,
    \cite{YatChiuLawJimmyLee04},
    []).

ctr_synonyms(
    int_value_precede_chain,
    [precede,precedence,value_precede_chain]).

ctr_arguments(
    int_value_precede_chain,
    ['VALUES'-collection(var-int),
     'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    int_value_precede_chain,
    [required('VALUES',var),
     distinct('VALUES',var),
     required('VARIABLES',var)]).

ctr_example(
    int_value_precede_chain,
    int_value_precede_chain(
        [[var-4],[var-0],[var-1]],
        [[var-4],[var-0],[var-6],[var-1],[var-0]])).

ctr_typical(
    int_value_precede_chain,
    [size('VALUES')>1,
     strictly_increasing('VALUES'),
     size('VARIABLES')>size('VALUES'),
     range('VARIABLES'\-var)>1,
     used_by('VARIABLES','VALUES')]).

ctr_exchangeable(
    int_value_precede_chain,
    [vals(
        ['VARIABLES'\-var],
        
```
int(notin('VALUES'^var)),
 =\=,
dontcare,
dontcare))).

ctr_eval(
  int_value_precede_chain,
  [automaton(int_value_precede_chain_a)]).

ctr_contractible(int_value_precede_chain,[],'VALUES',any).

ctr_contractible(int_value_precede_chain,[],'VARIABLES',suffix).

ctr_aggregate(int_value_precede_chain,[],[id,union]).

int_value_precede_chain_a(FLAG,[],VARIABLES) :-
  !,
  collection(VARIABLES,[dvar]),
  ( FLAG=1 ->
    true
  ; fail
).

int_value_precede_chain_a(FLAG,VALUES,[]) :-
  !,
  collection(VALUES,[int]),
  get_attr1(VALUES,VALS),
  all_different(VALS),
  ( FLAG=1 ->
    true
  ; fail
).

int_value_precede_chain_a(FLAG,VALUES,VARIABLES) :-
  collection(VALUES,[int]),
  collection(VARIABLES,[dvar]),
  length(VALUES,1),
  !,
  ( FLAG=1 ->
    true
  ; fail
).

int_value_precede_chain_a(FLAG,VALUES,VARIABLES) :-
  collection(VALUES,[int]),
  collection(VARIABLES,[dvar]),
get_attr1(VALUES, VALS),
all_different(VALS),
length(VALS, N),
get_attr1(VARIABLES, VARS),
int_value_precede_chain_gen_complement(
  VARS,
  VALS,
  COMPLEMENT),
int_value_precede_chain_gen_states1(0, N, STATES),
int_value_precede_chain_gen_transitions(
  N,
  VALS,
  COMPLEMENT,
  STATES,
  TRANSITIONS),
nth0(0, STATES, S0),
int_value_precede_chain_gen_states2(
  STATES,
  S0,
  AUTOMATON_STATES),
AUTOMATON=automaton(
  VARS,
  _26927,
  VARS,
  AUTOMATON_STATES,
  TRANSITIONS,
  [],
  [],
  []),
append(VALS, COMPLEMENT, ALPHABET),
automaton_bool(FLAG, ALPHABET, AUTOMATON).

int_value_precede_chain_gen_states2([], S0, [source(S0)]) :- !.

int_value_precede_chain_gen_states2([S|R], S0, [sink(S)|T]) :-
  int_value_precede_chain_gen_states2(R, S0, T).

int_value_precede_chain_gen_complement(VARS, VALS, COMPLEMENT) :-
  union_dom_set(VARS, UNION),
  list_to_fdset(VALS, VALUES),
  fdset_subtract(UNION, VALUES, DIFFERENCE),
  fdset_to_list(DIFFERENCE, COMPLEMENT).

int_value_precede_chain_gen_states1(I, N, []) :-
int_value_precede_chain_gen_states1(I,N,[INAME|R]) :-
I=<N,
number_codes(I,ICODE),
atom_codes(IATOM,ICODE),
atom_concat(s,IATOM,INAME),
I1 is I+1,
int_value_precede_chain_gen_states1(I1,N,R).

int_value_precede_chain_gen_transitions(N, VALS, COMPLEMENT, STATES, TRANSITIONS) :-
N1 is N-1,
int_value_precede_chain_gen_transitions1(0, N1, VALS, STATES, TR1),
int_value_precede_chain_gen_transitions2(1, N, VALS, STATES, TR2),
int_value_precede_chain_gen_transitions3(0, N, VALS, STATES, COMPLEMENT, TR3),
append(TR1,TR2,TR12),
append(TR12,TR3,TRANSITIONS).

int_value_precede_chain_gen_transitions1(I,N,_,_23423,_23424,[]) :-
I>N1,
!.

int_value_precede_chain_gen_transitions1(I,
N1, VALS, STATES, [arc(Si,Vii,Sii)|R]) :-
I=<N1,  
I1 is I+1,  
nth0(I,STATES, Si),  
nth1(I1,VALS,Vii),  
nth0(I1,STATES,Sii),  
int_value_precede_chain_gen_transitions1(
I1,
N1,
VALS,
STATES,
R).

int_value_precede_chain_gen_transitions2(I,N1, _23423, _23424, []) :-
I>N1, !.

int_value_precede_chain_gen_transitions2(I,
N1,
VALS,
STATES,
TRANSITIONS) :-
I=<N1,  
int_value_precede_chain_gen_transitions21(1,
I,
VALS,
STATES,
TR),
I1 is I+1,  
int_value_precede_chain_gen_transitions2(
I1,
N1,
VALS,
STATES,
R),
append(TR,R,TRANSITIONS).

int_value_precede_chain_gen_transitions21(J,I, _23423, _23424, []) :-
J>I, !.
int_value_precede_chain_gen.transitions21(  
  J,  
  I,  
  VALS,  
  STATES,  
  [arc(Si,Vj,Si)|R]) :-  
  J=<I,  
  nth0(I,STATES,Si),  
  nth1(J,VALS,Vj),  
  J1 is J+1,  
  int_value_precede_chain_gen.transitions21(  
    J1,  
    I,  
    VALS,  
    STATES,  
    R).

int_value_precede_chain_gen.transitions3(  
    _23656,  
    _23702,  
    _23748,  
    _23794,  
    [],  
    []) :-  
    !.

int_value_precede_chain_gen.transitions3(  
    I,  
    N1,  
    _23762,  
    _23808,  
    _23854,  
    []) :-  
    I>N1,  
    !.

int_value_precede_chain_gen.transitions3(  
    I,  
    N1,  
    VALS,  
    STATES,  
    [C|CC],  
    TRANSITIONS) :-  
    I=<N1,  
    length([C|CC],LC),  
    int_value_precede_chain_gen.transitions31(  
      J,  
      I,  
      VALS,  
      STATES,  
      [arc(Si,Vj,Si)|R])
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

1, LC, I, [C|CC], STATES, TR),
I1 is I+1,
int_value_precede_chain_gen_transitions31( I1, N1, VALS, STATES, [C|CC], R),
append(TR,R,TRANSITIONS).

int_value_precede_chain_gen_transitions31( J, LC, _23762, _23808, _23854, [] ) :-
J>LC, !.

int_value_precede_chain_gen_transitions31( J, LC, I, C, STATES, [arc(Si,Cj,Si)|R] ) :-
J=<LC, nth0(I,STATES, Si), nth1(J,C,Cj), J1 is J+1,
int_value_precede_chain_gen_transitions31( J1, LC, I, C, STATES, R).
B.180 interval_and_count

◊ META-DATA:

ctr_date(
    interval_and_count,
    ['20000128','20030820','20040530','20060810']).

ctr_origin(interval_and_count,'\cite{Cousin93}',[]).

ctr_arguments(
    interval_and_count,
    ['ATMOST'-int,
     'COLOURS'-collection(val-int),
     'TASKS'-collection(origin-dvar,colour-dvar),
     'SIZE_INTERVAL'-int]).

ctr_restrictions(
    interval_and_count,
    ['ATMOST'>=0,
     required('COLOURS',val),
     distinct('COLOURS',val),
     required('TASKS',[origin,colour]),
     'TASKS'\^origin>=0,
     'SIZE_INTERVAL'>0]).

ctr_example(
    interval_and_count,
    interval_and_count(2,
        [[val-4]],
        [[origin-1,colour-4],
         [origin-0,colour-9],
         [origin-10,colour-4],
         [origin-4,colour-4]],
        5)).

ctr_typical(
    interval_and_count,
    ['ATMOST'>0,
     'ATMOST'<size('TASKS'),
     size('COLOURS')>0,
     size('TASKS')>1,
     range('TASKS'\^origin)>1,
     range('TASKS'\^colour)>1,
     'SIZE_INTERVAL'>1]).
ctr_exchangeable(
    interval_and_count,
    [vals([‘ATMOST’], int, <, dontcare, dontcare),
     items(‘COLOURS’, all),
     items(‘TASKS’, all),
     translate([‘TASKS’^origin]),
     vals([‘TASKS’^origin],
          intervals(‘SIZE_INTERVAL’),
          =, dontcare, dontcare),
     vals([‘TASKS’^colour],
          comp(‘COLOURS’^val),
          =, dontcare, dontcare))).

ctr_graph(
    interval_and_count,
    [‘TASKS’, ‘TASKS’],
    2,
    [‘PRODUCT’>>collection(tasks1,tasks2)],
    [tasks1^origin/‘SIZE_INTERVAL’=tasks2^origin/‘SIZE_INTERVAL’],
    [],
    [],
    [Succ>>
     source,
     variables-
     col(‘VARIABLES’-collection(var-dvar),
         [item(var-‘TASKS’^colour)]),
     [among_low_up(0,’ATMOST’,variables,’COLOURS’)]).
collection(COLOURS,[int]),
get_attr1(COLOURS,COLS),
all_different(COLS),
collection(TASKS,[dvar_gteq(0),dvar]),
check_type(int_gteq(1),SIZE_INTERVAL),

( COLOURS=[] ->
  true
; TASKS=[] ->
  true
; get_attr1(TASKS,TORIS),
  get_attr2(TASKS,TCOLS),
  interval_and_count1(TCOLS,COLS,LB),
  get_maximum(TORIS,MAX),
  MAXK is(MAX+SIZE_INTERVAL-1)//SIZE_INTERVAL,
  interval_and_count2(
    0,
    MAXK,
    SIZE_INTERVAL,
    ATMOST,
    LB,
    TORIS)
).

interval_and_count1([],_41670,[]).

interval_and_count1([TC|R],COLS,[B|S]) :-
  build_or_var_in_values(COLS,TC,TERM),
  call(B#<=>TERM),
  interval_and_count1(R,COLS,S).

interval_and_count2(K,MAXK,_41674,_41675,_41676,_41677) :-
  K>MAXK,
  !.

interval_and_count2(K,MAXK,SIZE_INTERVAL,ATMOST,LB,TORIS) :-
  K=<MAXK,
  interval_and_count3(LB,TORIS,K,SIZE_INTERVAL,SUMB),
  call(SUMB#=<ATMOST),
  K1 is K+1,
  interval_and_count2(
    K1,
    MAXK,
    SIZE_INTERVAL,
    ATMOST,
    LB,
    TORIS).
interval_and_count3([],[],_41671,_41672,0).

interval_and_count3([B|R],[O|S],K,SIZE_INTERVAL,BK+T) :-
    SK is K*SIZE_INTERVAL,
    TK is SK+SIZE_INTERVAL-1,
    BK#<=>B#/\O#>=SK#/\O#=<TK,
    interval_and_count3(R,S,K,SIZE_INTERVAL,T).
B.181 interval_and_sum

◊ META-DATA:

ctr_date(interval_and_sum,[’20000128′,’20030820′,’20060810′]).

ctr_origin(interval_and_sum,’Derived from %c.’,[cumulative]).

ctr_arguments(
  interval_and_sum,
  [’SIZE_INTERVAL’-int,
   ’TASKS’-collection(origin-dvar,height-dvar),
   ’LIMIT’-int]).

ctr_restrictions(
  interval_and_sum,
  [’SIZE_INTERVAL’>0,
   required(’TASKS’,[origin,height]),
   ’TASKS’^origin>=0,
   ’TASKS’^height>=0,
   ’LIMIT’>=0]).

ctr_example(
  interval_and_sum,
  interval_and_sum( 5,
    [origin-1,height-2],
    [origin-10,height-2],
    [origin-10,height-3],
    [origin-4,height-1],
    5)).

ctr_typical(
  interval_and_sum,
  [’SIZE_INTERVAL’>1,
   size(’TASKS’)>1,
   range(’TASKS’^origin)>1,
   range(’TASKS’^height)>1,
   ’LIMIT’<sum(’TASKS’^height)]).

ctr_exchangeable(
  interval_and_sum,
  [items(’TASKS’,all),
   translate([’TASKS’^origin]),
   vals{
     [’TASKS’^origin],
intervals('SIZE_INTERVAL'),
  =,
  dontcare,
  dontcare),
vals(['TASKS'\^height],int(\=0),>,dontcare,dontcare),
vals(['LIMIT'],int,\<,dontcare,dontcare)).

ctr_graph(
  interval_and_sum,
  ['TASKS','TASKS'],
  2,
  ['PRODUCT']\>\collection(tasks1,tasks2]),
  [tasks1\^origin/'SIZE_INTERVAL'= tasks2\^origin/'SIZE_INTERVAL'],
  [],
  [],
  [SUCC>>
    [source,
      variables-
        col('VARIABLES'-\collection(var-dvar),
          [item(var-'TASKS'\^height)])],
    [sum_ctr(variables,\<,'LIMIT')]]).

ctr_eval(interval_and_sum,[reformulation(interval_and_sum_r)]).

ctr_contractible(interval_and_sum,[,'TASKS',any).

interval_and_sum_r(SIZE_INTERVAL,TASKS,LIMIT) :-
  check_type(int_gteq(1),SIZE_INTERVAL),
  collection(TASKS,[dvar_gteq(0),dvar_gteq(0)]),
  check_type(int_gteq(0),LIMIT),
  ( TASKS=[] ->
    true
  ;
    get_attr1(TASKS,ORIS),
    get_attr2(TASKS,HEIGHTS),
    get_maximum(ORIS,MAX),
    MAXK is(MAX+SIZE_INTERVAL-1)//SIZE_INTERVAL,
    interval_and_sum1(0,
      MAXK,
      SIZE_INTERVAL,
      LIMIT,
      ORIS,
      HEIGHTS)
  ).
interval_and_sum1(K, MAXK, _38784, _38785, _38786, _38787) :-
    K > MAXK,
    !.

interval_and_sum1(K, MAXK, SIZE_INTERVAL, LIMIT, ORIS, HEIGHTS) :-
    K =< MAXK,
    interval_and_sum2(ORIS, HEIGHTS, K, SIZE_INTERVAL, SUM),
    call(SUM#=<LIMIT),
    K1 is K+1,
    interval_and_sum1(
        K1,
        MAXK,
        SIZE_INTERVAL,
        LIMIT,
        ORIS,
        HEIGHTS).

interval_and_sum2([], [], _38781, _38782, 0).

interval_and_sum2([O|R], [H|S], K, SIZE_INTERVAL, H*B+T) :-
    SK is K*SIZE_INTERVAL,
    TK is SK+SIZE_INTERVAL-1,
    B#=<O#那就是SK/\O#=<TK,
B.182  inverse

◇ Meta-Data:

\begin{verbatim}
ctr_date(inverse,[’20000128’,’20030820’,’20040530’,’20060810’]).
ctr_origin(inverse,’\index{CHIP|indexuse}CHIP’,[]).
ctr_synonyms(inverse,[assignment,channel,inverse_channeling]).
ctr_arguments(
  inverse,
  [’NODES’-collection(index-int,succ-dvar,pred-dvar)]).
ctr_restrictions(
  inverse,
  [required(’NODES’,[index,succ,pred]),
   ’NODES’^index>=1,
   ’NODES’^index=<size(’NODES’),
   distinct(’NODES’,index),
   ’NODES’^succ>=1,
   ’NODES’^succ=<size(’NODES’),
   ’NODES’^pred>=1,
   ’NODES’^pred=<size(’NODES’)]).
ctr_example(
  inverse,
  inverse(
    [[index-1,succ-2,pred-2],
    [index-2,succ-1,pred-1],
    [index-3,succ-5,pred-4],
    [index-4,succ-3,pred-5],
    [index-5,succ-4,pred-3]]).
ctr_typical(inverse,[size(’NODES’)>,1]).
ctr_exchangeable(
  inverse,
  [items(’NODES’,all),
   attrs_sync(’NODES’,[[index],[succ,pred]])].
ctr_graph(
  inverse,
  [’NODES’],
  2,
  [’CLIQUE’]>>collection(nodes1,nodes2)],
\end{verbatim}
[nodes1\textasciitilde\text{succ}=nodes2\textasciitilde\text{index}, nodes2\textasciitilde\text{pred}=nodes1\textasciitilde\text{index}],
['\text{NARC}=\text{size('NODES')},
[]).

\text{ctr\_eval}(\text{inverse}, [\text{reformulation}(\text{inverse\_r})]).

\text{ctr\_pure\_functional\_dependency}(\text{inverse}, []).

\text{ctr\_functional\_dependency}(\text{inverse}, 1-2, [1-1, 1-3]).

\text{ctr\_functional\_dependency}(\text{inverse}, 1-3, [1-1, 1-2]).

\text{inverse\_r}([[]]) :-
!.

\text{inverse\_r}(\text{NODES}) :-
  \text{length}(\text{NODES}, \text{N}),
  \text{collection}(\text{NODES}, [\text{int}(1, \text{N}), \text{dvar}(1, \text{N}), \text{dvar}(1, \text{N})]),
  \text{get\_attr1}(\text{NODES}, \text{INDEXES}),
  \text{get\_attr2}(\text{NODES}, \text{SUCCS}),
  \text{get\_attr3}(\text{NODES}, \text{PREDS}),
  \text{all\_different}(\text{INDEXES}),
  \text{all\_different}(\text{SUCCS}),
  \text{all\_different}(\text{PREDS}),
  \text{inverse1}(\text{SUCCS}, \text{INDEXES}, \text{PREDS}, \text{INDEXES}).

\text{inverse1}([], [], _44185, _44186).

\text{inverse1}([\text{S}\_\text{I}|R], [\text{I}|S], \text{PREDS}, \text{INDEXES}) :-
  \text{inverse2}(\text{PREDS}, \text{INDEXES}, \text{S}\_\text{I}, \text{I}),
  \text{inverse1}(R, S, \text{PREDS}, \text{INDEXES}).

\text{inverse2}([], [], _44185, _44186).

\text{inverse2}([\text{P}\_\text{J}|\text{R}], [\text{J}|\text{S}], \text{S}\_\text{I}, \text{I}) :-
  \text{S}\_\text{I}\#=\text{J}\#\Leftarrow\text{P}\_\text{J}\#=\text{I},
  \text{inverse2}(R, S, \text{S}\_\text{I}, \text{I}).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.183 inverse_offset

◊ Meta-Data:

ctr_date(inverse_offset,['20091404']).

ctr_origin(inverse_offset,]\index{Gecode|indexuse} Gecode',[]).

ctr_synonyms(inverse_offset,[channel]).

ctr_arguments(
    inverse_offset,
    ['SOFFSET'-int,
     'POFFSET'-int,
     'NODES'-collection(index-int,succ-dvar,pred-dvar)]).

ctr_restrictions(
    inverse_offset,
    [required('NODES',[index,succ,pred]),
     'NODES'\index{index}index>=1,
     'NODES'\index{index}index=<size('NODES'),
     distinct('NODES',index),
     'NODES'\index{index}succ=1+'SOFFSET',
     'NODES'\index{index}succ=<size('NODES')+SOFFSET',
     'NODES'\index{index}pred=1+'POFFSET',
     'NODES'\index{index}pred=<size('NODES')+POFFSET']).

ctr_example(
    inverse_offset,
    inverse_offset(-1,0,[
        [index-1,succ-4,pred-3],
        [index-2,succ-2,pred-5],
        [index-3,succ-0,pred-2],
        [index-4,succ-6,pred-8],
        [index-5,succ-1,pred-1],
        [index-6,succ-7,pred-7],
        [index-7,succ-5,pred-4],
        [index-8,succ-3,pred-6]])).

ctr_typical(
    inverse_offset,
    ['SOFFSET'>= -1,
     'SOFFSET'=<1,
     'POFFSET'>= -1,
'POFFSET'=<1,
    size('NODES')>1]).

ctr_exchangeable(inverse_offset,[items('NODES',all)]).

ctr_graph(
    inverse_offset,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1\succ-'SOFFSET'=nodes2\index, 
     nodes2\pred-'POFFSET'=nodes1\index],
    ['NARC'=size('NODES')],
    []).

ctr_pure_functional_dependency(inverse_offset,[]).

ctr_functional_dependency(inverse_offset,3-2,[1,2,3-1,3-3]).

ctr_functional_dependency(inverse_offset,3-3,[1,2,3-1,3-2]).
B.184 inverse_set

◊ Meta-Data:

ctr_date(inverse_set, ['20041211', '20060810']).

ctr_origin(inverse_set, 'Derived from %c.', [inverse]).

ctr_arguments( inverse_set, ['X'-collection(index-int,set-svar), 'Y'-collection(index-int,set-svar)]).

ctr_restrictions( inverse_set, [required('X', [index,set]), required('Y', [index,set]), increasing_seq('X',index), increasing_seq('Y',index), 'X'°index>=1, 'X'°index=<size('X'), 'Y'°index>=1, 'Y'°index=<size('Y'), 'X'°set>=1, 'X'°set=<size('X'), 'Y'°set>=1, 'Y'°set=<size('X')]].

ctr_example( inverse_set, inverse_set( [[index-1,set-{2,4}]], [[index-2,set-{4}]], [[index-3,set-{1}]], [[index-4,set-{4}]]), [[index-1,set-{3}]], [[index-2,set-{1}]], [[index-3,set-{}]], [[index-4,set-{1,2,4}]], [[index-5,set-{}]]).

ctr_typical(inverse_set, [size('X')>1, size('Y')>1]).

ctr_exchangeable( inverse_set, [args([['X','Y']]), items('X', all), items('Y', all)]).
ctr_graph(
    inverse_set,
    ['X','Y'],
    2,
    ['PRODUCT'>>collection(x,y)],
    [y\text{\textasciitilde}index in_set x\text{\textasciitilde}set\subseteq\Rightarrow x\text{\textasciitilde}index in_set y\text{\textasciitilde}set],
    ['NARC'=size('X')\ast size('Y')],
    []).

APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.185 inverse_within_range

◊ Meta-Data:

ctr_date(inverse_within_range, ['20060517', '20060810']).

ctr_origin(inverse_within_range, 'Derived from %c.', [inverse]).

ctr_synonyms(
  inverse_within_range,
  [inverse_in_range, inverse_range]).

ctr_arguments(
  inverse_within_range,
  ['X'-collection(var-dvar), 'Y'-collection(var-dvar)]).

ctr_restrictions(
  inverse_within_range,
  [required('X', var), required('Y', var)]).

ctr_example(
  inverse_within_range,
  inverse_within_range(
    [[var-9], [var-4], [var-2]],
    [[var-9], [var-3], [var-9], [var-2]])).

ctr_typical(
  inverse_within_range,
  [size('X')>1,
   range('X' ^ var)>1,
   size('Y')>1,
   range('Y' ^ var)>1]).

ctr_exchangeable(inverse_within_range, [args([['X', 'Y']])]).

ctr_graph(
  inverse_within_range,
  ['X', 'Y'],
  2,
  ['SYMMETRIC PRODUCT'] >> collection(s1, s2],
  [s1 ^ var=s2 ^ key],
  [],
  ['BIPARTITE', 'NO_LOOP', 'SYMMETRIC']).
B.186 ith_pos_different_from_0

◊ **META-DATA:**

```prolog
ctr_date(ith_pos_different_from_0, ['20040811']).
ctr_origin(ith_pos_different_from_0, 'N.˘Beldiceanu', []).
ctr_arguments(
    ith_pos_different_from_0,
    ['ITH'-int,'POS'-dvar,'VARIABLES'-collection(var-dvar)]).
ctr_restrictions(
    ith_pos_different_from_0,
    ['ITH'>=1,
    'ITH'=<size('VARIABLES'),
    'POS'>='ITH',
    'POS'=<size('VARIABLES'),
    required('VARIABLES',var)]).
ctr_example(
    ith_pos_different_from_0,
    ith_pos_different_from_0(2,
        4,
        [[var-3],[var-0],[var-0],[var-8],[var-6]])).
ctr_typical(
    ith_pos_different_from_0,
    [size('VARIABLES')>1,
    range('VARIABLES'ˆvar)>1,
    atleast(1,'VARIABLES',0)]).
ctr_exchangeable(
    ith_pos_different_from_0,
    [vals(
        ['VARIABLES'ˆvar],
        int(\=\=(0)),
        \=\=,
        dontcare,
        dontcare)]).
ctr_eval(
    ith_pos_different_from_0,
    [automaton(ith_pos_different_from_0_a)]).
```
ctr_extensible(ith_pos_different_from_0,[],'VARIABLES',suffix).

ith_pos_different_from_0_a(FLAG,ITH,POS,VARIABLES) :-
collection(VARIABLES,[dvar]),
length(VARIABLES,N),
integer(ITH),
ITH>=1,
ITH=<N,
check_type(dvar(ITH,N),POS),
ith_pos_different_from_0_signature(VARIABLES,SIGNATURE),
automaton(
  SIGNATURE,
  _16768,
  SIGNATURE,
  [source(s),sink(s)],
  [arc(s,0,s,(C#<ITH->[C+1,D+1];C#>=ITH->[C,D]))],
  [arc(s,1,s,(C#<ITH->[C,D+1];C#>=ITH->[C,D]))],
  [C,D],
  [0,0],
  [C1,D1]),
  C1#=ITH#/\D1#<=POS#<=>FLAG.

ith_pos_different_from_0_signature([],[]).

ith_pos_different_from_0_signature([V#|VARs],[S|Ss]) :-
  V#=<S,
  ith_pos_different_from_0_signature(VARs,Ss).
B.187  \texttt{k\_alldifferent}

\textbf{\textcircled{\small{\# Meta-Data:}}}

\begin{verbatim}
ctr_date(k_alldifferent, ['20050618', '20060811']).

ctr_origin(k_alldifferent, \cite{ElbassioniKatrielKutzMahajan05}, []).

ctr_synonyms(k_alldifferent, [k_alldiff, k_alldistinct, someifferent]).

ctr_types(k_alldifferent, ['X' - collection(x-dvar)]).

ctr_arguments(k_alldifferent, ['VARS' - collection(vars-'X')]).

ctr_restrictions(k_alldifferent, [size('X')>=1, required('X',x), required('VARS',vars), size('VARS')>=1]).

ctr_example(k_alldifferent, k_alldifferent([[vars-[[x-5],[x-6],[x-0],[x-9],[x-3]]], [vars-[[x-5],[x-6],[x-1],[x-2]]]])).

ctr_typical(k_alldifferent, [size('X')>1, size('VARS')>1]).

ctr_exchangeable(k_alldifferent, [items('VARS',all), items('VARS'~vars,all), vals(['VARS'~vars~x,int,=\=,all,dontcare])].

ctr_graph(k_alldifferent, ['VARS'~vars], 2, foreach('VARS', ['CLIQUE'~collection(x1,x2)]), [x1\^x=x2\^x]).
\end{verbatim}
['MAX_NS CC'=<1],
[]).

ctr_eval(k_alldifferent,[reformulation(k_alldifferent_r)]).

ctr_contractible(k_alldifferent,[],'VARS',any).

k_alldifferent_r(VARS) :-
    length(VARS,N),
    N>0,
    collection(VARS,[non_empty_col([dvar])])),
    get_col_attr1(VARS,1,VS),
    k_alldifferent1(VS).

k_alldifferent1([]).

k_alldifferent1([V|R]) :-
    all_different(V),
    k_alldifferent1(R).
B.188 k_cut

◊ **META-DATA:**

ctr_date(k_cut,['20030820','20041230','20060811']).

ctr_origin(k_cut,'E. Althaus',[]).

ctr_arguments(
 k_cut,
 ['K'-int,'NODES'-collection(index-int,succ-svar)]).

ctr_restrictions(
 k_cut,
 ['K'=1,
  'K'=<size('NODES'),
  required('NODES',[index,succ]),
  'NODES'\index>=1,
  'NODES'\index=<size('NODES'),
  distinct('NODES',index),
  'NODES'\succ=1,
  'NODES'\succ=<size('NODES')]).

ctr_example(
 k_cut,
 k_cut(3,
  [[[index-1,succ-{}],
    [index-2,succ-{3,5}],
    [index-3,succ-{5}],
    [index-4,succ-{}],
    [index-5,succ-{2,3}]])).

ctr_typical(k_cut,[size('NODES')>1]).

ctr_exchangeable(
 k_cut,
 [vals(['K'],int>=1),>dontcare,dontcare),
  items('NODES',all)]).

ctr_graph(
 k_cut,
 ['NODES'],
 2,
  ['CLIQUE'>>collection(nodes1,nodes2)],
  [nodes1\index=nodes2\index#\/]}
nodes2.index in_set nodes1.succ],
['NCC'>'K'],
[]).
B.189 k_disjoint

◊ **META-DATA:**

```prolog
ctr_date(k_disjoint,['20050816','20060811']).
ctr_origin(k_disjoint,'Derived from %c',[disjoint]).
ctr_types(k_disjoint,['VARIABLES'-collection(var-dvar)]).
ctr_arguments(k_disjoint,['SETS'-collection(set-'VARIABLES')]).
ctr_restrictions( k_disjoint, [required('VARIABLES',var), size('VARIABLES')>=1, required('SETS',set), size('SETS')>1]).
ctr_example( k_disjoint, k_disjoint([ [set-[[var-1],[var-9],[var-1],[var-5]]], [set- [var-2],[var-7],[var-7],[var-0],[var-6],[var-8]], [set-[[var-4],[var-4],[var-3]]]])).
ctr_typical(k_disjoint,[size('VARIABLES')>1]).
ctr_exchangeable( k_disjoint, [items('SETS',all), items('SETS'~set,all), vals(['VARIABLES'~var],int,=\=,dontcare,in), vals(['SETS'~set~var],int,=\=,all,dontcare)]).
ctr_graph( k_disjoint, [SETS'], 2, ['CLIQUE'([^)]]>collection(set1,set2)], disjoint(set1~set,set2~set), ['NARC'=size('SETS')*(size('SETS')-1)/2], []).
ctr_eval(k_disjoint,[reformulation(k_disjoint_r)]).
```
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

ctr_contractible(k_disjoint,[],'SETS',any).

k_disjoint_r(SETS) :-
    length(SETS,N),
    N>1,
    collection(SETS,[non_empty_col([dvar])]),
    get_attr1(SETS,VARS),
    k_disjoint1(VARS).

k_disjoint1([_25855]) :-
    !.

k_disjoint1([V1,V2|R]) :-
    k_disjoint2([V2|R],V1),
    k_disjoint1([V2|R]).

k_disjoint2([],_25852).

k_disjoint2([U|R],V) :-
    eval(disjoint(V,U)),
    k_disjoint2(R,V).
B.190 k\_same

\textbf{Meta-Data:}

ctr\_date(k\_same,['20050808', '20060811']).

ctr\_origin(k\_same, '``\cite{ElbassioniKatrielKutzMahajan05}''', []).

ctr\_types(k\_same, ['VARIABLES'-collection(var-dvar)]).

ctr\_arguments(k\_same, ['SETS'-collection(set-'VARIABLES')]).

ctr\_restrictions(
  k\_same,
  [required('VARIABLES', var),
   size('VARIABLES') >= 1,
   required('SETS', set),
   size('SETS') > 1,
   same\_size('SETS', set)]).

ctr\_example(
  k\_same,
  k\_same(
    [set-
      [[var-1], [var-9], [var-1], [var-5], [var-2], [var-1]],
      [set-
        [[var-9], [var-1], [var-1], [var-1], [var-2], [var-5]],
      [set-
        [[var-5], [var-2], [var-1], [var-1], [var-9], [var-1]]])).

ctr\_typical(k\_same, [size('VARIABLES') > 1]).

ctr\_exchangeable(
  k\_same,
  [items('SETS', all),
   items('SETS' `^` set, all),
   vals(['SETS' `^` set `^` var, int, =\_\_\_], all, dontcare)]).

ctr\_graph(
  k\_same,
  ['SETS'],
  2,
  ['PATH' `>>` collection(set1, set2)],
  [same(set1 `^` set, set2 `^` set)],
  ['NARC' = size('SETS') - 1],
  []).
ctr_eval(k_same, [reformulation(k_same_r)]).

ctr_contractible(k_same, [], ‘SETS’, any).

k_same_r(SETS) :-
  length(SETS, N),
  N>1,
  collection(SETS, [non_empty_col([dvar])]),
  get_attr1(SETS, VARS),
  k_same1(VARS).

k_same1([_29598]) :-
  !.

k_same1([V1, V2 | R]) :-
  eval(same(V1, V2)),
  k_same1([V2 | R]).
B.191  k_same_interval

◊ **META-DATA:**

```prolog
ctr_date(k_same_interval,['20050810','20060811']).
```

```prolog
ctr_origin(
    k_same_interval,
    Derived from %c and from %c.,
    [same_interval,k_same]).
```

```prolog
ctr_types(k_same_interval,['VARIABLES'-collection(var-dvar)]).
```

```prolog
ctr_arguments(
    k_same_interval,
    ['SETS'-collection(set-'VARIABLES'),'SIZE_INTERVAL'-int]).
```

```prolog
ctr_restrictions(
    k_same_interval,
    [required('VARIABLES',var),
     size('VARIABLES')>=1,
     required('SETS',set),
     size('SETS')>1,
     same_size('SETS',set),
     'SIZE_INTERVAL'>0]).
```

```prolog
ctr_example(
    k_same_interval,
    k_same_interval(
        [set-
         [[var-1],[var-1],[var-6],[var-0],[var-1],[var-7]],
         [set-
          [[var-8],[var-8],[var-0],[var-0],[var-1],[var-2]],
          [set-
           [[var-2],[var-1],[var-1],[var-2],[var-6],[var-6]]],
          3]).
```

```prolog
ctr_typical(
    k_same_interval,
    [size('VARIABLES')>1,'SIZE_INTERVAL'>1]).
```

```prolog
ctr_exchangeable(
    k_same_interval,
    [items('SETS',all),
     items('SETS'~set,all),
     vals(
```
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

['SETS'\`set\`var],
intervals('SIZE_INTERVAL'),
=,
dontcare,
dontcare]).

ctr_graph(
    k_same_interval,
    ['SETS'],
    2,
    ['PATH'\textgreater\textgreater collection(set1,set2)],
    [same_interval(set1\`set,set2\`set,'SIZE_INTERVAL')],
    ['NARC'=size('SETS')-1],
    []).

ctr_eval(k_same_interval,[reformulation(k_same_interval_r)]).

ctr_contractible(k_same_interval,[],'SETS',any).

k_same_interval_r(SETS,SIZE_INTERVAL) :-
    length(SETS,N),
    N>1,
    collection(SETS,[non_empty_col([dvar])]),
    integer(SIZE_INTERVAL),
    SIZE_INTERVAL>0,
    get_attr1(SETS,VARS),
    k_same_interval1(VARS,SIZE_INTERVAL).

k_same_interval1([-28384],-28383) :-
    !.

k_same_interval1([V1,V2|R],SIZE_INTERVAL) :-
    eval(same_interval(V1,V2,SIZE_INTERVAL)),
    k_same_interval1([V2|R],SIZE_INTERVAL).
B.192  k_same_modulo

◊ **META-DATA:**

```prolog
ctr_date(k_same_modulo,['20050810','20060811']).
```

```prolog
ctr_origin(
  k_same_modulo,
  Derived from %c and from %c.,
  [same_modulo,k_same]).
```

```prolog
ctr_types(k_same_modulo,['VARIABLES'-collection(var-dvar)]).
```

```prolog
ctr_arguments(
  k_same_modulo,
  ['SETS'-collection(set-'VARIABLES'),'M'-int]).
```

```prolog
ctr_restrictions(
  k_same_modulo,
  [required('VARIABLES',var),
   size('VARIABLES')>=1,
   required('SETS',set),
   size('SETS')>1,
   same_size('SETS',set),
   'M'>0]).
```

```prolog
ctr_example(
  k_same_modulo,
  k_same_modulo(
    [[set-
      [[var-1],[var-9],[var-1],[var-5],[var-2],[var-1]],
      [set-
      [[var-6],[var-4],[var-1],[var-1],[var-5],[var-5]],
      [set-
      [[var-1],[var-3],[var-4],[var-2],[var-8],[var-7]],
      3]).
```

```prolog
ctr_typical(k_same_modulo,[size('VARIABLES')>1,'M'>1]).
```

```prolog
ctr_exchangeable(
  k_same_modulo,
  [items('SETS',all),
   items('SETS''set',all),
   vals(['SETS''set''var',mod('M'),=,dontcare,dontcare])].
```

```prolog
ctr_graph(
```
k_same_modulo,
["SETS"],
2,
["PATH"\>collection(set1,set2)],
[same_modulo(set1\set,set2\set,\'M\')],
[\'NARC\'=size\(\'SETS\')-1],

ctr_eval(k_same_modulo,[reformulation(k_same_modulo_r)]).

ctr_contractible(k_same_modulo,[],"SETS",any).

k_same_modulo_r(SETS,M) :-
length(SETS,N),
N>1,
collection(SETS,[non_empty_col([dvar])]),
i\nteger(M),
M=\=0,
get_attr1(SETS,VARS),
k_same_modulo1(VARS,M).

k_same_modulo1([-27720],[-27719]) :- !.

k_same_modulo1([V1,V2|R],M) :-
eval(same_modulo(V1,V2,M)),
k_same_modulo1([V2|R],M).
B.193  \textbf{k\_same\_partition}

\textbf{\textsc{meta-data}}:

\begin{verbatim}
ctr_date(k_same_partition,['20050810','20060811']).

ctr_origin(
    k_same_partition,
    Derived from %c and from %c.,
    [same_partition,k_same]).

ctr_types(
    k_same_partition,
    ['VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int)]).

ctr_arguments(
    k_same_partition,
    ['SETS'-collection(set-'VARIABLES'),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    k_same_partition,
    [required('VARIABLES',var),
     size('VARIABLES')>=1,
     size('VALUES')>=1,
     required('VALUES',val),
     distinct('VALUES',val),
     required('SETS',set),
     size('SETS')>=1,
     same_size('SETS',set),
     required('PARTITIONS',p),
     size('PARTITIONS')>=2]).

ctr_example(
    k_same_partition,
    k_same_partition(  
        [[set-
            [[var-1],[var-2],[var-6],[var-3],[var-1],[var-2]]],
        [set-
            [[var-6],[var-6],[var-2],[var-3],[var-1],[var-3]]],
        [set-
            [[var-2],[var-2],[var-2],[var-1],[var-1],[var-3]]],
        [[p-[[val-1],[val-3]]],
        [p-[[val-4]]],
        [p-[[val-2],[val-6]]]])).
\end{verbatim}
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

```
ctr_typical(k_same_partition,[size('VARIABLES')>1]).

ctr_exchangeable(k_same_partition,
    [items('SETS',all),
     items('SETS'~set,all),
     items('PARTITIONS',all),
     items('PARTITIONS'~p,all),
     vals([SETS~set~var],
           part('PARTITIONS'),
           =,
           dontcare,
           dontcare)]).

ctr_graph(k_same_partition,
    ['SETS'],
    2,
    ['PATH'=>collection(set1,set2)],
    [same_partition(set1~set,set2~set,'PARTITIONS')],
    ['NARC'=size('SETS')-1],
    []).

ctr_eval(k_same_partition,[reformulation(k_same_partition_r)]).

ctr_contractible(k_same_partition,[],'SETS',any).

k_same_partition_r(SETS,PARTITIONS) :-
    length(SETS,N),
    N>1,
    collection(SETS,[non_empty_col([dvar])]),
    collection(PARTITIONS,[col_len_gteq(1,[int])]),
    length(PARTITIONS,P),
    P>1,
    get_attr1(SETS,VARS),
    k_same_partition1(VARS,PARTITIONS).

k_same_partition1([_29717],_29716) :-
    !.

k_same_partition1([V1,V2|R],PARTITIONS) :-
    eval(same_partition(V1,V2,PARTITIONS)),
    k_same_partition1([V2|R],PARTITIONS).
```
B.194  k_used_by

◊ Meta-Data:

ctr_date(k_used_by, [’20050814’, ’20060811’]).

ctr_origin(k_used_by, ’Derived from %c’, [used_by]).

ctr_types(k_used_by, [’VARIABLES’-collection(var-dvar)]).

ctr_arguments(k_used_by, [’SETS’-collection(set-’VARIABLES’)]).

ctr_restrictions(
  k_used_by,
  [required(’VARIABLES’, var),
   size(’VARIABLES’) >= 1,
   required(’SETS’, set),
   size(’SETS’) > 1,
   non_increasing_size(’SETS’, set)]).

ctr_example(
  k_used_by,
  k_used_by(
    [set-
      [set-
        [var-1],[var-9],[var-1],[var-5],[var-2],[var-1]],
        [var-9],[var-1],[var-1],[var-1],[var-2],[var-5]],
        [set-[[var-1],[var-1],[var-2],[var-5]]])).

ctr_typical(k_used_by, [size(’VARIABLES’) > 1]).

ctr_exchangeable(
  k_used_by,
  [items(’SETS’, all),
   items(’SETS’^set, all),
   vals([’SETS’^set^var], int, =\=, all, dontcare)]).

ctr_graph(
  k_used_by,
  [’SETS’],
  2,
  [’PATH’>>collection(set1, set2)],
  [used_by(set1^set, set2^set)],
  [’NARC’=size(’SETS’)-1],
  []).
ctr_eval(k_used_by,[reformulation(k_used_by_r)]).

ctr_contractible(k_used_by,[],’SETS’,any).

k_used_by_r(SETS) :-
    length(SETS,N),
    N>1,
    collection(SETS,[non_empty_col([dvar])]),
    get_attr1(SETS,VARS),
    k_used_by1(VARS).

k_used_by1([_28800]) :-
    !.

k_used_by1([V1,V2|R]) :-
    eval(used_by(V1,V2)),
    k_used_by1([V2|R]).
B.195  \texttt{k\_used\_by\_interval}

\begin{itemize}
  \item \textbf{Meta-Data:} \\
  \texttt{ctr\_date(k\_used\_by\_interval,\['20050814','20060811'\]).}

  \texttt{ctr\_origin(}
  \hspace{1em} \texttt{k\_used\_by\_interval,}
  \hspace{1em} \texttt{Derived from \%c and from \%c.,}
  \hspace{1em} \texttt{[used\_by\_interval,k\_used\_by]).}

  \texttt{ctr\_types(}
  \hspace{1em} \texttt{k\_used\_by\_interval,}
  \hspace{1em} \texttt{['VARIABLES'-collection(var-dvar)].}

  \texttt{ctr\_arguments(}
  \hspace{1em} \texttt{k\_used\_by\_interval,}
  \hspace{1em} \texttt{['SETS'-collection(set-'VARIABLES'),'SIZE\_INTERVAL'-int]).}

  \texttt{ctr\_restrictions(}
  \hspace{1em} \texttt{k\_used\_by\_interval,}
  \hspace{1em} \texttt{[required('VARIABLES',var),}
  \hspace{1em} \texttt{size('VARIABLES')>=1,}
  \hspace{1em} \texttt{required('SETS',set),}
  \hspace{1em} \texttt{size('SETS')>1,}
  \hspace{1em} \texttt{non\_increasing\_size('SETS',set),}
  \hspace{1em} \texttt{'SIZE\_INTERVAL'>0]).}

  \texttt{ctr\_example(}
  \hspace{1em} \texttt{k\_used\_by\_interval,}
  \hspace{1em} \texttt{k\_used\_by\_interval(}
  \hspace{1em} \hspace{1em} \texttt{[[set-}
  \hspace{1em} \hspace{1em} \hspace{1em} \texttt{[[var-1],[var-1],[var-1],[var-8],[var-6],[var-2]]],}
  \hspace{1em} \hspace{1em} \hspace{1em} \texttt{[set-[var-1],[var-0],[var-7],[var-7]]],}
  \hspace{1em} \hspace{1em} \hspace{1em} \texttt{[set-[var-1],[var-2]]]],}
  \hspace{1em} \hspace{1em} \hspace{1em} \texttt{3)).}

  \texttt{ctr\_typical(}
  \hspace{1em} \texttt{k\_used\_by\_interval,}
  \hspace{1em} \texttt{[size('VARIABLES')>1,'SIZE\_INTERVAL'>0]).}

  \texttt{ctr\_exchangeable(}
  \hspace{1em} \texttt{k\_used\_by\_interval,}
  \hspace{1em} \texttt{[items('SETS',all),}
  \hspace{1em} \hspace{1em} \texttt{items('SETS'\^{set},all),}
  \hspace{1em} \hspace{1em} \hspace{1em} \texttt{vals(}
  \hspace{1em} \hspace{1em} \hspace{1em} \texttt{null)]).}
\end{itemize}
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

["SETS"\^set\^var],
intervals("SIZE_INTERVAL"),
=,  
dontcare,
dontcare])}.

ctr_graph(
k_used_by_interval,
["SETS"],
2,  
["PATH"\rangle\rightarrow\text{collection(set1,set2)}],
[used_by_interval(set1\^set,set2\^set,SIZE_INTERVAL')],
["NARC"=\text{size}(SETS')-1],
[]).

ctr_eval(
  k_used_by_interval,
  [reformulation(k_used_by_interval_r)]).

ctr_contractible(k_used_by_interval,[],"SETS",any).

k_used_by_interval_r(SETS,SIZE_INTERVAL) :-
  length(SETS,N),
  N>1,
  collection(SETS,[\text{non_empty_col}([dvar])]),
  integer(SIZE_INTERVAL),
  SIZE_INTERVAL>0,
  \text{get_attr1}(SETS,VARS),
  k_used_by_interval1(VARS,SIZE_INTERVAL).

k_used_by_interval1([_27341],_27340) :-
  !.

k_used_by_interval1([V1,V2|R],SIZE_INTERVAL) :-
  eval(used_by_interval(V1,V2,SIZE_INTERVAL)),
  k_used_by_interval1([V2|R],SIZE_INTERVAL).
B.196 k_used_by_modulo

◊ **META-DATA:**

```
ctr_date(k_used_by_modulo, ['20050814', '20060811']).
```

```
ctr_origin(
    k_used_by_modulo,
    Derived from %c and from %c,
    [used_by_modulo,k_used_by]).
```

```
ctr_types(k_used_by_modulo, ['VARIABLES'-collection(var-dvar)]).
```

```
ctr_arguments(
    k_used_by_modulo,
    ['SETS'-collection(set-'VARIABLES')],'M'-int]).
```

```
ctr_restrictions(
    k_used_by_modulo,
    [required('VARIABLES',var),
     size('VARIABLES')>=1,
     required('SETS',set),
     size('SETS')>1,
     non_increasing_size('SETS',set),
     'M'>0]).
```

```
ctr_example(
    k_used_by_modulo,
    k_used_by_modulo(
        [[set-
            [[var-1],[var-9],[var-4],[var-5],[var-2],[var-1]],
            [set-[[var-7],[var-1],[var-2],[var-5]]],
            [set-[[var-1],[var-1]]],
        3]).
```

```
ctr_typical(k_used_by_modulo, [size('VARIABLES')>1,'M'>1]).
```

```
ctr_exchangeable(
    k_used_by_modulo,
    [items('SETS',all),
     items('SETS`set',all),
     vals(['SETS`set`var',mod('M'),=,dontcare,dontcare])].
```

```
ctr_graph(
    k_used_by_modulo,
    ['SETS'],
    ```


```
2,
['PATH'>>collection(set1,set2)],
[used_by_modulo(set1\^set, set2\^set,'M')],
['NARC'=size('SETS')-1],
[]).

ctr_eval(K_used_by_modulo, [reformulation(K_used_by_modulo_r)]).

ctr_contractible(K_used_by_modulo, [], 'SETS', any).

K_used_by_modulo_r (SETS, M) :-
  length (SETS, N),
  N>1,
  collection (SETS, [non_empty_col([dvar])]),
  integer (M),
  M\=0,
  get_attr1 (SETS, VARS),
  K_used_by_modulo1 (VAR1S, M).

K_used_by_modulo1 (\[26982,26981\]) :-
  !.

K_used_by_modulo1 (\[V1,V2|R\], M) :-
  eval (used_by_modulo (V1, V2, M)),
  K_used_by_modulo1 (\[V2|R\], M).
```
B.197  k_used_by_partition

◊ Meta-Data:

ctr_date(k_used_by_partition,[’20050814’,’20060811’]).

ctr_origin(
    k_used_by_partition,
    Derived from %c and from %c.,
    [used_by_partition,k_used_by]).

ctr_types(
    k_used_by_partition,
    [’VARIABLES’-collection(var-dvar),
     ’VALUES’-collection(val-int)]).

ctr_arguments(
    k_used_by_partition,
    [’SETS’-collection(set-’VARIABLES’),
     ’PARTITIONS’-collection(p-’VALUES’)]).

ctr_restrictions(
    k_used_by_partition,
    [required(’VARIABLES’,var),
     size(’VARIABLES’)>=1,
     size(’VALUES’)>=1,
     required(’VALUES’,val),
     distinct(’VALUES’,val),
     required(’SETS’,set),
     size(’SETS’)1,
     non_increasing_size(’SETS’,set),
     required(’PARTITIONS’,p),
     size(’PARTITIONS’)2].

ctr_example(
    k_used_by_partition,
    k_used_by_partition([set-
        [[var-1],[var-9],[var-1],[var-6],[var-2],[var-3]]],
        [set-[[var-1],[var-3],[var-6],[var-6]]],
        [set-[[var-2],[var-2]]],
        [[p-[[val-1],[val-3]]],
         [p-[[val-4]]],
         [p-[[val-2],[val-6]]])].

ctr_typical(k_used_by_partition,[size(’VARIABLES’)1]).
ctr_exchangeable(
    k_used_by_partition,
    [items('SETS',all),
     items('SETS'\set,all),
     items('PARTITIONS',all),
     items('PARTITIONS'\p,all),
     vals(
         ['SETS'\set\var],
         part('PARTITIONS'),
         =,  
dontcare,
         dontcare))].

ctr_graph(
    k_used_by_partition,
    ['SETS'],
    2,
    ['PATH'\>collection(set1,set2)],
    [used_by_partition(set1\set,set2\set,'PARTITIONS')],
    ['NARC'=size('SETS')-1],
    []).

ctr_eval(
    k_used_by_partition,
    [reformulation(k_used_by_partition_r)]).

ctr_contractible(k_used_by_partition,[],'SETS',any).

k_used_by_partition_r(SETS,PARTITIONS) :-
    length(SETS,N),
    N>1,
    collection(SETS,[non_empty_col([dvar])]),
    collection(PARTITIONS,[col_len_gteq(1,[int])]),
    length(PARTITIONS,P),
    P>1,
    get_attr1(SETS,VARS),
    k_used_by_partition1(VARS,PARTITIONS).

k_used_by_partition1([_28487],_28486) :- !.

k_used_by_partition1([V1,V2\R],PARTITIONS) :-
    eval(used_by_partition(V1,V2,PARTITIONS)),
    k_used_by_partition1([V2\R],PARTITIONS).
B.198 length_first_sequence

◊ **META-DATA:**

```prolog
ctr_date(length_first_sequence,['20081123']).
```

```prolog
ctr_origin(
    length_first_sequence,
    Inspired by %c,
    [stretch_path]).
```

```prolog
ctr_arguments( 
    length_first_sequence, 
    ['LEN'-dvar,'VARIABLES'-collection(var-dvar)]).
```

```prolog
ctr_restrictions( 
    length_first_sequence, 
    ['LEN'>=0, 
    'LEN'=<size('VARIABLES'), 
    required('VARIABLES',var)]).
```

```prolog
ctr_example( 
    length_first_sequence, 
    length_first_sequence( 
    3, 
    [[var-4],[var-4],[var-4],[var-5],[var-5],[var-4]]). 
```

```prolog
ctr_typical( 
    length_first_sequence, 
    ['LEN'<size('VARIABLES'),size('VARIABLES')>1]).
```

```prolog
ctr_exchangeable( 
    length_first_sequence, 
    [vals(['VARIABLES'\^var],int,\=,all,dontcare)]).
```

```prolog
ctr_eval( 
    length_first_sequence, 
    [reformulation(length_first_sequence_r), 
    automaton(length_first_sequence_a)]).
```

```prolog
length_first_sequence_r(LEN,VARIABLES) :- 
    collection(VARIABLES,[dvar]), 
    length(VARIABLES,N), 
    check_type(dvar(0,N),LEN), 
    get_attr1(VARIABLES,VARS), 
    ( N=0 ->
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

LEN#=0 ; N=1 -> LEN#=1 ; reverse(VARS,RVARS), length_first_sequence1(RVARS,_15813,TERM), call(LEN#=TERM).

length_first_sequence1([_15740],1,1) :- !.

length_first_sequence1([VAR1,VAR2|R],AND1,AND1+S) :- length_first_sequence1([VAR2|R],AND2,S), B12#<=>VAR1#=VAR2, AND1#<=>AND2#/\B12.

length_first_sequence_a(1,0,[]) :- !.

length_first_sequence_a(0,0,[]) :- !, fail.

length_first_sequence_a(1,1,[_15740]) :- !.

length_first_sequence_a(0,1,[_15740]) :- !, fail.

length_first_sequence_a(FLAG,LEN,VARIABLES) :- collection(VARIABLES,[dvar]), length(VARIABLES,N), check_type(dvar(0,N),LEN), length_first_sequence_signature(VARIABLES,SIGNATURE), automaton( SIGNATURE, _17056, SIGNATURE, [source(s),sink(s),sink(t)], [arc(s,0,t)], arc(s,1,s,[C+1]), arc(t,0,t), arc(t,1,t)], [C], [1],
length_first_sequence_signature([],[]).
length_first_sequence_signature([_15739],[]) :- !.
length_first_sequence_signature(
  [[var-VAR1],[var-VAR2]|VARs],
  [S|Ss]) :-
  S in 0..1,
  VAR1#=VAR2#<=>S,
  length_first_sequence_signature([[var-VAR2]|VARs],Ss).
B.199 length_last_sequence

◊ **Meta-Data:**

```prolog
ctr_date(length_last_sequence, ['20081123']).

ctr_origin(
    length_last_sequence,
    Inspired by %c, [stretch_path]).

ctr_arguments(
    length_last_sequence,
    ['LEN'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    length_last_sequence,
    ['LEN'>=0,
     'LEN'=<size('VARIABLES'),
     required('VARIABLES',var)]).

ctr_example(
    length_last_sequence,
    length_last_sequence(1,
             [[var-4],[var-4],[var-4],[var-5],[var-5],[var-4]])).

ctr_typical(
    length_last_sequence,
    ['LEN'<size('VARIABLES'),size('VARIABLES')>1]).

ctr_exchangeable(
    length_last_sequence,
    [vals(['VARIABLES''var'],int,\=,all,dontcare)]).

ctr_eval(
    length_last_sequence,
    [reformulation(length_last_sequence_r),
     automaton(length_last_sequence_a)]).

length_last_sequence_r(LEN,VARIABLES) :-
    check_type(dvar,LEN),
    collection(VARIABLES,[dvar]),
    get_attr1(VARIABLES,VARS),
    length(VARIABLES,N),
    ( N=0 ->
      ... 
    ).
```

```prolog
ctr_eval(
    length_last_sequence,
    [reformulation(length_last_sequence_r),
     automaton(length_last_sequence_a)]).

length_last_sequence_r(LEN,VARIABLES) :-
    check_type(dvar,LEN),
    collection(VARIABLES,[dvar]),
    get_attr1(VARIABLES,VARS),
    length(VARIABLES,N),
    ( N=0 ->
      ... 
    ).
```
LEN#=0
; N=1 ->
LEN#=1

length_last_sequence1(VARS,[_15726],TERM),
call(LEN#=TERM)
).

length_last_sequence1([_15726],1,1) :-
!.

length_last_sequence1([VAR1,VAR2|R],AND1,AND1+S) :-
length_last_sequence1([VAR2|R],AND2,S),
B12#<=VAR1#=VAR2, AND1#<=AND2# \ B12.

length_last_sequence_a(1,0,[]) :-
!.

length_last_sequence_a(0,0,[]) :-
!, fail.

length_last_sequence_a(1,1,[_15726]) :-
!.

length_last_sequence_a(0,1,[_15726]) :-
!, fail.

length_last_sequence_a(FLAG,LEN,VARIABLES) :-
collection(VARIABLES,[dvar]),
length(VARIABLES,N),
check_type(dvar(0,N),LEN),
length_last_sequence_signature(VARIABLES,SIGNAL),
automaton(
    SIGNAL,
    _17006,
    SIGNAL,
    [source(s),sink(s)],
    [arc(s,0,s,[1]),arc(s,1,s,[C+1])],
    [C],
    [1],
    [COUNT]),
COUNT#=LEN#<=>FLAG.

length_last_sequence_signature([],[]).
length_last_sequence_signature([\_15725],[\]) :-
  !.

length_last_sequence_signature([\[\text{\texttt{var-VAR1}},\text{\texttt{var-VAR2}}]|\text{\texttt{VARs}},
  \[\text{\texttt{S}}\text{\texttt{Ss}}]) :-
  \text{\texttt{S}} \text{\texttt{in 0..1,}}
  \text{\texttt{VAR1\#=VAR2\#<=\texttt{S,}}}
  length_last_sequence_signature([\[\text{\texttt{var-VAR2}}]\text{\texttt{|VARs}},\text{\texttt{Ss}}].
B.200  leq

◊ **META-DATA:**

ctr_predefined(leq).

ctr_date(leq,['20070821']).

ctr_origin(leq,'Arithmetic.',[]).

ctr_synonyms(leq,[rel,xlteqy]).

ctr_arguments(leq,['VAR1'-dvar,'VAR2'-dvar]).

ctr_example(leq,leq(1,8)).

ctr_typical(leq,['VAR1'<'VAR2']).

ctr_exchangeable(
    leq,
    [vals(['VAR1'],int(=<('VAR2')),=\,all,dontcare),
     vals(['VAR2'],int(>=('VAR1')),=\,all,dontcare)].

ctr_eval(leq,[builtin(leq_b)]).

leq_b(VAR1,VAR2) :-
    check_type(dvar,VAR1),
    check_type(dvar,VAR2),
    VAR1#=<VAR2.
B.201 leq_cst

◊ **Meta-Data:**

ctr_predefined(leq_cst).

ctr_date(leq_cst,['20090912']).

ctr_origin(leq_cst,'Arithmetic.',[]).

ctr_arguments(leq_cst,['VAR1'-dvar,'VAR2'-dvar,'CST2'-int]).

ctr_example(leq_cst,leq_cst(5,2,4)).

ctr_typical(leq_cst,['CST2'=\=0,'VAR1'<'VAR2'+CST2']).

ctr_exchangeable(
    leq_cst,
    [args([['VAR1'],['VAR2','CST2']]),
    vals(['VAR1'],int(=\=('VAR2'+CST2)),\=,all,dontcare),
    vals(['VAR2'],int(>\=('VAR1'-CST2)),\=,all,dontcare),
    vals(['CST2'],int(\>=('VAR1'-VAR2)),\=,all,dontcare)].

ctr_eval(leq_cst,[builtin(leq_cst_b)]).

leq_cst_b(VAR1,VAR2,CST2) :-
    check_type(dvar,VAR1),
    check_type(dvar,VAR2),
    check_type(int,CST2),
    VAR1\=\=VAR2+CST2.
B.202 lex2

◊ Meta-Data:

ctr_predefined(lex2).

ctr_date(lex2,[‘20031008’,’20040530’,’20060811’]).

ctr_origin(lex2, \cite{FlenerFrischHnichKiziltanMiguelPearsonWalsh02}, []).

ctr_synonyms(lex2,[double_lex,row_and_column_lex]).

ctr_types(lex2,[‘VECTOR’-collection(var-dvar)]).

ctr_arguments(lex2,[‘MATRIX’-collection(vec-‘VECTOR’)]).

ctr_restrictions(lex2,
[size(‘VECTOR’)>=1,
 required(‘VECTOR’,var),
 required(‘MATRIX’,vec),
 same_size(‘MATRIX’,vec)]).

ctr_example(lex2,
lex2(lex2(
 [vec-[[var-2],[var-2],[var-3]],
  [vec-[[var-2],[var-3],[var-1]]])).

ctr_typical(lex2,[size(‘VECTOR’)\geq 1,size(‘MATRIX’)\geq 1]).

ctr_exchangeable(lex2,[translate([‘MATRIX’\^vec\^var])]).

ctr_eval(lex2,[reformulation(lex2_r)]).

lex2_r(MATRIX) :-
 collection(MATRIX,[col([dvar])]),
 same_size(MATRIX),
 get_attr11(MATRIX,MAT),
 lex_chain(MAT,[op(#=<)]),
 transpose(MAT,TMAT),
 lex_chain(TMAT,[op(#=<)]).
B.203  **lex_alldifferent**  

**Meta-Data:**

```prolog
ctr_date(
    lex_alldifferent,
    ['20030820','20040530','20051008','20060811','20111102']).

ctr_origin(lex_alldifferent,'J. Pearson',[]).

ctr_synonyms(
    lex_alldifferent,
    [lex_alldiff,
     lex_alldistinct,
     alldiff_on_tuples,
     alldifferent_on_tuples,
     alldistinct_on_tuples]).

ctr_types(lex_alldifferent,['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    lex_alldifferent,
    ['VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    lex_alldifferent,
    [size('VECTOR')>=1,
     required('VECTOR',var),
     required('VECTORS',vec),
     same_size('VECTORS',vec)]).

ctr_example(
    lex_alldifferent,
    lex_alldifferent(
        [[vec-[[var-5],[var-2],[var-3]]],
         [vec-[[var-5],[var-2],[var-6]]],
         [vec-[[var-5],[var-3],[var-3]]])).

ctr_typical(
    lex_alldifferent,
    [size('VECTOR')>1,size('VECTORS')>1]).

ctr_exchangeable(
    lex_alldifferent,
    [items('VECTORS',all),
     items_sync('VECTORS' vec,all),
     items_sync('VECTORS' vec,all),
     items_sync('VECTORS' vec,all)]).
```

vals([\text{\texttt{VECTORS}}^{\text{\texttt{vec}}},\text{int},=\text{\_\_},\text{all},\text{dontcare}]).

ctr\_graph(
  \text{\texttt{lex\_alldifferent}},
  [\text{\texttt{VECTORS}}],
  2,
  [\text{\texttt{CLIQUE}}(<)\gg\text{collection}(\text{\texttt{vectors1}},\text{\texttt{vectors2}})],
  [\text{\texttt{lex\_different}}(\text{\texttt{vectors1}}^{\text{\texttt{vec}}},\text{\texttt{vectors2}}^{\text{\texttt{vec}}})],
  [\text{\texttt{NARC}}=\text{size}(\text{\texttt{VECTORS}})\ast(\text{size}(\text{\texttt{VECTORS}})-1)/2],
  []).

ctr\_eval(
  \text{\texttt{lex\_alldifferent}},
  [\text{\texttt{checker}}(\text{\texttt{lex\_alldifferent\_c}}),
   \text{\texttt{reformulation}}(\text{\texttt{lex\_alldifferent\_r}})].

ctr\_contractible(\text{\texttt{lex\_alldifferent}},[],\text{\texttt{VECTORS}},\text{any}).

ctr\_extensible(\text{\texttt{lex\_alldifferent}},[],\text{\texttt{VECTORS}}^{\text{\texttt{vec}}},\text{any}).

\text{\texttt{lex\_alldifferent\_c}}(\text{\texttt{VECTORS}}) :-
  \text{\texttt{collection}}(\text{\texttt{VECTORS}},[\text{\texttt{col}}([\text{\texttt{int}}])]),
  \text{\texttt{length}}(\text{\texttt{VECTORS}},L),
  \text{\texttt{sort}}(\text{\texttt{VECTORS}},\text{\texttt{SVECTORS}}),
  \text{\texttt{length}}(\text{\texttt{SVECTORS}},L).

\text{\texttt{lex\_alldifferent\_r}}(\text{\texttt{VECTORS}}) :-
  \text{\texttt{collection}}(\text{\texttt{VECTORS}},[\text{\texttt{col}}([\text{\texttt{dvar}}])]),
  \text{\texttt{lex\_alldifferent\_1}}(\text{\texttt{VECTORS}}).

\text{\texttt{lex\_alldifferent\_1}}([]).

\text{\texttt{lex\_alldifferent\_1}}([\text{\_\_\_34592-VECTOR}|\text{\_\_\_34584}]) :-
  \text{\texttt{length}}(\text{\texttt{R}},\text{\_\_\_34584}),
  \text{\texttt{lex\_alldifferent\_2}}(\text{\texttt{R}},\text{\_\_\_34584}).

\text{\texttt{lex\_alldifferent\_2}}([\text{\_\_\_34593-VECTOR}|\text{\_\_\_34584}],\text{\texttt{VECTOR}}) :-
  \text{\texttt{eval}}(\text{\texttt{lex\_different}}(\text{\texttt{VECTOR}},\text{\texttt{VECTORi}})),
  \text{\texttt{lex\_alldifferent\_2}}(\text{\texttt{R}},\text{\texttt{VECTOR}}).
B.204 lex_between

◊ **Meta-Data:**

ctr_date(lex_between,['20030820','20040530','20060811']).

ctr_origin(lex_between,'\cite{BeldiceanuCarlsson02c}',[]).

ctr_synonyms(lex_between,[between]).

ctr_arguments(
    lex_between,
    ['LOWER_BOUND'-collection(var-int),
     'VECTOR'-collection(var-dvar),
     'UPPER_BOUND'-collection(var-int)]).

ctr_restrictions(
    lex_between,
    [required('LOWER_BOUND',var),
     required('VECTOR',var),
     required('UPPER_BOUND',var),
     size('LOWER_BOUND')=size('VECTOR'),
     size('UPPER_BOUND')=size('VECTOR'),
     lex_lesseq('LOWER_BOUND','VECTOR'),
     lex_lesseq('VECTOR','UPPER_BOUND')]).

ctr_example(
    lex_between,
    lex_between(%
    [[var-5],[var-2],[var-3],[var-9]],
    [[var-5],[var-2],[var-6],[var-2]],
    [[var-5],[var-2],[var-6],[var-3]])).}

ctr_typical(
    lex_between,
    [size('LOWER_BOUND')>1,
     lex_lesseq('LOWER_BOUND','UPPER_BOUND')]).

ctr_exchangeable(
    lex_between,
    [vals(['LOWER_BOUND'\var],int,>,dontcare,dontcare),
     vals(['UPPER_BOUND'\var],int,<,dontcare,dontcare)]).

ctr_eval(
    lex_between,
    [reformulation(lex_between_r),automaton(lex_between_a)]).
ctr_contractible(
    lex_between,
    [],
    ['LOWER_BOUND', 'VECTOR', 'UPPER_BOUND'],
    suffix).

lex_between_r(LOWER_BOUND, VECTOR, UPPER_BOUND) :-
    collection(LOWER_BOUND, [int]),
    collection(VECTOR, [dvar]),
    collection(UPPER_BOUND, [int]),
    length(LOWER_BOUND, LB),
    length(VECTOR, LV),
    length(UPPER_BOUND, LU),
    LB = LV,
    LU = LV,
    eval(lex_lesseq(LOWER_BOUND, VECTOR)),
    eval(lex_lesseq(VECTOR, UPPER_BOUND)).

lex_between_a(FLAG, LOWER_BOUND, VECTOR, UPPER_BOUND) :-
    collection(LOWER_BOUND, [int]),
    collection(VECTOR, [dvar]),
    collection(UPPER_BOUND, [int]),
    length(LOWER_BOUND, LB),
    length(VECTOR, LV),
    length(UPPER_BOUND, LU),
    LB = LV,
    LU = LV,
    lex_between_signature(
        LOWER_BOUND,
        VECTOR,
        UPPER_BOUND,
        SIGNATURE),
    AUTOMATON =
    automaton(
        SIGNATURE,
        _27016,
        SIGNATURE,
        [source(s), sink(a), sink(b), sink(s), sink(t)],
        [arc(s, 4, s),
         arc(s, 0, t),
         arc(s, 3, a),
         arc(s, 1, b),
         arc(a, 3, a),
         arc(a, 4, a),
         arc(a, 5, a),
arc(a,0,t),
arc(a,1,t),
arc(a,2,t),
arc(b,1,b),
arc(b,4,b),
arc(b,7,b),
arc(b,0,t),
arc(b,3,t),
arct(b,6,t),
arct(t,0,t),
arct(t,1,t),
arct(t,2,t),
arct(t,3,t),
arct(t,4,t),
arct(t,5,t),
arct(t,6,t),
arct(t,7,t),
arct(t,8,t)],
,[],[],[]),
automaton_bool(FLAG,[0,1,2,3,4,5,6,7,8],AUTOMATON).

lex_between_signature([],[],[],[]).

lex_between_signature(
 [[var-A1]|As],
 [[var-X1]|Xs],
 [[var-B1]|Bs],
 [L1|Ls]) :-
 Adown is A1-1,
 Aup is A1+1,
 Bdown is B1-1,
 Bup is B1+1,
 ( A1+1<B1 ->
 case( X-L,
 [X1-L1],
 [node(-1,
 X,
 [(inf..Adown)-6,
 (A1..A1)-3,
 (Aup..Bdown)-0,
 (B1..B1)-1,
 (Bup..sup)-2)]),

APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE
\begin{verbatim}
node(0,L,[0..0]),
node(1,L,[1..1]),
node(2,L,[2..2]),
node(3,L,[3..3]),
node(6,L,[6..6]))

; A1 < B1 ->
  case(
    X-L, [X1-L1],
    [node(
      -1, X,
      [(inf..Adown)-6, (A1..A1)-3, (B1..B1)-1, (Bup..sup)-2]),
      node(1,L,[1..1]), node(2,L,[2..2]), node(3,L,[3..3]), node(6,L,[6..6]))
  )

; A1 =:= B1 ->
  case(
    X-L, [X1-L1],
    [node(
      -1, X,
      [(inf..Adown)-6, (A1..A1)-4, (Aup..sup)-2]),
      node(2,L,[2..2]), node(4,L,[4..4]), node(6,L,[6..6]))
  )

; A1 =:= B1+1 ->
  case(
    X-L, [X1-L1],
    [node(
      -1, X,
      [(inf..Bdown)-6, (B1..B1)-7, (A1..A1)-5, (Aup..sup)-2]),
      node(2,L,[2..2]), node(5,L,[5..5]),
    )
  )
\end{verbatim}
node(6,L,[6..6]),
node(7,L,[7..7]))
; A1>B1 ->
case(
  X-L,
  [X1-L1],
  [node(
    1,
    X,
    [(inf..Bdown)-6,
      (B1..B1)-7,
      (Bup..Adown)-8,
      (A1..A1)-5,
      (Aup..sup)-2]),
    node(2,L,[2..2]),
    node(5,L,[5..5]),
    node(6,L,[6..6]),
    node(7,L,[7..7]),
    node(8,L,[8..8])])
),
lex_between_signature(As,Xs,Bs,Ls).
B.205 lex_chain_less

◊ META-DATA:

ctr_date(
        lex_chain_less,
        ['20030820','20040530','20060811','20090116']).

ctr_origin(lex_chain_less,'\cite{BeldiceanuCarlsson02c}',[]).

ctr_usual_name(lex_chain_less,lex_chain).

ctr_types(lex_chain_less,['VECTOR'-collection(var-dvar)]).

ctr_arguments(
        lex_chain_less,
        ['VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
        lex_chain_less,
        [size('VECTOR')>=1,
         required('VECTOR',var),
         required('VECTORS',vec),
         same_size('VECTORS',vec)]).

ctr_example(
        lex_chain_less,
        lex_chain_less(
            [[vec-[[var-5],[var-2],[var-3],[var-9]]],
             [vec-[[var-5],[var-2],[var-6],[var-2]]],
             [vec-[[var-5],[var-2],[var-6],[var-3]]]]).

ctr_typical(
        lex_chain_less,
        [size('VECTOR')>1,size('VECTORS')>1]).

ctr_graph(
        lex_chain_less,
        ['VECTORS'],
        2,
        ['PATH'>collection(vectors1,vectors2)],
        [lex_less(vectors1``vec,vectors2``vec),
         ['NARC'=size('VECTORS')-1],
         []).

ctr_eval(lex_chain_less,[builtin(lex_chain_less_b)]).
ctr_contractible(lex_chain_less,[],'VECTORS',any).

ctr_extensible(lex_chain_less,[],'VECTORS'\^vec,suffix).

lex_chain_less_b(VECTORS) :-
collection(VECTORS,[col([dvar])]),
same_size(VECTORS),
get_attr11(VECTORS,VECTS),
lex_chain(VECTS,[op(#<)]).
B.206  lex_chain_lesseq

◊ Meta-Data:

\begin{verbatim}
ctr_date(
    lex_chain_lesseq,
    ['20030820','20040530','20060811','20090116']).

ctr_origin(lex_chain_lesseq, '\cite{BeldiceanuCarlsson02c}', []).

ctr_usual_name(lex_chain_lesseq, lex_chain).

ctr_types(lex_chain_lesseq, ['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    lex_chain_lesseq, ['VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    lex_chain_lesseq, 
    [size('VECTOR')>=1, 
    required('VECTOR', var), 
    required('VECTORS', vec), 
    same_size('VECTORS', vec)]).

ctr_example(
    lex_chain_lesseq, 
    lex_chain_lesseq( 
        [[vec-[[var-5],[var-2],[var-3],[var-9]]], 
        [vec-[[var-5],[var-2],[var-6],[var-2]]], 
        [vec-[[var-5],[var-2],[var-6],[var-2]]]]).

ctr_typical(
    lex_chain_lesseq, 
    [size('VECTOR')>1, size('VECTORS')>1]).

ctr_graph(
    lex_chain_lesseq, 
    ['VECTORS'], 2, 
    ['PATH'>>collection(vectors1,vectors2)], 
    [lex_lesseq(vectors1`vec,vectors2`vec), 
    ['NARC'=size('VECTORS')-1], 
    []].

ctr_eval(lex_chain_lesseq, [builtin(lex_chain_lesseq_b)]).
\end{verbatim}
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

ctr_contractible(lex_chain_lesseq,[],'VECTORS',any).
ctr_contractible(lex_chain_lesseq,[],'VECTORS'\textasciitilde vec,suffix).

\texttt{lex\_chain\_lesseq\_b(VECTORS) :-}
\begin{itemize}
  \item \texttt{collection(VECTORS,[col([dvar]))},
  \item \texttt{same\_size(VECTORS)},
  \item \texttt{get\_attr11(VECTORS,VECTS)},
  \item \texttt{lex\_chain(VECTS,[op(#=<)])}.
\end{itemize}
B.207  lex_different

◊ META-DATA:

ctr_date(lex_different,['20030820','20040530']).

ctr_origin(
    lex_different,
    Used for defining %c.,
    [lex_alldifferent]).

ctr_synonyms(lex_different,[different,diff]).

ctr_arguments(
    lex_different,
    ['VECTOR1'-collection(var-dvar),
    'VECTOR2'-collection(var-dvar)]).

ctr_restrictions(
    lex_different,
    [required('VECTOR1',var),
    required('VECTOR2',var),
    size('VECTOR1')>0,
    size('VECTOR1')=size('VECTOR2')]).

ctr_example(
    lex_different,
    lex_different(
        [[var-5],[var-2],[var-7],[var-1]],
        [[var-5],[var-3],[var-7],[var-1]])).

ctr_typical(
    lex_different,
    [size('VECTOR1')>1,
    range('VECTOR1'°var)>1,
    range('VECTOR2'°var)>1]).

ctr_exchangeable(
    lex_different,
    [args([['VECTOR1', 'VECTOR2']]),
    items_sync('VECTOR1', 'VECTOR2', all)]).

ctr_graph(
    lex_different,
    ['VECTOR1', 'VECTOR2'],
    2,
[‘PRODUCT’ (=)>>collection(vector1,vector2)],
[vector1^var=\=vector2^var],
[‘NARC’>=1],
[]).

ctr_eval(
    lex_different,
    [reformulation(lex_different_r),
    automaton(lex_different_a)]).

ctr_extensible(lex_different,[],[‘VECTOR1’,'VECTOR2’],any).

lex_different_r(VECTOR1,VECTOR2) :-
    collection(VECTOR1,[dvar]),
    collection(VECTOR2,[dvar]),
    length(VECTOR1,L1),
    L1>0,
    length(VECTOR2,L2),
    L1=L2,
    get_attr1(VECTOR1,VECT1),
    get_attr1(VECTOR2,VECT2),
    lex_different1(VECT1,VECT2,Term),
    call(Term).

lex_different1([],[],0).

lex_different1([V1|R1],[V2|R2],V1#\=V2#/T) :-
    lex_different1(R1,R2,T).

lex_different_a(FLAG,VECTOR1,VECTOR2) :-
    collection(VECTOR1,[dvar]),
    collection(VECTOR2,[dvar]),
    length(VECTOR1,L1),
    L1>0,
    length(VECTOR2,L2),
    L1=L2,
    lex_different_signature(VECTOR1,VECTOR2,SIGNATURE),
    AUTOMATON=
    automaton(
        SIGNATURE,
        _34223,
        SIGNATURE,
        [source(s),sink(t)],
        [arc(s,1,s),arc(s,0,t),arc(t,0,t),arc(t,1,t)],
        [],
        [],
automaton_bool(FLAG,[0,1],AUTOMATON).

lex_different_signature([],[],[]).

lex_different_signature([[\text{var-VAR1}]|Xs],[[\text{var-VAR2}]|Ys],[S|Ss]) :-
  \text{VAR1}#=\text{VAR2}#<=>S,
  lex_different_signature(Xs,Ys,Ss).

APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.208  lex_equal

◊ Meta-Data:

ctr_date(lex_equal, ['20081220']).

ctr_origin(
    lex_equal,
    Initially introduced for defining %c, [nvector]).

ctr_synonyms(lex_equal, [equal, eq]).

ctr_arguments(
    lex_equal,
    ['VECTOR1'-collection(var-dvar),
     'VECTOR2'-collection(var-dvar)]).

ctr_restrictions(
    lex_equal,
    [required('VECTOR1', var),
     required('VECTOR2', var),
     size('VECTOR1')=size('VECTOR2')]).

ctr_example(
    lex_equal,
    lex_equal(
        [[var-1],[var-9],[var-1],[var-5]],
        [[var-1],[var-9],[var-1],[var-5]]).

ctr_typical(
    lex_equal,
    [size('VECTOR1')>1,
     range('VECTOR1'~var)>1,
     range('VECTOR2'~var)>1]).

ctr_exchangeable(
    lex_equal,
    [args([['VECTOR1','VECTOR2']]),
     items_sync('VECTOR1','VECTOR2',all)]).

ctr_graph(
    lex_equal,
    ['VECTOR1','VECTOR2'],
    2,
    ['PRODUCT' (=))>collection(vector1,vector2)],
[vector1\textsuperscript{\textasciitilde}var=vector2\textsuperscript{\textasciitilde}var],
['\textsc{NARC}=\text{size('VECTOR1')}'],
['\textsc{ACYCLIC}','\textsc{BIPARTITE}',\textsc{NO_LOOP}]).

\texttt{ctr\_eval(}
\texttt{\quad lex\_equal,}
\texttt{\quad [reformulation(lex\_equal\_r),automaton(lex\_equal\_a)])}.

\texttt{ctr\_contractible(lex\_equal,\[],['VECTOR1','VECTOR2'],any).}

\texttt{lex\_equal\_r(VECTOR1,VECTOR2) :-}
\texttt{\quad collection(VECTOR1,[dvar]),}
\texttt{\quad collection(VECTOR2,[dvar]),}
\texttt{\quad length(VECTOR1,L1),}
\texttt{\quad length(VECTOR2,L2),}
\texttt{\quad L1=L2,}
\texttt{\quad get\_attr1(VECTOR1,VECT1),}
\texttt{\quad get\_attr1(VECTOR2,VECT2),}
\texttt{\quad lex\_equal1(VECT1,VECT2).}

\texttt{lex\_equal1([],[]).}

\texttt{lex\_equal1([V1|R1],[V2|R2]) :-}
\texttt{\quad V1#=V2,}
\texttt{\quad lex\_equal1(R1,R2).}

\texttt{lex\_equal\_a(FLAG,VECTOR1,VECTOR2) :-}
\texttt{\quad collection(VECTOR1,[dvar]),}
\texttt{\quad collection(VECTOR2,[dvar]),}
\texttt{\quad length(VECTOR1,L1),}
\texttt{\quad length(VECTOR2,L2),}
\texttt{\quad L1=L2,}
\texttt{\quad lex\_equal\_signature(VECTOR1,VECTOR2,SIGNAL),}
\texttt{\quad AUTOMATON=}
\texttt{\quad automaton(}
\texttt{\quad \quad \quad SIGNAL,}
\texttt{\quad \quad \quad \_37798,}
\texttt{\quad \quad \quad SIGNAL,}
\texttt{\quad \quad \quad [source(s),sink(s)],}
\texttt{\quad \quad \quad [arc(s,1,s)],}
\texttt{\quad \quad \quad [],}
\texttt{\quad \quad \quad []],}
\texttt{\quad \quad \quad []),}
\texttt{\quad automaton\_bool(FLAG,[0,1],AUTOMATON).}

\texttt{lex\_equal\_signature([],[],[]).}
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

\[
\text{lex\_equal\_signature}([\text{var-VAR1}|Xs],[\text{var-VAR2}|Ys],[S|Ss]) :-
S \text{ in } 0..1, \\
\text{VAR1#=}\text{VAR2#} \iff S, \\
\text{lex\_equal\_signature}(Xs,Ys,Ss).
\]
B.209 lex_greater

◊ **META-DATA:**

```prolog
ctr_date(lex_greater,["20030820","20040530","20060811"]).

ctr_origin(lex_greater,\index{CHIP\indexuse}CHIP,[],).

ctr_synonyms(lex_greater,[lex,lex_chain,rel,greater,gt]).

ctr_arguments(
    lex_greater,
    ['VECTOR1'-collection(var-dvar),
    'VECTOR2'-collection(var-dvar)]).

ctr_restrictions(
    lex_greater,
    [required('VECTOR1',var),
    required('VECTOR2',var),
    size('VECTOR1')=size('VECTOR2')]).

ctr_example(
    lex_greater,
    lex_greater(
        [[var-5],[var-2],[var-7],[var-1]],
        [[var-5],[var-2],[var-6],[var-2]]).

ctr_typical(lex_greater,[size('VECTOR1')>1]).

ctr_exchangeable(
    lex_greater,
    [vals(['VECTOR1'\^var],int,<,dontcare,dontcare),
    vals(['VECTOR2'\^var],int,>,dontcare,dontcare)]).

ctr_derived_collections(
    lex_greater,
    [col('DESTINATION'-collection(index-int,x-int,y-int),
        [item(index-0,x-0,y-0)]),
    col('COMPONENTS'-collection(index-int,x-dvar,y-dvar),
        [item(
            index-'VECTOR1'\^key,
            x-'VECTOR1'\^var,
            y-'VECTOR2'\^var)]).

ctr_graph(
    lex_greater,
```
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

['COMPONENTS', 'DESTINATION'],
2,
['PRODUCT'('PATH', 'VOID')>>collection(item1, item2)],
[item2^index>0#/item1^x=item1^y#/item2^index=0#/item1^x>item1^y],
['PATH_FROM_TO' (index, 1, 0)=1],
[]).

ctr_eval(
  lex_greater,
  [builtin(lex_greater_b), automaton(lex_greater_a)].

ctr_extensible(lex_greater, [], ['VECTOR1', 'VECTOR2'], suffix).

lex_greater_b(VECTOR1, VECTOR2) :-
  collection(VECTOR1, [dvar]),
  collection(VECTOR2, [dvar]),
  length(VECTOR1, L1),
  length(VECTOR2, L2),
  L1=L2,
  get_attr1(VECTOR1, VECT1),
  get_attr1(VECTOR2, VECT2),
  lex_chain([VECT2, VECT1], [op(#<)]).

lex_greater_a(FLAG, VECTOR1, VECTOR2) :-
  collection(VECTOR1, [dvar]),
  collection(VECTOR2, [dvar]),
  length(VECTOR1, L1),
  length(VECTOR2, L2),
  L1=L2,
  lex_greater_signature(VECTOR1, VECTOR2, SIGNATURE),
  AUTOMATON=automaton(
    SIGNATURE,
    _47469,
    SIGNATURE,
    [source(s), sink(t)],
    [arc(s, 2, s),
     arc(s, 3, t),
     arc(t, 1, t),
     arc(t, 2, t),
     arc(t, 3, t)],
    [],
    [],
    []),
  automaton_bool(FLAG, [1, 2, 3], AUTOMATON).
lex_greater_signature([],[],[]).

lex_greater_signature([[var-VAR1]|Xs],[[var-VAR2]|Ys],[S|Ss]) :-
  S in 1..3,
  VAR1#<VAR2#<=>S#=1,
  VAR1#=VAR2#<=>S#=2,
  VAR1#>VAR2#<=>S#=3,
  lex_greater_signature(Xs,Ys,Ss).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.210 lex_greatereq

◊ **Meta-Data:**

\[
\begin{align*}
\text{ctr_date}(\text{lex_greatereq}, ['20030820', '20040530', '20060811']). \\
\text{ctr_origin}(\text{lex_greatereq}, ' CHIP', []). \\
\text{ctr_synonyms}( \\
\quad \text{lex_greatereq}, \\
\quad \text{lexeq, lex_chain, rel, greatereq, geq, lex_geq}). \\
\text{ctr_arguments}( \\
\quad \text{lex_greatereq}, \\
\quad ['VECTOR1'\text{-collection}(\text{var-dvar}), \\
\quad 'VECTOR2'\text{-collection}(\text{var-dvar})]). \\
\text{ctr_restrictions}( \\
\quad \text{lex_greatereq}, \\
\quad \text{required('VECTOR1', var),} \\
\quad \text{required('VECTOR2', var),} \\
\quad \text{size('VECTOR1')=size('VECTOR2'))}. \\
\text{ctr_example}( \\
\quad \text{lex_greatereq}, \\
\quad \text{lex_greatereq}([
\begin{array}{c}
\var-5, \\
\var-2, \\
\var-8, \\
\var-9
\end{array}], \\
\begin{array}{c}
\var-5, \\
\var-2, \\
\var-6, \\
\var-2
\end{array}), \\
\text{lex_greatereq}([
\begin{array}{c}
\var-5, \\
\var-2, \\
\var-3, \\
\var-9
\end{array}], \\
\begin{array}{c}
\var-5, \\
\var-2, \\
\var-3, \\
\var-9
\end{array}])). \\
\text{ctr_typical}(\text{lex_greatereq}, \text{size('VECTOR1')}>1]). \\
\text{ctr_exchangeable}( \\
\quad \text{lex_greatereq}, \\
\quad \text{vals(['VECTOR1'\text{\^var}],\text{int},<,\text{dontcare},\text{dontcare}),} \\
\quad \text{vals(['VECTOR2'\text{\^var}],\text{int},>,\text{dontcare},\text{dontcare}]).} \\
\text{ctr_derived_collections}( \\
\quad \text{lex_greatereq}, \\
\quad \text{col('DESTINATION’-collection(index-int,x-int,y-int),} \\
\quad \text{[item(index-0,x-0,y-0))]}, \\
\quad \text{col('COMPONENTS'\text{-collection}(index-int,x-dvar,y-dvar),} \\
\quad \text{[item(index-’VECTOR1’\text{\^key,} \\
\quad \text{index-’VECTOR1’\text{\^key,)}}]}. 
\end{align*}
\]
x='VECTOR1'\text{^var},
y='VECTOR2'\text{^var})).}.

\text{ctr\_graph(}
  \text{lex\_greatereq},
  ['COMPONENTS','DESTINATION'],
  2,
  ['PRODUCT'('PATH','VOID')\text{>>collection(item1,item2)}],
  [item2\text{\_index}0#/item1\text{\_x}=item1\text{\_y}/
   item1\text{\_index}<\text{size('VECTOR1')}/item2\text{\_index}=0#/\item1\text{\_x}>item1\text{\_y}/
   item1\text{\_index}=\text{size('VECTOR1')}/item2\text{\_index}=0#/\item1\text{\_x}=item1\text{\_y}],
  ['PATH\_FROM\_TO'(index,1,0)=1],
  []).}

\text{ctr\_eval(}
  \text{lex\_greatereq},
  [\text{builtin(lex\_greatereq\_b)},\text{automaton(lex\_greatereq\_a)})].

\text{ctr\_contractible(lex\_greatereq,[],['VECTOR1','VECTOR2'],suffix).}

\text{lex\_greatereq\_b(VECTOR1,VECTOR2) :-}
  \text{collection(VECTOR1,[dvar])},
  \text{collection(VECTOR2,[dvar])},
  \text{length(VECTOR1,L1)},
  \text{length(VECTOR2,L2)},
  L1=L2,
  \text{get\_attr1(VECTOR1,VECT1)},
  \text{get\_attr1(VECTOR2,VECT2)},
  \text{lex\_chain([VECT2,VECT1],[op(#=<)])].}

\text{lex\_greatereq\_a(FLAG,VECTOR1,VECTOR2) :-}
  \text{collection(VECTOR1,[dvar])},
  \text{collection(VECTOR2,[dvar])},
  \text{length(VECTOR1,L1)},
  \text{length(VECTOR2,L2)},
  L1=L2,
  \text{lex\_greatereq\_signature(VECTOR1,VECTOR2,SIGNALATURE),}
  \text{AUTOMATON=}
  \text{automaton(}
    \text{SIGNATURE,}
    _50073,
    \text{SIGNATURE,}
    [source(s),sink(s),sink(t)],
    [arc(s,2,s),}
arc(s,3,t),
arc(t,1,t),
arc(t,2,t),
arc(t,3,t)],
[[],
[[]),
automaton_bool(FLAG,[1,2,3],AUTOMATON).

lex_greatareq_signature([],[],[]).

lex_greatareq_signature([[var-VAR1]|Xs],[[var-VAR2]|Ys],[S|Ss]) :-
  S in 1..3,
  VAR1#$<VAR2#<=>S#=1,
  VAR1#$=VAR2#$<=>S#=2,
  VAR1#$>VAR2#$<=>S#=3,
  lex_greatareq_signature(Xs,Ys,Ss).
B.211 lex_less

◊META-DATA:

ctr_date(lex_less,
[’20030820’,’20040530’,’20060811’]).

c ctr_origin(lex_less,’\index{CHIP|indexuse}CHIP’,[]).

ctr_synonyms(lex_less,[lex,lex_chain,rel,less]).

ctr_arguments(

lex_less,
[’VECTOR1’-collection(var-dvar),
 ’VECTOR2’-collection(var-dvar)]).

ctr_restrictions(

lex_less,
[required(’VECTOR1’,var),
 required(’VECTOR2’,var),
 size(’VECTOR1’)=size(’VECTOR2’)].

ctr_example(

lex_less,
lex_less(

[[var-5],[var-2],[var-3],[var-9]],
 [[var-5],[var-2],[var-6],[var-2]]).

ctr_exchangeable(

lex_less,
[vals([’VECTOR1’^var],int,>,dontcare,dontcare),
 vals([’VECTOR2’^var],int,<,dontcare,dontcare)].

ctr_derived_collections(

lex_less,
[col(’DESTINATION’-collection(index-int,x-int,y-int),
 [item(index-0,x-0,y-0)]),
 col(’COMPONENTS’-collection(index-int,x-dvar,y-dvar),
 [item(
    index-’VECTOR1’^key,
    x-’VECTOR1’^var,
    y-’VECTOR2’^var)])].

ctr_graph(

lex_less,
[’COMPONENTS’,’DESTINATION’],
2,
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

['PRODUCT'('PATH','VOID')>>collection(item1,item2),
item2^index>0#/item1^x=item1^y#/item2^index=0#/item1^x<item1^y],
['PATH_FROM_TO'(index,1,0)=1],
[]).

ctr_eval(lex_less,[builtin(lex_less_b),automaton(lex_less_a)]).

ctr_extensible(lex_less,[],['VECTOR1','VECTOR2'],suffix).

lex_less_b(VECTOR1,VECTOR2) :-
collection(VECTOR1,[dvar]),
collection(VECTOR2,[dvar]),
length(VECTOR1,L1),
length(VECTOR2,L2),
L1=L2,
get_attr1(VECTOR1,VECT1),
get_attr1(VECTOR2,VECT2),
lex_chain([VECT1,VECT2],[op(#<)]).

lex_less_a(FLAG,VECTOR1,VECTOR2) :-
collection(VECTOR1,[dvar]),
collection(VECTOR2,[dvar]),
length(VECTOR1,L1),
length(VECTOR2,L2),
L1=L2,
lex_less_signature(VECTOR1,VECTOR2,SIGNATURE),
AUTOMATON=
automaton(

SIGNATURE,
_47575,
SIGNATURE,
[source(s),sink(t)],
[arc(s,2,s),
 arc(s,1,t),
 arc(t,1,t),
 arc(t,2,t),
 arc(t,3,t)],
[],
[],[]),
automaton_bool(FLAG,[1,2,3],AUTOMATON).

lex_less_signature([],[],[]).

lex_less_signature([[var-VAR1]|Xs],[[var-VAR2]|Ys],[S|Ss]) :-
S in 1..3,
VAR1<VAR2#<=>S#=1,
VAR1#=VAR2#<=>S#=2,
VAR1#>VAR2#<=>S#=3,
lex_less_signature(Xs,Ys,Ss).
B.212 lex_lesseq

◊ **Meta-Data:**

```
ctr_date(lex_lesseq,['20030820','20040530','20060811']).

ctr_origin(lex_lesseq,'\index{CHIP|indexuse}CHIP',[]).

ctr_synonyms(lex_lesseq, [lexeq, lex_chain, rel, lesseq, leq, lex_leq]).

ctr_arguments(lex_lesseq, ['VECTOR1'-collection(var-dvar),
                         'VECTOR2'-collection(var-dvar)]).

ctr_restrictions(lex_lesseq, [required('VECTOR1',var),
                             required('VECTOR2',var),
                             size('VECTOR1')=size('VECTOR2')]).

ctr_example(lex_lesseq, [lex_lesseq(lex_lesseq([var-5],[var-2],[var-3],[var-1]),
                                  [var-5],[var-2],[var-6],[var-2]),
                          lex_lesseq([var-5],[var-2],[var-3],[var-9]),
                                  [var-5],[var-2],[var-3],[var-9])).

ctr_typical(lex_lesseq, [size('VECTOR1')>1]).

ctr_exchangeable(lex_lesseq, [vals(['VECTOR1'\-var],int,>,dontcare,dontcare),
                              vals(['VECTOR2'\-var],int,<,dontcare,dontcare)]).

ctr_derived_collections(lex_lesseq, [col('DESTINATION'-collection(index-int,x-int,y-int),
                                       [item(index-0,x-0,y-0)]),
                                    col('COMPONENTS'-collection(index-int,x-dvar,y-dvar),
                                       [item(index-'VECTOR1'\-key,]...
x='VECTOR1' \^ var,  
y='VECTOR2' \^ var).

ctr_graph(
  lex_lesseq,
  ['COMPONENTS', 'DESTINATION'],
  2,
  ['PRODUCT' ('PATH', 'VOID') >> collection(item1, item2)],
  [item2^index>0#\item1^x=item1^y#/
   item1^index<size('VECTOR1')#\item2^index=0#/\n   item1^x<item1^y#/\n   item1^index=size('VECTOR1')#\item2^index=0#/\n   item1^x=<item1^y],
  ['PATH_FROM_TO' (index, 1, 0) = 1],
).

ctr_eval(
  lex_lesseq,
  [builtin(lex_lesseq_b), automaton(lex_lesseq_a)]).

ctr_contractible(lex_lesseq, [], ['VECTOR1', 'VECTOR2'], suffix).

lex_lesseq_b(VECTOR1, VECTOR2) :-
  collection(VECTOR1, [dvar]),
  collection(VECTOR2, [dvar]),
  length(VECTOR1, L1),
  length(VECTOR2, L2),
  L1 = L2,
  get_attr1(VECTOR1, VECT1),
  get_attr1(VECTOR2, VECT2),
  lex_chain([VECT1, VECT2], [op(#=<)]).

lex_lesseq_a(FLAG, VECTOR1, VECTOR2) :-
  collection(VECTOR1, [dvar]),
  collection(VECTOR2, [dvar]),
  length(VECTOR1, L1),
  length(VECTOR2, L2),
  L1 = L2,
  lex_lesseq_signature(VECTOR1, VECTOR2, SIGNATURE),
  AUTOMATON =
  automaton(
    SIGNATURE,
    _52970,
    SIGNATURE,
    [source(s), sink(s), sink(t)],
    [arc(s, 2, s),]


\[
\text{arc}(s,1,t), \\
\text{arc}(t,1,t), \\
\text{arc}(t,2,t), \\
\text{arc}(t,3,t)], \\
\text{[]}, \\
\text{[]}, \\
\text{[]}, \\
\text{automaton_bool(FLAG, [1,2,3], AUTOMATON)}.
\]

\[
\text{lex_lesseq_signature}([], [], []). \\
\text{lex_lesseq_signature}([[\text{var-VAR1}] | Xs], [[\text{var-VAR2}] | Ys], [S | Ss]) :- \\
\hspace{1cm} S \text{ in } 1..3, \\
\hspace{1cm} \text{VAR1}# < \text{VAR2}# <\Rightarrow S# = 1, \\
\hspace{1cm} \text{VAR1}# = \text{VAR2}# <\Rightarrow S# = 2, \\
\hspace{1cm} \text{VAR1}# > \text{VAR2}# <\Rightarrow S# = 3, \\
\hspace{1cm} \text{lex_lesseq_signature}(Xs, Ys, Ss).
\]
B.213 lex_lesseq_allperm

◊ Meta-Data:

ctr_predefined(lex_lesseq_allperm).

ctr_date(lex_lesseq_allperm,['20070916']).

ctr_origin(lex_lesseq_allperm,
    Inspired by \cite{FlenerFrischHnichKiziltanMiguelPearsonWalsh02}, []).

ctr_synonyms(lex_lesseq_allperm,[leximin]).

ctr_arguments(lex_lesseq_allperm,
    ['VECTOR1'-collection(var-dvar),
     'VECTOR2'-collection(var-dvar)]).

ctr_restrictions(lex_lesseq_allperm,
    [required('VECTOR1',var),
     required('VECTOR2',var),
     size('VECTOR1')=size('VECTOR2')]).

ctr_example(lex_lesseq_allperm,
    lex_lesseq_allperm(
        [[var-1],[var-2],[var-3]],
        [[var-3],[var-1],[var-2]]).

ctr_typical(lex_lesseq_allperm,[size('VECTOR1')>1]).

ctr_exchangeable(lex_lesseq_allperm,
    [vals({'VECTOR1''var','VECTOR2'\var},int,\=,all,dontcare)].

ctr_contractible(lex_lesseq_allperm,
    [],
    ['VECTOR1','VECTOR2'],
    suffix).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.214  link_set_to_booleans

♦ META-DATA:

\( \text{ctr\_date(link\_set\_to\_booleans,[}’20030820’,’20060811’)\). 

\( \text{ctr\_origin(link\_set\_to\_booleans,}
\begin{align*}
\text{Inspired by } & \%c., \\
\text{[domain\_constraint].} 
\end{align*}
\)

\( \text{ctr\_arguments(link\_set\_to\_booleans,}
\begin{align*}
\text{[}’\text{SVAR’}-\text{svar,’BOOLEANS’}-\text{collection(bool-dvar,} & \text{val-int)\}).} 
\end{align*}
\)

\( \text{ctr\_restrictions(link\_set\_to\_booleans,}
\begin{align*}
\text{[required(’BOOLEANS’,}\ & \text{[bool,} \text{val]})}, \\
\text{’BOOLEANS’}^{\text{bool}} & \geq 0, \\
\text{’BOOLEANS’}^{\text{bool}} & < 1, \\
\text{distinct(’BOOLEANS’,} \text{val}])\). 
\end{align*}
\)

\( \text{ctr\_example(link\_set\_to\_booleans,}
\begin{align*}
\text{link\_set\_to\_booleans(} \\
\text{\{1,3,4\},} \\
\text{[[bool-0,val-0],} \\
\text{[bool-1,val-1],} \\
\text{[bool-0,val-2],} \\
\text{[bool-1,val-3],} \\
\text{[bool-1,val-4],} \\
\text{[bool-0,val-5]\})].} 
\end{align*}
\)

\( \text{ctr\_typical(link\_set\_to\_booleans,}
\begin{align*}
\text{[size(’BOOLEANS’)\geq 1, range(’BOOLEANS’}^{\text{bool}}\geq 1\]).} 
\end{align*}
\)

\( \text{ctr\_exchangeable(link\_set\_to\_booleans,} \text{[items(’BOOLEANS’,all)\).} 
\)

\( \text{ctr\_derived\_collections(link\_set\_to\_booleans,}
\begin{align*}
\text{[col(’SET’}-\text{collection(one-int,} & \text{setvar-svar)\text{),} \\
\text{[item(one-1,} & \text{setvar-’SVAR’)]\}).} 
\end{align*}
\)

\( \text{ctr\_graph(link\_set\_to\_booleans,}
\begin{align*}
\end{align*}
\)

\( \)
['SET', 'BOOLEANS'],
2,
['PRODUCT'>>collection(set, booleans)],
[booleans^bool=set^one#<=>booleans^val in_set set^setvar],
['NARC'=size('BOOLEANS')],
[[]].
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.215 longest_change

◊ META-DATA:

ctr_date(
    longest_change,  
    ['20000128','20030820','20040530','20060811']).

ctr_origin(longest_change,'Derived from %c.',[change]).

ctr_arguments(
    longest_change,  
    ['SIZE'-dvar,'VARIABLES'-collection(var-dvar),'CTR'-atom]).

ctr_restrictions(
    longest_change,  
    ['SIZE'>=0,  
     'SIZE'<size('VARIABLES'),  
     required('VARIABLES',var),  
     in_list('CTR',[=,\=,<,>,>=,=\=])].

ctr_example(
    longest_change,  
    longest_change(  
        4,  
        [[var-8],  
         [var-8],  
         [var-3],  
         [var-4],  
         [var-1],  
         [var-1],  
         [var-5],  
         [var-5],  
         [var-2]],  
         =\=)).

ctr_typical(
    longest_change,  
    [size('VARIABLES')>2,  
     range('VARIABLES'\var)>1,  
     in_list('CTR',[=\=])].

ctr_exchangeable(longest_change,[translate(['VARIABLES'\var])]).

ctr_graph(
    longest_change,
['VARIABLES'],
2,
['PATH' \rightarrow collection(variables1, variables2)],
['CTR' (variables1 \^ var, variables2 \^ var),
['MAX_NCC' = 'SIZE'],
[]).

ctr_eval(longest_change, [automaton(longest_change_a)]).

ctr_pure_functional_dependency(longest_change, []).

ctr_functional_dependency(longest_change, 1, [2, 3]).

longest_change_a(FLAG, SIZE, VARIABLES, CTR) :-
collection(VARIABLES, [dvar]),
length(VARIABLES, N),
N_1 is N-1,
check_type(dvar(0, N_1), SIZE),
memberchk(CTR, [=, =\-, <, >=, >, =\-<]),
longest_change_signature(VARIABLES, SIGNATURE, CTR),
automaton(  
  SIGNATURE,  
  _32993,  
  SIGNATURE,  
  [source(s), sink(s)],  
  [arc(s, 1, s, [C, D+1]), arc(s, 0, s, [max(C, D), 1])],  
  [C, D],  
  [0, 1],  
  [C1, D1]),
SIZE#=max(C1, D1)#\leftrightarrow FLAG.

longest_change_signature([], [], _31257).

longest_change_signature([_31261], [], _31260) :- !.

longest_change_signature([[var-VAR1], [var-VAR2]|VARs], [S|Ss], =) :- !,
  VAR1#=VAR2#\leftrightarrow S,
  longest_change_signature([[var-VAR2]|VARs], Ss, =).

longest_change_signature(  
  [[var-VAR1], [var-VAR2]|VARs],  
  [S|Ss],  
  =\leftrightarrow =) :- !,
VAR1\#\=VAR2#<=>S, 
longest_change_signature([[var-VAR2]|VARs],Ss,\=\=).

longest_change_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss],<) :- 
!,
VAR1#<VAR2#<=>S, 
longest_change_signature([[var-VAR2]|VARs],Ss,<).

longest_change_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    >=) :- 
!,
VAR1#>=VAR2#<=>S, 
longest_change_signature([[var-VAR2]|VARs],Ss,>=).

longest_change_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss],>) :- 
!,
VAR1#>VAR2#<=>S, 
longest_change_signature([[var-VAR2]|VARs],Ss,>).

longest_change_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    =<) :- 
!,
VAR1#=<VAR2#<=>S, 
longest_change_signature([[var-VAR2]|VARs],Ss,=<).
**B.216  lt**

◊ **Meta-Data:**

```prolog
ctr_predefined(lt).
ctr_date(lt,['20070821']).
ctr_origin(lt,'Arithmetic.',[]).
ctr_synonyms(lt,[rel,xlt]).
ctr_arguments(lt,['VAR1'-dvar,'VAR2'-dvar]).
ctr_example(lt,lt(1,8)).
ctr_exchangeable(lt,
  [vals(['VAR1'],int(<('VAR2')),\=,all,dontcare),
   vals(['VAR2'],int(>'VAR1')),\=,all,dontcare)]).
ctr_eval(lt,[builtin(lt_b)]).

lt_b(VAR1,VAR2) :-
    check_type(dvar,VAR1),
    check_type(dvar,VAR2),
    VAR1<VAR2.
```
B.217  map

◊ **Meta-Data:**

```prolog
ctr_date(map,[‘20000128’,’20030820’,’20060811’]).

ctr_origin(map,’Inspired by \cite{SedgewickFlajolet96’],[]).

ctr_arguments(  
    map,  
    [’NBCYCLE’-dvar,  
    ’NBTREE’-dvar,  
    ’NODES’-collection(index-int,succ-dvar)])).

ctr_restrictions(  
    map,  
    [’NBCYCLE’>=0,  
    ’NBTREE’>=0,  
    required(’NODES’,[index,succ]),  
    ’NODES’^index>=1,  
    ’NODES’^index=<size(’NODES’),  
    distinct(’NODES’,index),  
    ’NODES’^succ>=1,  
    ’NODES’^succ=<size(’NODES’)]).

ctr_example(  
    map,  
    map(2,  
    3,  
    [[index-1,succ-5],  
    [index-2,succ-9],  
    [index-3,succ-8],  
    [index-4,succ-2],  
    [index-5,succ-9],  
    [index-6,succ-2],  
    [index-7,succ-9],  
    [index-8,succ-8],  
    [index-9,succ-1]])).

ctr_typical(  
    map,  
    [’NBCYCLE’>0,  
    ’NBTREE’>0,  
    ’NBCYCLE’<size(’NODES’),  
    ’NBCYCLE’<’NBTREE’,  
    size(’NODES’)>,2]).
```
ctr_exchangeable(map, [items('NODES', all)]).

ctr_graph(
    map,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1, nodes2)],
    [nodes1^succ=nodes2^index],
    ['NCC'='NBCYCLE', 'NTREE'='NBTREE'],
    []).

ctr_pure_functional_dependency(map, []).

ctr_functional_dependency(map, 1, [3]).

ctr_functional_dependency(map, 2, [3]).
B.218  max_index

♦ Meta-Data:

c_{\text{date}}(\text{max\_index},
\text{\[20030820,\ 20040530,\ 20041230,\ 20060811\]}).

c_{\text{origin}}(\text{max\_index},\text{N. Beldiceanu},[{}]).

\text{c}_{\text{arguments}}(\text{max\_index},
\text{\[\text{MAX\_INDEX}\rightarrow \text{dvar},
\text{VARIABLES}\rightarrow \text{collection(index\_int,\text{var\_dvar})}\]}).

\text{c}_{\text{restrictions}}(\text{max\_index},
\text{\[\text{\text{size\('VARIABLES')}>0,\
\text{\text{MAX\_INDEX}}\geq 0,\
\text{\text{MAX\_INDEX}}\leq \text{\text{size\('VARIABLES')},\
\text{\text{required\('VARIABLES',[\text{index,\text{var}}])},\
\text{\text{VARIABLES}'^\text{index}}\geq 1,\
\text{\text{VARIABLES}'^\text{index}}\leq \text{\text{size\('VARIABLES')},\
\text{\text{distinct\('VARIABLES',\text{index})}}\].\]}).

\text{c}_{\text{example}}(\text{max\_index},
\text{max\_index}(\text{3,}
\text{\[\text{index-1,\text{var-3},\text{index-2,\text{var-2},\text{index-3,\text{var-7},\text{index-4,\text{var-2},\text{index-5,\text{var-7}}}}\]})).

\text{c}_{\text{typical}}(\text{max\_index},
\text{\[\text{\text{size\('VARIABLES')}>0,\text{\text{range\('VARIABLES'\_\text{var}>)1\]}\].\]}).

\text{c}_{\text{exchangeable}}(\text{max\_index},
\text{\[\text{\text{items\('VARIABLES',\text{all})},\text{\text{translate\([\text{'VARIABLES'\_\text{var}}])}}\].\]}).

\text{c}_{\text{graph}}(\text{max\_index},
\text{\[\text{\text{'VARIABLES'}]}\],
2,
['CLIQUE'>>collection(variables1,variables2)],
[variables1\^key=variables2\^key#
  variables1\^var>variables2\^var],
['ORDER' (0,0,index)='MAX_INDEX'],
[]).
B.219 max_n

◊ Meta-Data:

`ctr_date(max_n, [‘20000128’, ‘20030820’, ‘20041230’, ‘20060811’]).`

`ctr_origin(max_n, ‘\cite{Beldiceanu01}’, []).`

`ctr_arguments(
  max_n,
  [‘MAX’-dvar, ‘RANK’-int, ‘VARIABLES’-collection(var-dvar)]).
`

`ctr_restrictions(
  max_n,
  [‘RANK’>=0,
   ‘RANK’<size(‘VARIABLES’),
   size(‘VARIABLES’)>0,
   required(‘VARIABLES’, var)]).
`

`ctr_example(
  max_n,
  max_n(6, 1, [[var-3], [var-1], [var-7], [var-1], [var-6]]).
`

`ctr_typical(
  max_n,
  [‘RANK’>0,
   ‘RANK’<3,
   size(‘VARIABLES’)>1,
   range(‘VARIABLES’-^var)>1]).
`

`ctr_exchangeable(
  max_n,
  [items(‘VARIABLES’, all),
   translate([‘MAX’, ‘VARIABLES’-^var])].
`

`ctr_graph(
  max_n,
  [‘VARIABLES’],
  2,
  [‘CLIQUE’>>collection(variables1, variables2)],
  [variables1-^key=variables2-^key#/variables1-^var>variables2-^var],
  [‘ORDER’(‘RANK’, ‘MININT’, var)=‘MAX’],
  []).
`

`ctr_eval(max_n, [reformulation(max_n_r)]).`
ctr_pure_functional_dependency(max_n,[]).
ctr_functional_dependency(max_n,1,[2,3]).

max_n_r(MAX,RANK,VARIABLES) :-
    length(VARIABLES,N),
    N>0,
    N1 is N-1,
    check_type(dvar,MAX),
    check_type(int(0,N1),RANK),
    collection(VARIABLES,[dvar]),
    get_attr1(VARIABLES,VARS),
    create_collection([MAX],var,VMAX),
    create_collection(VARS,val,VALUES),
    eval(among_var(1,VMAX,VALUES)),
    NVAL in 0..N,
    eval(nvalue(NVAL,VARIABLES)),
    length(RANKS,N),
    domain(RANKS,0,N1),
    max_n1(VARS,RANKS,MAX,RANK,NVAL).

max_n1([],[],_31277,_31278,_31279).

max_n1([V|RV],[R|RR],MAX,RANK,NVAL) :-
    R#<NVAL,
    R#=RANK#<=>V#=MAX,
    max_n2(RV,RR,V,R),
    max_n1(RV,RR,MAX,RANK,NVAL).

max_n2([],[],_31277,_31278).

max_n2([Vj|RV],[Rj|RR],Vi,Ri) :-
    Vi#>Vj#<=>Ri#<Rj,
    Vi#=Vj#<=>Ri#=Rj,
    Vi#<Vj#<=>Ri#>Rj,
    max_n2(RV,RR,Vi,Ri).
B.220  \text{max\_nvalue}

\textbf{Meta-Data:}

\texttt{ctr\_date(max\_nvalue,[’20000128’,’20030820’,’20060811’]).}

\texttt{ctr\_origin(max\_nvalue,’Derived from \&c.’,[nvalue]).}

\texttt{ctr\_arguments(
  max\_nvalue,
  [’MAX’-dvar,’VARIABLES’-collection(var-dvar)]).}

\texttt{ctr\_restrictions(
  max\_nvalue,
  [’MAX’>=1,
   ’MAX’=<size(’VARIABLES’),
   required(’VARIABLES’,var)]).}

\texttt{ctr\_example(
  max\_nvalue,
  max\_nvalue(3,
    [[var-9],
     [var-1],
     [var-7],
     [var-1],
     [var-1],
     [var-6],
     [var-7],
     [var-7],
     [var-4],
     [var-9]])).}

\texttt{ctr\_typical(
  max\_nvalue,
  [’MAX’1,
   ’MAX’<size(’VARIABLES’),
   size(’VARIABLES’)>1,
   range(’VARIABLES’\^{}var)>1]).}

\texttt{ctr\_exchangeable(
  max\_nvalue,
  [items(’VARIABLES’,all),
   vals([’VARIABLES’\^{}var],int,=\&all,dontcare)]).}

\texttt{ctr\_graph(}
max_nvalue,
['VARIABLES'],
2,
['CLIQUE'>>collection(variables1,variables2)],
[variables1^var=variables2^var],
['MAX_NSNC'='MAX'],
[]).

ctr_eval(max_nvalue,[reformulation(max_nvalue_r)]).

ctr_pure_functional_dependency(max_nvalue,[]).

ctr_functional_dependency(max_nvalue,1,[2]).

max_nvalue_r(0,[]) :-
  !.

max_nvalue_r(MAX,VARIABLES) :-
  length(VARIABLES,N),
  check_type(dvar(1,N),MAX),
  collection(VARIABLES,[dvar]),
  get_attr1(VARIABLES,VARS),
  get_minimum(VARS,MINVARS),
  get_maximum(VARS,MAXVARS),
  max_nvalue1(MINVARS,MAXVARS,N,VALS_OCCS,OCCS),
  global_cardinality(VARS,VALS_OCCS),
  maximum(MAX,OCCS).

max_nvalue1(MIN,MAX,_,50292,[],[]) :-
  MIN>MAX,
  !.

max_nvalue1(CUR,MAX,N,[CUR-OCC|R],[OCC|S]) :-
  CUR=<MAX,
  OCC in 0..N,
  CUR1 is CUR+1,
  max_nvalue1(CUR1,MAX,N,R,S).
B.221  max_size_set_of_consecutive_var

◊  **Meta-Data:**

```prolog
ctr_date(
    max_size_set_of_consecutive_var,
    [‘20030820’,’20040530’,’20060811’]).

ctr_origin(max_size_set_of_consecutive_var,’N.„Beldiceanu’,[]).

ctr_arguments(
    max_size_set_of_consecutive_var,
    [‘MAX’-dvar,’VARIABLES’-collection(var-dvar)]).

ctr_restrictions(
    max_size_set_of_consecutive_var,
    [‘MAX’>=1,
     ‘MAX’=<size(‘VARIABLES’),
     required(‘VARIABLES’,var)]).

ctr_example(
    max_size_set_of_consecutive_var,
    max_size_set_of_consecutive_var(6,
     [[var-3],[var-1],[var-3],[var-7],[var-4],[var-1],[var-2],[var-8],[var-7],[var-6]])).

ctr_typical(
    max_size_set_of_consecutive_var,
    [‘MAX’<size(‘VARIABLES’),
     size(‘VARIABLES’)>0,
     range(‘VARIABLES’^var)>1]).

ctr_exchangeable(
    max_size_set_of_consecutive_var,
    [items(‘VARIABLES’,all),
     vals([‘VARIABLES’^var],int,=\=,all,in),
     translate([‘VARIABLES’^var]])).
```
ctr_graph(
    max_size_set_of_consecutive_var,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [abs(variables1^var-variables2^var)=<1],
    ['MAX_NSNC'='MAX'],
    []).

ctr_pure_functional_dependency(
    max_size_set_of_consecutive_var,
    []).

ctr_functional_dependency(
    max_size_set_of_consecutive_var,
    1,
    [2]).
B.222 maximum

◊ Meta-Data:

$$\texttt{ctr\_date(maximum, [20000128, 20030820, 20040530, 20041230, 20060811, 20090416]).}$$

$$\texttt{ctr\_origin(maximum, \\index{CHIP|indexuse}CHIP', []).}$$

$$\texttt{ctr\_synonyms(maximum, [max]).}$$

$$\texttt{ctr\_arguments(maximum, ['MAX'-dvar, 'VARIABLES'-collection(var-dvar)].}$$

$$\texttt{ctr\_restrictions(maximum, [size('VARIABLES')>0, required('VARIABLES', var)].}$$

$$\texttt{ctr\_example(maximum, maximum(7, [[var-3], [var-2], [var-7], [var-2], [var-6]].}$$

$$\texttt{ctr\_typical(maximum, [size('VARIABLES')>1, range('VARIABLES'\^var)>1)].}$$

$$\texttt{ctr\_exchangeable(maximum, [items('VARIABLES', all), vals(['VARIABLES'\^var], int, =\\=, all, in), translate(['MAX', 'VARIABLES'\^var]).}$$

$$\texttt{ctr\_graph(maximum, ['VARIABLES'], 2, ['CLIQUE'>>collection(variables1, variables2)], [variables1'\texttt{\textbackslash key}=variables2'\texttt{\textbackslash key}#\slash variables1'\texttt{\textbackslash var}>variables2'\texttt{\textbackslash var}],}$$
['ORDER'(0,'MININT',var)='MAX'],
[]).

ctr_eval(maximum,[builtin(maximum_b),automaton(maximum_a)]).

ctr_pure_functional_dependency(maximum,[]).

ctr_functional_dependency(maximum,1,[2]).

ctr_aggregate(maximum,[],[max,union]).

maximum_b(MAX,VARIABLES) :-
  check_type(dvar,MAX),
  collection(VARIABLES,[dvar]),
  length(VARIABLES,N),
  N>0,
  get_attr1(VARIABLES,VARS),
  maximum(MAX,VARS).

maximum_a(FLAG,MAX,VARIABLES) :-
  check_type(dvar,MAX),
  collection(VARIABLES,[dvar]),
  length(VARIABLES,N),
  N>0,
  maximum_signature(VARIABLES,SIGNATURE,MAX),
  AUTOMATON=
  automaton(
    SIGNATURE,
    _42452,
    SIGNATURE,
    [source(s),sink(t)],
    [arc(s,0,s),arc(s,1,t),arc(t,1,t),arc(t,0,t)],
    [],
    [],
    []),
  automaton_bool(FLAG,[0,1,2],AUTOMATON).

maximum_signature([],[],_40726).

maximum_signature([[var-VAR]|VARs],[S|Ss],MAX) :-
  S in 0..2,
  MAX#>VAR#<=>S#=0,
  MAX#=VAR#<=>S#=1,
  MAX#<VAR#<=>S#=2,
  maximum_signature(VARs,Ss,MAX).
B.223 maximum_modulo

◊ Meta-Data:

```
ctr_date(maximum_modulo, ['20000128', '20030820', '20041230', '20060811']).
ctr_origin(maximum_modulo, 'Derived from %c.', [maximum]).
ctr_arguments(maximum_modulo, ['MAX'-dvar, 'VARIABLES'-collection(var-dvar), 'M'-int]).
ctr_restrictions(maximum_modulo, [size('VARIABLES')>0, 'M'>0, required('VARIABLES', var)]).
ctr_example(maximum_modulo, maximum_modulo(5, [[var-9], [var-1], [var-7], [var-6], [var-5]], 3)).
ctr_typical(maximum_modulo, ['M'>1, 'M'<maxval('VARIABLES'ˆvar), size('VARIABLES')>1, range('VARIABLES'ˆvar)>1]).
ctr_exchangeable(maximum_modulo, [items('VARIABLES', all)]).
ctr_graph(maximum_modulo, ['VARIABLES'], 2, ['CLIQUE'>>collection(variables1, variables2)], [variables1'key=variables2'key#
\variables1'var mod 'M'>variables2'var mod 'M'], ['ORDER' (0, 'MININT', var)= 'MAX'], []).
ctr_pure_functional_dependency(maximum_modulo, []).```
ctr_functional_dependency(maximum_modulo,1,[2,3]).
B.224 meet_sboxes

◊ Meta-Data:

\[
\text{ctr\_date(meet\_sboxes, ['20070622', '20090725'])}.
\]

\[
\text{ctr\_origin(}
\begin{array}{l}
\text{meet\_sboxes,} \\
\text{Geometry, derived from } \text{cite(RandellCuiCohn92),} \\
\text{[]).}
\end{array}
\]

\[
\text{ctr\_synonyms(meet\_sboxes, [meet]).}
\]

\[
\text{ctr\_types(}
\begin{array}{l}
\text{meet\_sboxes,} \\
\text{['VARIABLES'-collection(v-dvar),} \\
\text{'INTEGERS'-collection(v-int),} \\
\text{'POSITIVES'-collection(v-int)]}.}
\end{array}
\]

\[
\text{ctr\_arguments(}
\begin{array}{l}
\text{meet\_sboxes,} \\
\text{['K'-int,} \\
\text{'DIMS'-sint,} \\
\text{'OBJECTS'-collection(oid-int, sid-int, x-'VARIABLES'),} \\
\text{'SBOXES'-collection(sid-int, t-'INTEGERS', l-'POSITIVES')].}
\end{array}
\]

\[
\text{ctr\_restrictions(}
\begin{array}{l}
\text{meet\_sboxes,} \\
\text{[size('VARIABLES')>=1,} \\
\text{size('INTEGERS')>=1,} \\
\text{size('POSITIVES')>=1,} \\
\text{required('VARIABLES', v),} \\
\text{size('VARIABLES')='K',} \\
\text{required('INTEGERS', v),} \\
\text{size('INTEGERS')='K',} \\
\text{required('POSITIVES', v),} \\
\text{size('POSITIVES')='K',} \\
\text{'POSITIVES'\^{}v>0,} \\
\text{'K'>0,} \\
\text{'DIMS'>=0,} \\
\text{'DIMS'<'K',} \\
\text{increasing_seq('OBJECTS', [oid]),} \\
\text{required('OBJECTS', [oid, sid, x]),} \\
\text{'OBJECTS'\^{}oid>=1,} \\
\text{'OBJECTS'\^{}oid=<size('OBJECTS'),} \\
\text{'OBJECTS'\^{}sid>=1,}
\end{array}
\]
'OBJECTS' sid=size('SBOXES'),
size('SBOXES')>=1,
required('SBOXES', [sid, t, l]),
'SBOXES' sid=1,
'SBOXES' sid=size('SBOXES'),
do_not_overlap('SBOXES')).

ctr_example(
    meet_sboxes,
    meet_sboxes(2,
        [0,1],
        [[oid-1, sid-1, x-[[v-3], [v-2]]],
         [oid-2, sid-2, x-[[v-4], [v-1]]],
         [oid-3, sid-4, x-[[v-3], [v-4]]],
         [[sid-1, t-[[v-0], [v-0]], l-[[v-1], [v-2]]],
         [sid-2, t-[[v-0], [v-0]], l-[[v-1], [v-1]]],
         [sid-2, t-[[v-1], [v-0]], l-[[v-1], [v-3]]],
         [sid-2, t-[[v-0], [v-2]], l-[[v-1], [v-1]]],
         [sid-3, t-[[v-0], [v-0]], l-[[v-3], [v-1]]],
         [sid-3, t-[[v-0], [v-1]], l-[[v-1], [v-1]]],
         [sid-3, t-[[v-2], [v-1]], l-[[v-1], [v-1]]],
         [sid-4, t-[[v-0], [v-0]], l-[[v-1], [v-1]]])))

ctr_typical(meet_sboxes, [size('OBJECTS')>1]).

ctr_exchangeable(
    meet_sboxes,
    [items('OBJECTS', all),
    items('SBOXES', all),
    items_sync('OBJECTS' x, 'SBOXES' t, 'SBOXES' l, all)]).

ctr_eval(meet_sboxes, [logic(meet_sboxes_g)]).

ctr_logic(
    meet_sboxes,
    [DIMENSIONS, OIDS],
    [(origin(O1, S1, D)---O1'x(D)+S1't(D)),
    (end(O1, S1, D)---O1'x(D)+S1't(D)+S1'l(D)),
    (non_overlap_sboxes(Dims, O1, S1, O2, S2)---
    exists(
    D,
    Dims,
    end(O1, S1, D)#=<origin(O2, S2, D)#/
    end(O2, S2, D)#=<origin(O1, S1, D))],
    (meet_sboxes(Dims, O1, S1, O2, S2)---
exists(D, Dims, end(O1,S1,D)#=origin(O2,S2,D)#\/
   end(O2,S2,D)#=origin(O1,S1,D))),
(meet_objects(Dims,O1,O2)--->
forall(S1, sboxes([O1\^sid]),
forall(S2, sboxes([O2\^sid]),
non_overlap_sboxes(Dims,O1,S1,O2,S2))#/\exists(S1, sboxes([O1\^sid]),
exists(S2, sboxes([O2\^sid]),
meet_sboxes(Dims,O1,S1,O2,S2)))),
(all_meet(Dims,OIDS)--->
forall(O1, objects(OIDS),
forall(O2, objects(OIDS),
O1\^oid#<O2\^oid#==>
meet_objects(Dims,O1,O2))),
all_meet(DIMENSIONS,OIDS))).

ctr_contractible(meet_sboxes,[],'OBJECTS',suffix).

meet_sboxes_g(K,_32954,[],_32956) :- !,
   check_type(int_gteq(1),K).

meet_sboxes_g(K,_DIMS,OBJECTS,SBOXES) :-
   length(OBJECTS,O),
   length(SBOXES,S),
   O>0,
   S>0,
   check_type(int_gteq(1),K),
   collection(OBJECTS,[int(1,O),dvar(1,S),col(K,[dvar])]),
   collection(SBOXES,
   [int(1,S),col(K,[int]),col(K,[int_gteq(1)])]),
get_attr1(OBJECTS, OIDS),
get_attr2(OBJECTS, SIDS),
get_col_attr3(OBJECTS, 1, XS),
gest1(SIDES, TS, TL, Sboxes),
collection_increasing_seq(OBJECTS, [1]),
geost1(OIDS, SIDS, XS, Objects),
geost2(SIDES, TS, TL, Sboxes),
geost_dims(1, K, DIMENSIONS),
ctr_logic(meet_sboxes, [DIMENSIONS, OIDS], Rules),
geost(Objects, Sboxes, [overlap(true)], Rules).
B.225  min_index

◊ **Meta-Data:**

```prolog
ctr_date(
    min_index,
    ['20030820','20040530','20041230','20060811']).

ctr_origin(min_index,'N. Beldiceanu',[]).

ctr_arguments(
    min_index,
    ['MIN_INDEX'-dvar,
     'VARIABLES'-collection(index-int,var-dvar)]).

ctr_restrictions(
    min_index,
    [size('VARIABLES')>0,
     'MIN_INDEX'>=0,
     'MIN_INDEX'=<size('VARIABLES'),
     required('VARIABLES',[index,var]),
     'VARIABLES'\index>=1,
     'VARIABLES'\index=<size('VARIABLES'),
     distinct('VARIABLES',index)]).

ctr_example(
    min_index,
    [min_index(2,
        [[index-1,var-3],
         [index-2,var-2],
         [index-3,var-7],
         [index-4,var-2],
         [index-5,var-6]]),
     min_index(4,
        [[index-1,var-3],
         [index-2,var-2],
         [index-3,var-7],
         [index-4,var-2],
         [index-5,var-6]])]).

ctr_typical(
    min_index,
    [size('VARIABLES')>0,range('VARIABLES'\var)>1]).
```


```prolog
ctr_exchangeable(
    min_index,
    [items('VARIABLES', all), translate(['VARIABLES' \^ var])].

ctr_graph(
    min_index,
    ['VARIABLES'],
    2,
    ['CLIQUE' \^ collection(variables1, variables2)],
    [variables1 \^ key=variables2 \^ key \
      variables1 \^ var<variables2 \^ var],
    ['ORDER' (0, 0, index)= 'MIN_INDEX'],
    []).```

APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.226 $\text{min}_n$

♦ Meta-Data:

$\text{ctr_date}(\text{min}_n, ["20000128","20030820","20040530","20041230","20060811"]).$

$\text{ctr_origin}(\text{min}_n, \cite{Beldiceanu01}, []).$

$\text{ctr_arguments}(\text{min}_n, [\text{MIN'}-\text{dvar}, \text{RANK'}-\text{int}, \text{VARIABLES'}-\text{collection(}\text{var}-\text{dvar})]).$

$\text{ctr_restrictions}(\text{min}_n, [\text{size(}\text{VARIABLES}')>0,
\text{RANK'}>=0,
\text{RANK'}<\text{size(}\text{VARIABLES}')],
\text{required(}\text{VARIABLES'},\text{var})).$

$\text{ctr_example}(\text{min}_n, \text{min}_n(3,1,[\text{var}-3],[\text{var}-1],[\text{var}-7],[\text{var}-1],[\text{var}-6])).$

$\text{ctr_typical}(\text{min}_n, [\text{RANK'}>0,
\text{RANK'}<3,
\text{size(}\text{VARIABLES}')>1,
\text{range(}\text{VARIABLES'}^\text{var}>1)].$

$\text{ctr_exchangeable}(\text{min}_n, [\text{items(}\text{VARIABLES'},\text{all}),
\text{translate([\text{MIN'},}\text{VARIABLES'}^\text{var}])]).$

$\text{ctr_graph}(\text{min}_n, [\text{VARIABLES'}],
2,
[\text{CLIQUE}>>\text{collection(}\text{variables1},\text{variables2})],
[\text{variables1\text{key}=variables2\text{key#}/}
\text{variables1\text{var}<variables2\text{var}}],
[\text{ORDER}('\text{RANK'},\text{MAXINT'},\text{var}=\text{MIN'}],
[]).$
ctr_eval(min_n,[reformulation(min_n_r)]).

ctr_pure_functional_dependency(min_n,[]).

ctr_functional_dependency(min_n,1,[2,3]).

min_n_r(MIN,RANK,VARIABLES) :-
  length(VARIABLES,N),
  N>0,
  N1 is N-1,
  check_type(dvar,MIN),
  check_type(int(0,N1),RANK),
  collection(VARIABLES,[dvar]),
  get_attr1(VARIABLES,VARS),
  create_collection([MIN],var,VMIN),
  create_collection(VARS,val,VALUES),
  eval(among_var(1,VMIN,VALUES)),
  NVAL in 0..N,
  eval(nvalue(NVAL,VARIABLES)),
  length(RANKS,N),
  domain(RANKS,0,N1),
  min_n1(VARS,RANKS,MIN,RANK,NVAL).

min_n1([],[],_33802,_33803,_33804).

min_n1([V|RV],[R|RR],MIN,RANK,NVAL) :-
  R#<NVAL,
  R#=RANK#<=>V#=MIN,
  min_n2(RV,RR,V,R),
  min_n1(RV,RR,MIN,RANK,NVAL).

min_n2([],[],_33802,_33803).

min_n2([Vj|RV],[Rj|RR],Vi,Ri) :-
  Vi#<Vj#<=>Ri#<Rj,
  Vi#=Vj#<=>Ri#=Rj,
  Vi#>Vj#<=>Ri#>Rj,
  min_n2(RV,RR,Vi,Ri).
B.227  min\_nvalue

\begin{itemize}
\item **Meta-Data:**
\end{itemize}

\begin{verbatim}
ctr_date(min_nvalue,['20000128','20030820','20060811']).
ctr_origin(min_nvalue,'N.˜Beldiceanu',[]).
ctr_arguments(
  min_nvalue,
  ['MIN'-dvar,'VARIABLES'-collection(var-dvar)]).
ctr_restrictions(
  min_nvalue,
  ['MIN'>=1,
   'MIN'=<size('VARIABLES'),
   required('VARIABLES',var)]).
ctr_example(
  min_nvalue,
  min_nvalue(
      2,
      [[var-9],
       [var-1],
       [var-7],
       [var-1],
       [var-1],
       [var-7],
       [var-7],
       [var-7],
       [var-7],
       [var-9]])).
ctr_typical(
  min_nvalue,
  [2*\textquoteleft MIN'}=<size('VARIABLES'),
   size('VARIABLES')>1,
   range('VARIABLES'\textquoteleft var}1]).
ctr_exchangeable(
  min_nvalue,
  [items('VARIABLES',all),
   vals(['VARIABLES'\textquoteleft var},int,\textquoteleft =\textquoteright ,all,dontcare])].
ctr_graph(
  min_nvalue,
['VARIABLES'],
2,
['CLIQUE'>>collection(variables1,variables2)],
[variables1^var=variables2^var],
['MIN_NSCC'='MIN'],
[]).

ctr_eval(min_nvalue,[reformulation(min_nvalue_r)]).

ctr_pure_functional_dependency(min_nvalue,[]).

ctr_functional_dependency(min_nvalue,1,[2]).

min_nvalue_r(0,[]) :-
  !.

min_nvalue_r(MIN,VARIABLES) :-
  length(VARIABLES,N),
  check_type(dvar(1,N),MIN),
  collection(VARIABLES,[dvar]),
  get_attr1(VARIABLES,VARS),
  get_minimum(VARS,MINVARS),
  get_maximum(VARS,MAXVARS),
  max_nvalue1(MINVARS,MAXVARS,N,VALS_OCCS,OCCS),
  global_cardinality(VARS,VALS_OCCS),
  append([0],OCCS,OCCS0),
  create_collection(OCCS0,var,VOCCS0),
  eval(min_n(MIN,1,VOCCS0)).
B.228 min_size_set_of_consecutive_var

\textbf{Meta-Data:}

\texttt{ctr\_date}(
  \texttt{min\_size\_set\_of\_consecutive\_var},
  [\texttt{20030820'},\texttt{20040530'},\texttt{20060811'}]).

\texttt{ctr\_origin}min_size_set_of_consecutive_var,\texttt{N.'Beldiceanu',[]}.

\texttt{ctr\_arguments}
  \texttt{min\_size\_set\_of\_consecutive\_var},
  [\texttt{MIN'-dvar,'VARIABLES'-collection(var-dvar)}].

\texttt{ctr\_restrictions}(
  \texttt{min\_size\_set\_of\_consecutive\_var},
  [\texttt{MIN'}=\texttt{1},
   \texttt{MIN'}=\texttt{size('VARIABLES'),
     required('VARIABLES',var)}]).

\texttt{ctr\_example}(
  \texttt{min\_size\_set\_of\_consecutive\_var},
  \texttt{min\_size\_set\_of\_consecutive\_var}(
    \texttt{4},
    \texttt{[var-3],
      [var-1],
      [var-3],
      [var-7],
      [var-4],
      [var-1],
      [var-2],
      [var-8],
      [var-7],
      [var-6]])).

\texttt{ctr\_typical}(
  \texttt{min\_size\_set\_of\_consecutive\_var},
  [\texttt{MIN'}>\texttt{1},
   \texttt{MIN'}<\texttt{size('VARIABLES'),
     size('VARIABLES')}>\texttt{0},
     range('VARIABLES'\texttt{`var}>\texttt{1}]).

\texttt{ctr\_exchangeable}(
  \texttt{min\_size\_set\_of\_consecutive\_var},
  \texttt{items('VARIABLES',all),
     vals(['VARIABLES'\texttt{`var},int,=\texttt{|},all,in),}]}
translate(['VARIABLES'\^\(\text{\textasciitilde}\text{\textvar})\])).

ctr_graph(
    min_size_set_of_consecutive_var,
    ['VARIABLES'],
    2,
    ['\text{\textCLIQUE}'\>>\text{collection}(\text{variables1},\text{variables2})],
    [abs(\text{variables1}^\text{\textvar}-\text{variables2}^\text{\textvar})=<1],
    ['\text{MIN\_NSCC'}='\text{MIN'}],
    []).

ctr_pure_functional_dependency(
    min_size_set_of_consecutive_var,
    []).

ctr_functional_dependency(
    min_size_set_of_consecutive_var,
    1,
    [2]).
B.229 minimum

♦ Meta-Data:

ctr_date(
    minimum,
    [20000128,
     20030820,
     20040530,
     20041230,
     20060811,
     20090416]).

ctr_origin(minimum,'\index{CHIP|indexuse}CHIP',[]).

ctr_synonyms(minimum,[min]).

ctr_arguments(
    minimum,
    ['MIN'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    minimum,
    [size('VARIABLES')>0,required('VARIABLES',var)]).

ctr_example(
    minimum,
    minimum(2,[[var-3],[var-2],[var-7],[var-2],[var-6]])).

ctr_typical(
    minimum,
    [size('VARIABLES')>1,range('VARIABLES'\^\var)>1]).

ctr_exchangeable(
    minimum,
    [items('VARIABLES',all),
     vals(['VARIABLES'\^\var],int,\=\,all,in),
     translate(['MIN','VARIABLES'\^\var])].

ctr_graph(
    minimum,
    ['VARIABLES'],
    2,
    ['CLIQUE']>>collection(variables1,variables2),
    [variables1\^key=variables2\^key#/
     variables1\^var<variables2\^var],
['ORDER'(0,'MAXINT',var)=MIN'],

ctr_eval(minimum,[builtin(minimum_b),automaton(minimum_a)]).

ctr_pure_functional_dependency(minimum,[]).

ctr_functional_dependency(minimum,1,[2]).

ctr_aggregate(minimum,[],[min,union]).

minimum_b(MIN,VARIABLES) :-
    check_type(dvar,MIN),
    collection(VARIABLES,[dvar]),
    length(VARIABLES,N),
    N>0,
    get_attr1(VARIABLES,VARS),
    minimum(MIN,VARS).

minimum_a(FLAG,MIN,VARIABLES) :-
    check_type(dvar,MIN),
    collection(VARIABLES,[dvar]),
    length(VARIABLES,N),
    N>0,
    minimum_signature(VARIABLES,SIGNATURE,MIN),
    AUTOMATON=automaton(
        SIGNATURE,
        _43615,
        SIGNATURE,
        [source(s),sink(t)],
        [arc(s,0,s),arc(s,1,t),arc(t,1,t),arc(t,0,t)],
        [],
        [],
        []),
    automaton_bool(FLAG,[0,1,2],AUTOMATON).

minimum_signature([],[],_41889).

minimum_signature([[var-VAR]|VARs],[S|Ss],MIN) :-
    S in 0..2,
    MIN$<VAR#$<=>S#=0,
    MIN$=VAR#$<=>S#=1,
    MIN$>VAR#$<=>S#=2,
    minimum_signature(VARs,Ss,MIN).
B.230  minimum_except_0

◊ Meta-Data:

ctr_date(
    minimum_except_0,
    ['20030820','20040530','20041230','20060812','20090101']).

ctr_origin(minimum_except_0,'Derived from %c.',[minimum]).

ctr_arguments(
    minimum_except_0,
    ['MIN'-dvar,
     'VARIABLES'-collection(var-dvar),
     'DEFAULT'-int]).

ctr_restrictions(
    minimum_except_0,
    ['MIN'>0,
     'MIN'=<'DEFAULT',
     size('VARIABLES')>0,
     required('VARIABLES',var),
     'VARIABLES'\var>=0,
     'VARIABLES'\var=<'DEFAULT',
     'DEFAULT'>0]).

ctr_example(
    minimum_except_0,
    [minimum_except_0( 
      3,
      [[var-3],[var-7],[var-6],[var-7],[var-4],[var-7]],
      1000000),
    minimum_except_0( 
      2,
      [[var-3],[var-2],[var-0],[var-7],[var-2],[var-6]],
      1000000),
    minimum_except_0( 
      1000000,
      [[var-0],[var-0],[var-0],[var-0],[var-0],[var-0]],
      1000000))].

ctr_typical(
    minimum_except_0,
    [size('VARIABLES')>1,
     range('VARIABLES'\var)>1,
     atleast(1,'VARIABLES',0)]).
ctr_exchangeable(
    minimum_except_0,
    [items('VARIABLES',all),
    vals(['VARIABLES'\textasciitilde\textasciitilde var],int,\textasciitilde\textasciitilde =\textasciitilde\textasciitilde ,all,in)]).

ctr_graph(
    minimum_except_0,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1\textasciitilde var=\textasciitilde\textasciitilde =\textasciitilde\textasciitilde 0,
    variables2\textasciitilde var=\textasciitilde\textasciitilde =\textasciitilde\textasciitilde 0,
    variables1\textasciitilde key=variables2\textasciitilde key\textasciitilde\textasciitilde /
    variables1\textasciitilde var<variables2\textasciitilde var],
    ['ORDER'(0,'DEFAULT',var)=\textasciitilde\textasciitilde 'MIN'],
    []).

ctr_eval(
    minimum_except_0,
    [reformulation(minimum_except_0_r),
    automaton(minimum_except_0_a)]).

ctr_pure_functional_dependency(minimum_except_0,[]).

ctr_functional_dependency(minimum_except_0,1,[2,3])

minimum_except_0_r(MIN,VARIABLES,DEFAULT) :-
    check_type(int_gteq(1),DEFAULT),
    check_type(dvar(1,DEFAULT),MIN),
    collection(VARIABLES,[dvar(0,DEFAULT)]),
    length(VARIABLES,N),
    N>0,
    get_attr1(VARIABLES,VARS),
    minimum_except_01(VARS,ALLZEROS),
    call(ALLZEROS#=>MIN#=DEFAULT),
    append([0],VARS,VARS0),
    N1 is N+1,
    length(RANKS,N1),
    domain(RANKS,0,N),
    min_n1(VARS0,RANKS,MIN,1).

minimum_except_01([],1).

minimum_except_01([V|R],V\#=0#/\S) :-
    minimum_except_01(R,S).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

minimum_except_0_a(FLAG, MIN, VARIABLES, DEFAULT) :-
    check_type(int_gteq(1), DEFAULT),
    check_type(dvar(1, DEFAULT), MIN),
    collection(VARIABLES, [dvar(0, DEFAULT)]),
    length(VARIABLES, N),
    N>0,
    minimum_except_0_signature(
        VARIABLES,
        SIGNATURE,
        MIN,
        DEFAULT),
    AUTOMATON = automaton(
        SIGNATURE,
        _40221,
        SIGNATURE,
        [source(s), sink(j), sink(k)],
        [arc(s, 0, s),
        arc(s, 3, s),
        arc(s, 2, j),
        arc(s, 1, k),
        arc(j, 0, j),
        arc(j, 1, j),
        arc(j, 2, j),
        arc(j, 3, j),
        arc(k, 1, k)],
        [],
        [],
        []),
    automaton_bool(FLAG, [0, 1, 2, 3, 4], AUTOMATON).

minimum_except_0_signature([], [], _37817, _37818).

minimum_except_0_signature([[var-VAR] | VARs], [S | Ss], MIN, DEFAULT) :-
    S in 0..4,
    VAR# = 0#/MIN# = DEFAULT# => S#=0,
    VAR# = 0#/MIN# = DEFAULT# => S#=1,
    VAR# = 0#/MIN# = VAR# <= S#=2,
    VAR# = 0#/MIN# < VAR# <= S#=3,
    VAR# = 0#/MIN# > VAR# <= S#=4,
    minimum_except_0_signature(VARs, Ss, MIN, DEFAULT).
B.231 minimum_greater_than

◊ META-DATA:

ctr_date(minimum_greater_than, ['20030820', '20060812']).

ctr_origin(minimum_greater_than, 'N.˘Beldiceanu', []).

ctr_arguments(
    minimum_greater_than,
    ['VAR1'-dvar, 'VAR2'-dvar, 'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    minimum_greater_than,
    ['VAR1'> 'VAR2',
     size('VARIABLES')>0,
     required('VARIABLES', var)]).

ctr_example(
    minimum_greater_than,
    minimum_greater_than(5,
     3,
     [[var-8], [var-5], [var-3], [var-8]])).

ctr_typical(
    minimum_greater_than,
    [size('VARIABLES')>1, range('VARIABLES' var)>1]).

ctr_exchangeable(minimum_greater_than, [items('VARIABLES', all)]).

ctr_derived_collections(
    minimum_greater_than,
    [col('ITEM'-collection(var-dvar), [item(var='VAR2')])]).

ctr_graph(
    minimum_greater_than,
    ['ITEM', 'VARIABLES'],
    2,
    ['PRODUCT' >> collection(item, variables)],
    [item var < variables var],
    ['NARC'>0],
    [],
    ['SUCC' >> [source, variables]],
    [minimum('VAR1', variables)]).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

\begin{verbatim}
ctr_eval(
    minimum_greater_than,
    [reformulation(minimum_greater_than_r),
     automaton(minimum_greater_than_a)]).

ctr_aggregate(minimum_greater_than,[],[min,id,union]).

minimum_greater_than_r(VAR1,VAR2,VARIABLES) :-
    check_type(dvar,VAR1),
    check_type(dvar,VAR2),
    collection(VARIABLES,[dvar]),
    length(VARIABLES,N),
    N>0,
    get_attr1(VARIABLES,VARS),
    maximum(MAX,VARS),
    VAR1#>VAR2,
    VAR1#=<MAX,
    minimum_greater_than1(VARS,VAR2,MAX,UARS),
    minimum(VAR1,UARS).

minimum_greater_than1([],_31290,_31291,[]).

minimum_greater_than1([V|R],VAR2,MAX,[U|S]) :-
    fd_min(V,Min),
    fd_max(MAX,Max),
    U in Min..Max,
    V#=<VAR2#=>U#=MAX,
    V#>VAR2#=>U#=V,
    minimum_greater_than1(R,VAR2,MAX,S).

minimum_greater_than_a(FLAG,VAR1,VAR2,VARIABLES) :-
    check_type(dvar,VAR1),
    check_type(dvar,VAR2),
    collection(VARIABLES,[dvar]),
    length(VARIABLES,N),
    N>0,
    VAR1#>VAR2,
    minimum_greater_than_signature(
        VARIABLES,
        SIGNATURE,
        VAR1,
        VAR2),
    AUTOMATON=
    automaton(
        SIGNATURE,
        _33851,
        _33851,
        _33851,
        _33851).
\end{verbatim}
SIGNATURE,
[sourc(s),sink(t)],
[arc(s,0,s),
 arc(s,1,s),
 arc(s,2,s),
 arc(s,5,s),
 arc(s,4,t),
 arc(t,0,t),
 arc(t,1,t),
 arc(t,2,t),
 arc(t,4,t),
 arc(t,5,t)],
[],
[],
[]),
automaton_bool(FLAG,[0,1,2,3,4,5],AUTOMATON).

minimum_greater_than_signature([],[],_31291,_31292).

minimum_greater_than_signature(
 [[var-VAR]|VARs],
 [S|Ss],
 VAR1,
 VAR2) :-
 S in 0..5,
 VAR#<VAR1#/\VAR#=<VAR2#<=S#=0,
 VAR#=VAR1#/\VAR#=<VAR2#<=S#=1,
 VAR#>VAR1#/\VAR#=<VAR2#<=S#=2,
 VAR#<VAR1#/\VAR#>VAR2#<=S#=3,
 VAR#=VAR1#/\VAR#>VAR2#<=S#=4,
 VAR#>VAR1#/\VAR#>VAR2#<=S#=5,
 minimum_greater_than_signature(VARs,Ss,VAR1,VAR2).
B.232  minimum_modulo

◊ Meta-Data:

ctr_date(
  minimum_modulo,
  ['20000128', '20030820', '20041230', '20060812']).

date(origin(minimum_modulo, 'Derived from %c.', [minimum]).

ctr_arguments(
  minimum_modulo,
  ['MIN'-dvar,'VARIABLES'-collection(var-dvar),'M'-int]).

ctr_restrictions(
  minimum_modulo,
  [size('VARIABLES')>0,'M'>0,required('VARIABLES', var))].

ctr_example(
  minimum_modulo,
  [minimum_modulo(
    6,
    [[var-9],[var-1],[var-7],[var-6],[var-5]],
    minimum_modulo(
      9,
      [[var-9],[var-1],[var-7],[var-6],[var-5]],
      3))].

ctr_typical(
  minimum_modulo,
  [size('VARIABLES')>1,
   range('VARIABLES`var)>1,
   'M'>1,
   'M'<maxval('VARIABLES`var])].

ctr_exchangeable(minimum_modulo, [items('VARIABLES', all)])].

ctr_graph(
  minimum_modulo,
  ['VARIABLES'],
  2,
  ['CLIQUE']>>collection(variables1,variables2)],
  [variables1`key=variables2`key#/
    variables1`var mod `M'<variables2`var mod 'M'],
  ['ORDER'(0,'MAXINT',var)=`MIN'])}


\[
[]). \\
ctr\_pure\_functional\_dependency(minimum\_modulo, []). \\
ctr\_functional\_dependency(minimum\_modulo, 1, [2, 3]).
\]
B.233  minimum_weight_alldifferent

◊ Meta-Data:

\begin{verbatim}
ctr_date(
   minimum_weight_alldifferent,
   ['20030820','20040530','20060812']).

ctr_origin(
   minimum_weight_alldifferent,
   \cite{FocacciLodiMilano99},
   []).

ctr_synonyms(
   minimum_weight_alldifferent,
   [minimum_weight_alldiff,
    minimum_weight_alldistinct,
    min_weight_alldiff,
    min_weight_alldifferent,
    min_weight_alldistinct]).

ctr_arguments(
   minimum_weight_alldifferent,
   ['VARIABLES'-collection(var-dvar),
    'MATRIX'-collection(i-int,j-int,c-int),
    'COST'-dvar]).

ctr_restrictions(
   minimum_weight_alldifferent,
   [size('VARIABLES')>0,
    required('VARIABLES',var),
    'VARIABLES'\^var>=1,
    'VARIABLES'\^var=<size('VARIABLES'),
    required('MATRIX',[i,j,c]),
    increasing_seq('MATRIX',[i,j]),
    'MATRIX'\^i>=1,
    'MATRIX'\^i=<size('VARIABLES'),
    'MATRIX'\^j>=1,
    'MATRIX'\^j=<size('VARIABLES'),
    size('MATRIX')=size('VARIABLES')*size('VARIABLES'))].

ctr_example(
   minimum_weight_alldifferent,
   minimum_weight_alldifferent(
      [[var-2],[var-3],[var-1],[var-4]],
      [[i-1,j-1,c-4],
      ]).
\end{verbatim}
[i-1, j-2, c-1],
[i-1, j-3, c-7],
[i-1, j-4, c-0],
[i-2, j-1, c-1],
[i-2, j-2, c-0],
[i-2, j-3, c-8],
[i-2, j-4, c-2],
[i-3, j-1, c-3],
[i-3, j-2, c-2],
[i-3, j-3, c-1],
[i-3, j-4, c-6],
[i-4, j-1, c-0],
[i-4, j-2, c-0],
[i-4, j-3, c-6],
[i-4, j-4, c-5],
17}).

ctr_typical(
    minimum_weight_alldifferent,
    [size('VARIABLES')>1,range('MATRIX`c)>1,'MATRIX`c>0]).

ctr_graph(
    minimum_weight_alldifferent,
    ['VARIABLES'],
    2,
    ['CLIQUE']>>collection(variables1,variables2),
    [variables1`var=variables2`key],
    ['NTREE'=0,
     'SUM_WEIGHT_ARC'(
         MATRIX@
         ((variables1`key-1)*size('VARIABLES')+variables1`var)^c)=
         COST],
    []).

ctr_functional_dependency(minimum_weight_alldifferent,3,[1,2]).
B.234 multi_global_contiguity

◊ Meta-Data:

ctr_predefined(multi_global_contiguity).

ctr_date(multi_global_contiguity,[’20120212’]).

ctr_origin(
    multi_global_contiguity,
    Derived from %c.,
    [global_contiguity]).

ctr_synonyms(multi_global_contiguity,[multi_contiguity]).

ctr_arguments(
    multi_global_contiguity,
    [’VARIABLES’-collection(var-dvar)]).

ctr_restrictions(
    multi_global_contiguity,
    [required(’VARIABLES’,var),’VARIABLES’~var>=0]).

ctr_example(
    multi_global_contiguity,
    multi_global_contiguity(
        [[var-0],
         [var-2],
         [var-2],
         [var-1],
         [var-1],
         [var-0],
         [var-0],
         [var-5]])).

ctr_typical(
    multi_global_contiguity,
    [size(’VARIABLES’)>2,range(’VARIABLES’~var)>2]).

ctr_exchangeable(
    multi_global_contiguity,
    [items(’VARIABLES’,reverse)]).

ctr_eval(
    multi_global_contiguity,
    [reformulation(multi_global_contiguity_c)]).
ctr_contractible(multi_global_contiguity,[],'VARIABLES',any).

multi_global_contiguity_c([]) :- !.

multi_global_contiguity_c(VARIABLES) :-
    collection(VARIABLES,[int_gteq(0)]),
    get_kattr1(VARIABLES,1,VARKEYS),
    sort(VARKEYS,SVARKEYS),
    multi_global_contiguity_c1(SVARKEYS).

multi_global_contiguity_c1([]) :- !.

multi_global_contiguity_c1([_13208]) :- !.

multi_global_contiguity_c1([0-_13212|R]) :- !,
    multi_global_contiguity_c1(R).

multi_global_contiguity_c1([I-P,I-Q|R]) :- !,
    Q is P+1,
    multi_global_contiguity_c1([I-Q|R]).

multi_global_contiguity_c1([_13208,J-Q|R]) :-
    multi_global_contiguity_c1([J-Q|R]).
B.235 multi_inter_distance

◊ **Meta-Data:**

```prolog
ctr_predefined(multi_inter_distance).

ctr_date(multi_inter_distance,['20110814']).

ctr_origin(multi_inter_distance,'\cite{OuelletQuimper11}',[]).

ctr_synonyms(
    multi_inter_distance,
    [multi_all_min_distance,
     multi_all_min_dist,
     sliding_atmost,
     atmost_sliding]).

ctr_arguments(
    multi_inter_distance,
    ['VARIABLES'-collection(var-dvar),'LIMIT'-int,'DIST'-int]).

ctr_restrictions(
    multi_inter_distance,
    [required('VARIABLES',var), 'LIMIT'>0,'DIST'>0]).

ctr_example(
    multi_inter_distance,
    multi_inter_distance(
        [[var-4],[var-0],[var-9],[var-4],[var-7]],
        2,
        3)).

ctr_typical(
    multi_inter_distance,
    ['LIMIT'>1,
     'LIMIT'<size('VARIABLES'),
     'DIST'>1,
     'DIST'<range('VARIABLES'\^var)]).

ctr_exchangeable(
    multi_inter_distance,
    [items('VARIABLES',all),
     translate(['VARIABLES'\^var]),
     vals(['LIMIT'],int,<,dontcare,dontcare),
     vals(['MINDIST'],int>(=1),>,dontcare,dontcare)]).
```
ctr_eval(
    multi_inter_distance,
    [reformulation(multi_inter_distance_r)]).

ctr_contractible(multi_inter_distance,[],’VARIABLES’,any).

multi_inter_distance_r([],LIMIT,DIST) :-
    !,
    integer(LIMIT),
    integer(DIST),
    LIMIT>0,
    DIST>0.

multi_inter_distance_r(VARIABLES,LIMIT,DIST) :-
    collection(VARIABLES,[dvar]),
    integer(LIMIT),
    integer(DIST),
    LIMIT>0,
    DIST>0,
    get_attr1(VARIABLES,ORIGINS),
    length(VARIABLES,N),
    length(DURATIONS,N),
    length(ENDS,N),
    length(HEIGHTS,N),
    domain(DURATIONS,DIST,DIST),
    domain(HEIGHTS,1,1),
    ori_dur_end(ORIGINS,DURATIONS,ENDS),
    gen_cum_tasks(ORIGINS,DURATIONS,ENDS,HEIGHTS,1,Tasks),
    cumulative(Tasks,[limit(LIMIT)]).
B.236  nand

♦ Meta-Data:

ctr_date(nand, ['20051226']).

ctr_origin(nand, 'Logic', []).

ctr_synonyms(nand, [clause]).

ctr_arguments(nand, ['VAR'-dvar, 'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(nand, ['VAR'>=0, 'VAR'=<1, size('VARIABLES')>=2, required('VARIABLES', var), 'VARIABLES'~var>=0, 'VARIABLES'~var=<1]).

ctr_example(nand, [nand(1, [[var-0], [var-0]]), nand(1, [[var-0], [var-1]]), nand(1, [[var-1], [var-0]]), nand(0, [[var-1], [var-1]]), nand(1, [[var-1], [var-0], [var-1]])].

ctr_exchangeable(nand, [items('VARIABLES', all)]).

ctr_eval(nand, [automaton(nand_a)]).

ctr_pure_functional_dependency(nand, []).

ctr_functional_dependency(nand, 1, [2]).

ctr_contractible(nand, ['VAR'=0, 'VARIABLES', any]).

ctr_extensible(nand, ['VAR'=1, 'VARIABLES', any]).

ctr_aggregate(nand, [], [‘#’, ‘\’, ‘union’]).

nand_a(FLAG, VAR, VARIABLES) :-
check_type(dvar(0,1),VAR),
collection(VARIABLES,[dvar(0,1)]),
length(VARIABLES,L),
L>1,
get_attr1(VARIABLES,LIST),
append([VAR],LIST,LIST_VARIABLES),
AUTOMATON=
automaton(
    LIST_VARIABLES,
    _20905,
    LIST_VARIABLES,
    [source(s),sink(j),sink(k)],
    [arc(s,1,i),
      arc(s,0,j),
      arc(i,0,k),
      arc(i,1,i),
      arc(k,0,k),
      arc(k,1,k),
      arc(j,1,j)],
    [],
    [],
    []),
automaton_bool(FLAG,[0,1],AUTOMATON).
B.237 nclass

◊ Meta-Data:

\[
\text{ctr\_date(nclass, ['200000128', '20030820', '20060812'])}.
\]

\[
\text{ctr\_origin(nclass, 'Derived from %c.', [nvalue])}.
\]

\[
\text{ctr\_types(nclass, ['VALUES'-collection(val-int)])}.
\]

\[
\text{ctr\_arguments(nclass, ['NCLASS'-dvar, 'VARIABLES'-collection(var-dvar), 'PARTITIONS'-collection(p-'VALUES')])}.
\]

\[
\text{ctr\_restrictions(nclass, [size('VALUES')>=1, required('VALUES', val), distinct('VALUES', val), 'NCLASS'>=0, 'NCLASS'=<min(size('VARIABLES'), size('PARTITIONS'))), 'NCLASS'=<range('VARIABLES'\^\var), required('VARIABLES', \var), required('PARTITIONS', p), size('PARTITIONS')>=2])}.
\]

\[
\text{ctr\_example(nclass, nclass(2, [[\var-3], [\var-2], [\var-7], [\var-2], [\var-6]], [[p-[[\val-1], [\val-3]]], [p-[[\val-4]]], [p-[[\val-2], [\val-6]]]]))}.
\]

\[
\text{ctr\_typical(nclass, ['NCLASS'>1, 'NCLASS'<size('VARIABLES'), 'NCLASS'<range('VARIABLES'\^\var), size('VARIABLES')>size('PARTITIONS')])}.
\]

\[
\text{ctr\_exchangeable(nclass,)}
\]
items('VARIABLES',all),
items('PARTITIONS',all),
items('PARTITIONS'\p,all),
vals(
    ['VARIABLES'\var],
    part('PARTITIONS'),
    =,
    dontcare,
    dontcare),
vals(
    ['VARIABLES'\var,'PARTITIONS'\p\val],
    int,
    =\=,
    all,
    dontcare)).

ctr_graph(
    nclass,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [in_same_partition(
        variables1\var,
        variables2\var,
        PARTITIONS)],
    ['NSCC'='NCLASS'],
    []).

ctr_pure_functional_dependency(nclass,[]).

ctr_functional_dependency(nclass,1,[2,3]).

ctr_extensible(
    nclass,
    ['NCLASS'=size('PARTITIONS')],
    VARIABLES,
    any).
B.238  neq

◊ **Meta-Data:**

- `ctr_predefined(neq).`
- `ctr_date(neq,['20070821']).`
- `ctr_origin(neq,'Arithmetic.',[]).`
- `ctr_synonyms(neq,[rel]).`
- `ctr_arguments(neq,['VAR1'-dvar,'VAR2'-dvar]).`
- `ctr_example(neq,neq(1,8)).`
- `ctr_exchangeable(
  neq,
  [args([['VAR1','VAR2']]),
   vals([['VAR1','VAR2'],int,=\=,all,dontcare])].

- `ctr_eval(neq,[builtin(neq_b)]).`

- `neq_b(VAR1,VAR2) :-
    check_type(dvar,VAR1),
    check_type(dvar,VAR2),
    VAR1\=\VAR2.`
B.239   neq_cst

◊ **META-DATA:**

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctr_predefined(neq_cst)</td>
<td></td>
</tr>
<tr>
<td>ctr_date(neq_cst,['20090923'])</td>
<td></td>
</tr>
<tr>
<td>ctr_origin(neq_cst,'Arithmetic.',[])</td>
<td></td>
</tr>
<tr>
<td>ctr_arguments(neq_cst,['VAR1'-dvar,'VAR2'-dvar,'CST2'-int])</td>
<td></td>
</tr>
<tr>
<td>ctr_example(neq_cst,neq_cst(8,2,7))</td>
<td></td>
</tr>
<tr>
<td>ctr_typical(neq_cst,['CST2'='=0,'VAR1'='=VAR2'+CST2'])</td>
<td></td>
</tr>
</tbody>
</table>
| ctr_exchangeable(neq_cst,neq_cst) | [args([[VAR1],[VAR2,CST2]]),
                       translate([VAR1,VAR2]),
                       translate([VAR1,CST2])] |                                    |
| ctr_eval(neq_cst, [builtin(neq_cst_b)]) |                                    |

neq_cst_b(VAR1,VAR2,CST2) :-
    check_type(dvar,VAR1),
    check_type(dvar,VAR2),
    check_type(int,CST2),
    VAR1\=VAR2+CST2.
B.240 nequivalence

♦ Meta-Data:

ctr_date(nequivalence,['20000128','20030820','20060812']).

ctr_origin(nequivalence,'Derived from %c.',[nvalue]).

ctr_arguments(
    nequivalence,
    ['NEQUIV'-dvar,'M'-int,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    nequivalence,
    [required('VARIABLES',var),
     'NEQUIV'>=min(1,size('VARIABLES')),
     'NEQUIV'=<min('M',size('VARIABLES')),
     'NEQUIV'=<range('VARIABLES'\^var),
     'M'>0]).

ctr_example(
    nequivalence,
    nequivalence(2,3,
            [[var-3],
             [var-2],
             [var-5],
             [var-6],
             [var-15],
             [var-3],
             [var-3]]]).

ctr_typical(
    nequivalence,
    ['NEQUIV'>1,
     'NEQUIV'<size('VARIABLES'),
     'NEQUIV'=<range('VARIABLES'\^var),
     'M'>1,
     'M'<maxval('VARIABLES'\^var)]).

ctr_exchangeable(
    nequivalence,
    [items('VARIABLES',all),
     vals(['VARIABLES'\^var],mod('M'),=,dontcare,dontcare)]).
ctr_graph(  
    nequivalence,  
    ['VARIABLES'],  
    2,  
    ['CLIQUE'>>collection(variables1,variables2)],  
    [variables1\var mod 'M'=variables2\var mod 'M'],  
    ['NSCC'='NEQUIV'],  
    []).  

ctr_pure_functional_dependency(nequivalence,[]).

ctr_functional_dependency(nequivalence,1,[2,3]).

ctr_contractible(  
    nequivalence,  
    ['NEQUIV'=1,size('VARIABLES')>0],  
    VARIABLES,  
    any).

ctr_contractible(  
    nequivalence,  
    ['NEQUIV'=size('VARIABLES')],  
    VARIABLES,  
    any).

ctr_extensible(nequivalence,['NEQUIV'='M'],['VARIABLES',any].
B.241  next_element

◊ Meta-Data:

ctr_date(next_element,[’20030820’,’20040530’,’20060812’]).

ctr_origin(next_element,’N.˘Beldiceanu’,[]).

ctr_arguments(
    next_element,
    [’THRESHOLD’-dvar,
     ’INDEX’-dvar,
     ’TABLE’-collection(index-int,value-dvar),
     ’VAL’-dvar]).

ctr_restrictions(
    next_element,
    [’INDEX’>=1,
     ’INDEX’<=size(’TABLE’),
     ’THRESHOLD’<’INDEX’,
     required(’TABLE’,[index,value]),
     size(’TABLE’)>0,
     ’TABLE’^index>=1,
     ’TABLE’^index<=size(’TABLE’),
     distinct(’TABLE’,index)]).

ctr_example(
    next_element,
    next_element(2,
      3,
      [[index-1,value-1],
       [index-2,value-8],
       [index-3,value-9],
       [index-4,value-5],
       [index-5,value-9]],
    9)).

ctr_typical(
    next_element,
    [size(’TABLE’)>1,range(’TABLE’^value)>1]).

ctr_derived_collections(
    next_element,
    [col(’ITEM’-collection(index-dvar,value-dvar),
      [item(index–’THRESHOLD’,value–’VAL’)])].)
ctr_graph(
    next_element,
    ['ITEM','TABLE'],
    2,
    ['PRODUCT'>>collection(item,table)],
    [item^index<table^index, item^value=table^value],
    ['NARC'>0],
    [],
    [SUCC>>
        [source,
            variables-
            col('VARIABLES'-collection(var-dvar),
                [item(var-'TABLE'^index)])],
        [minimum('INDEX',variables)]).

ctr_eval(next_element,[automaton(next_element_a)]).

next_element_a(FLAG,THRESHOLD,INDEX,TABLE,VAL) :-
    length(TABLE,N),
    N>0,
    check_type(dvar,THRESHOLD),
    check_type(dvar(1,N),INDEX),
    collection(TABLE,[int(1,N),dvar]),
    check_type(dvar,VAL),
    THRESHOLD#<INDEX,
    get_attr1(TABLE,INDEXES),
    all_different(INDEXES),
    next_element_signature(
        TABLE,
        SIGNATURE,
        THRESHOLD,
        INDEX,
        VAL),
    AUTOMATON=
    automaton(
        SIGNATURE,
        _35603,
        SIGNATURE,
        [source(s),sink(t)],
        [arc(s,0,s),
            arc(s,1,s),
            arc(s,2,s),
            arc(s,3,s),
            arc(s,4,s),
            arc(s,5,s),
            arc(s,6,s),
            arc(s,7,s),
            arc(s,8,s),
            arc(s,9,s)])
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

```prolog
arc(s,7,s),
arc(s,9,s),
arcs,10,s),
arcs,11,s),
arcs,8,t),
arcs,10,t),
arcs,11,t),
[],
[]),
automaton_bool(  
  FLAG,  
  [0,1,2,3,4,5,6,7,8,9,10,11],
  AUTOMATON).

next_element_signature([],[],_32135,_32136,_32137).

next_element_signature(
  [[index-I,value-V]|Ts],
  [S|Ss],
  THRESHOLD,
  INDEX,
  VAL) :-
  S in 0..11,
  I#=<THRESHOLD#/\I#<INDEX#/\V#=VAL#<=S#=0,
  I#=<THRESHOLD#/\I#<INDEX#/\V#=VAL#==>S#=1,
  I#=<THRESHOLD#/\I#=INDEX#/\V#=VAL#<=S#=2,
  I#=<THRESHOLD#/\I#=INDEX#/\V#=VAL#==>S#=3,
  I#=<THRESHOLD#/\I#>INDEX#/\V#=VAL#<=S#=4,
  I#=<THRESHOLD#/\I#>INDEX#/\V#=VAL#==>S#=5,
  I#=<THRESHOLD#/\I#<INDEX#/\V#=VAL#<=S#=6,
  I#>=THRESHOLD#/\I#<INDEX#/\V#=VAL#<=S#=7,
  I#>=THRESHOLD#/\I#=INDEX#/\V#=VAL#<=S#=8,
  I#>=THRESHOLD#/\I#=INDEX#/\V#=VAL#<=S#=9,
  I#>=THRESHOLD#/\I#>INDEX#/\V#=VAL#<=S#=10,
  I#>=THRESHOLD#/\I#>INDEX#/\V#=VAL#<=S#=11,
next_element_signature(Ts,Ss,THRESHOLD,INDEX,VAL).
```
B.242  next_greater_element

◊ Meta-Data:

ctr_date(
    next_greater_element,
    ['20030820', '20040530', '20060812']).

ctr_origin(next_greater_element, 'M.˚Carlsson', []).

ctr_arguments(
    next_greater_element,
    ['VAR1'-dvar, 'VAR2'-dvar, 'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    next_greater_element,
    ['VAR1'<'VAR2',
     size('VARIABLES')>0,
     required('VARIABLES', var)]).

ctr_example(
    next_greater_element,
    next_greater_element(7,
    8,
    [[var-3], [var-5], [var-8], [var-9]])).

ctr_typical(
    next_greater_element,
    [size('VARIABLES')>1, range('VARIABLES'~var)>1]).

ctrDerivedCollections(
    next_greater_element,
    [col('V'-collection(var-dvar), [item(var~'VAR1')])]).

ctr_graph(
    next_greater_element,
    ['VARIABLES'],
    2,
    ['PATH'>collection(variables1, variables2)],
    [variables1~var<variables2~var],
    ['NARC'=size('VARIABLES')-1],
    []).
[‘V’, ‘VARIABLES’],
2,
[‘PRODUCT’ >>= collection(v, variables)],
[v `var < variables `var],
[‘NARC’ > 0],
[],
[‘SUCC’ >>= [source, variables]],
[minimum(‘VAR2’, variables)].

ctr_eval(
    next_greater_element,
    [reformulation(next_greater_element_r)]).
	next_greater_element_r(VAR1, VAR2, VARIABLES) :-
    check_type(dvar, VAR1),
    check_type(dvar, VAR2),
    collection(VARIABLES, [dvar]),
    length(VARIABLES, N),
    N > 0,
    get_attr1(VARIABLES, VARS),
    maximum(MAX, VARS),
    VAR2 `#> VAR1,
    VAR2 `#=< MAX,
    next_greater_element1(VARS, VAR1, MAX, UARS),
    minimum(VAR2, UARS).
	next_greater_element1([V], VAR1, MAX, [U]) :-
    !,
    fd_min(V, Min),
    fd_max(MAX, Max),
    U in Min..Max,
    V `=#< VAR1 `#=> U `#=< MAX,
    V `#> VAR1 `#=> U `#= V.
	next_greater_element1([V1, V2 | R], VAR1, MAX, [U1 | S]) :-
    V1 `#< V2,
    fd_min(V1, Min),
    fd_max(MAX, Max),
    U1 in Min..Max,
    V1 `=#< VAR1 `#=> U1 `#=< MAX,
    V1 `#> VAR1 `#=> U1 `#= V1,
    next_greater_element1([V2 | R], VAR1, MAX, S).
B.243 ninterval

◇ Meta-Data:

ctr_date(ninterval,[‘20030820’,‘20040530’,‘20060812’]).

ctr_origin(ninterval,’Derived from %c.’,[nvalue]).

ctr_arguments(ninterval,
    [’NVAL’-dvar,
     ’VARIABLES’-collection(var-dvar),
     ’SIZE_INTERVAL’-int]).

ctr_restrictions(ninterval,
    [’NVAL’>=min(1,size(’VARIABLES’)),
     ’NVAL’=<size(’VARIABLES’),
     required(’VARIABLES’,var),
     ’SIZE_INTERVAL’>0]).

ctr_example(ninterval,
    ninterval(2,[[var-3],[var-1],[var-9],[var-1],[var-9]],4)).

ctr_typical(ninterval,
    [’NVAL’>1,
     ’NVAL’<size(’VARIABLES’),
     ’SIZE_INTERVAL’>1,
     ’SIZE_INTERVAL’<range(’VARIABLES’^var)]).

ctr_exchangeable(ninterval,
    [items(’VARIABLES’,all),
     vals([’VARIABLES’^var],
          intervals(’SIZE_INTERVAL’),
          =,
          dontcare,
          dontcare)]).

ctr_graph(ninterval,
    [’VARIABLES’],
    2,
['CLIQUE'>>collection(variables1,variables2)],
[variables1\var/'SIZE_INTERVAL'= variables2\var/'SIZE_INTERVAL'],
['NSCC'='NVAL']
[]).

ctr_pure_functional_dependency(ninterval,[]).

ctr_functional_dependency(ninterval,1,[2,3]).

ctr_contractible(ninterval,
['NVAL'=1, size('VARIABLES')>0],
VARIABLES,
any).

ctr_contractible(ninterval,
['NVAL'=size('VARIABLES')],
VARIABLES,
any).
B.244 no_peak

Meta-Data:

ctr_date(no_peak, ['20031101', '20040530']).

ctr_origin(no_peak, 'Derived from %c.', [peak]).

ctr_arguments(no_peak, ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    no_peak,
    [size('VARIABLES')>0, required('VARIABLES', var)]).

ctr_example(
    no_peak,
    no_peak([[var-1], [var-1], [var-4], [var-8], [var-8]])).

ctr_typical(
    no_peak,
    [size('VARIABLES')>2, range('VARIABLES' ^ var)>1]).

ctr_exchangeable(
    no_peak,
    [items('VARIABLES', reverse), translate(['VARIABLES' ^ var])]).

ctr_eval(no_peak, [automaton(no_peak_a)]).

ctr_contractible(no_peak, [], 'VARIABLES', any).

no_peak_a(FLAG, VARIABLES) :-
    collection(VARIABLES, [dvar]),
    length(VARIABLES, N),
    N>0,
    no_peak_signature(VARIABLES, SIGNATURE),
    AUTOMATON = automaton(
        SIGNATURE,
        _16572,
        SIGNATURE,
        [source(s), sink(t), sink(s)],
        [arc(s, 1, s),
         arc(s, 2, s),
         arc(s, 0, t),
         arc(t, 0, t),
         arc(t, 1, t)],
         any).

[\],
[\],
[\],
automaton_bool(FLAG,[0,1,2],AUTOMATON).

no_peak_signature([],[]).

no_peak_signature([_14981],[]) :- !.

no_peak_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss]) :-
  S in 0..2,
  VAR1#<VAR2#<==>S#=0,
  VAR1#=VAR2#<==>S#=1,
  VAR1#>VAR2#<==>S#=2,
  no_peak_signature([[var-VAR2]|VARs],Ss).
B.245  no_valley

◇ Meta-Data:

ctr_date(no_valley, ['20031101', '20040530']).

ctr_origin(no_valley, 'Derived from %c.', [valley]).

ctr_arguments(no_valley, ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
  no_valley,
  [size('VARIABLES')>0, required('VARIABLES', var)]).

ctr_example(
  no_valley,
  no_valley(
    [[var-1], [var-1], [var-4], [var-8], [var-8], [var-2]]).

ctr_typical(
  no_valley,
  [size('VARIABLES')>2, range('VARIABLES' var)>1]).

ctr_exchangeable(
  no_valley,
  [items('VARIABLES', reverse), translate(['VARIABLES' var])]).

ctr_eval(no_valley, [automaton(no_valley_a)]).

ctr_contractible(no_valley, [], 'VARIABLES', any).

no_valley_a(FLAG, VARIABLES) :-
  collection(VARIABLES, [dvar]),
  length(VARIABLES, N),
  N>0,
  no_valley_signature(VARIABLES, SIGNATURE),
  AUTOMATON =
  automaton(
    SIGNATURE, 17219,
    SIGNATURE,
    [source(s), sink(t), sink(s)],
    [arc(s, 0, s), arc(s, 1, s), arc(s, 2, t), arc(t, 1, t),
arc(t,2,t],
[],
[],
[]),
automaton_bool(FLAG,[0,1,2],AUTOMATON).

no_valley_signature([],[]).

no_valley_signature([-15628],[]) :- !.

no_valley_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss]) :-
S in 0..2,
VAR1#<VAR2#<=>S#=0,
VAR1#=VAR2#<=>S#=1,
VAR1#>VAR2#<=>S#=2,
no_valley_signature([[var-VAR2]|VARs],Ss).
B.246 non_overlap_sboxes

◊ **Meta-Data:**

```prolog
ctr_date(non_overlap_sboxes, ['20070622', '20090725']).

ctr_origin(
    non_overlap_sboxes,
    Geometry, derived from \cite{BeldiceanuCarlssonPoderSadekTruchet07}, []).

ctr_synonyms(non_overlap_sboxes, [non_overlap, non_overlapping]).

ctr_types(
    non_overlap_sboxes,
    ['VARIABLES'-collection(v-dvar),
     'INTEGERS'-collection(v-int),
     'POSITIVES'-collection(v-int)]).

ctr_arguments(
    non_overlap_sboxes,
    ['K'-int,
     'DIMS'-sint,
     'OBJECTS'-collection(oid-int, sid-int, x-'VARIABLES'),
     'SBOXES'-collection(sid-int, t-'INTEGERS', l-'POSITIVES')]).

ctr_restrictions(
    non_overlap_sboxes,
    [size('VARIABLES')>=1,
     size('INTEGERS')>=1,
     size('POSITIVES')>=1,
     required('VARIABLES', v),
     size('VARIABLES')='K',
     required('INTEGERS', v),
     size('INTEGERS')='K',
     required('POSITIVES', v),
     size('POSITIVES')='K',
     'POSITIVES'\ v>0,
     'K'>0,
     'DIMS'>=0,
     'DIMS'<'K',
     increasing_seq('OBJECTS', [oid]),
     required('OBJECTS', [oid, sid, x]),
     'OBJECTS'\ oid>1,
     'OBJECTS'\ oid=<size('OBJECTS'),
     'OBJECTS'\ sid>1,
     ...]
)
'OBJECTS'~sid=<size('SBOXES'),
required('SBOXES',[sid,t,l]),
'SBOXES'~sid>=1,
'SBOXES'~sid=<size('SBOXES'))).

ctr_example(
  non_overlap_sboxes,
  non_overlap_sboxes(2,
    {0,1},
    [[oid-1,sid-1,x-[[v-4],[v-1]],
      [oid-2,sid-3,x-[[v-2],[v-2]],
      [oid-3,sid-4,x-[[v-5],[v-4]]],
      [[sid-1,t-[[v-0],[v-0]],l-[[v-1],[v-1]]],
      [sid-1,t-[[v-1],[v-0]],l-[[v-1],[v-1]]],
      [sid-1,t-[[v-2],[v-1]],l-[[v-1],[v-1]]],
      [sid-2,t-[[v-0],[v-0]],l-[[v-3],[v-1]]],
      [sid-2,t-[[v-0],[v-1]],l-[[v-1],[v-1]]],
      [sid-2,t-[[v-2],[v-1]],l-[[v-1],[v-1]]],
      [sid-3,t-[[v-0],[v-0]],l-[[v-1],[v-2]]],
      [sid-4,t-[[v-0],[v-0]],l-[[v-1],[v-1]]])).

ctr_typical(non_overlap_sboxes,[size('OBJECTS')>1]).

ctr_exchangeable(
  non_overlap_sboxes,
  [items('OBJECTS',all),
   items('SBOXES',all),
   items_sync('OBJECTS'~x,'SBOXES'~t,'SBOXES'~l,all),
   vals([`SBOXES'~l`\`v],int(>=(1)),>,dontcare,dontcare)].

ctr_eval(non_overlap_sboxes,[logic(non_overlap_sboxes_g)]).

ctr_logic(
  non_overlap_sboxes,
  [DIMENSIONS,OIDS],
  [(origin(O1,S1,D)---O1\x(D)+S1\t(D)),
    (end(O1,S1,D)---O1\x(D)+S1\t(D)+S1\l(D)),
    (non_overlap_sboxes(Dims,O1,S1,O2,S2)---
      exists(D,
        Dims,
        end(O1,S1,D)<=origin(O2,S2,D)\/
        end(O2,S2,D)<=origin(O1,S1,D)),
      (non_overlap_objects(Dims,O1,O2)---
        forall(}
S1,  
sboxes([O1^sid]),
forall(
  S2,  
sboxes([O2^sid]),
  non_overlap_sboxes(Dims,O1,S1,O2,S2)),
(all_non_overlap(Dims,OIDS)--->
forall(
  O1,  
objects(OIDS),
  forall(
    O2,  
objects(OIDS),
    O1^oid#<O2^oid#=>
    non_overlap_objects(Dims,O1,O2))),
all_non_overlap(DIMENSIONS,OIDS)).

ctr_contractible(non_overlap_sboxes,[],'OBJECTS',suffix).

non_overlap_sboxes_g(K,_32773,[],_32775) :-
  !,
  check_type(int_gteq(1),K).

non_overlap_sboxes_g(K,_DIMS,OBJECTS,SBOXES) :-
  length(OBJECTS,O),
  length(SBOXES,S),
  O>0,
  S>0,
  check_type(int_gteq(1),K),
  collection(OBJECTS,[int(1,O),dvar(1,S),col(K,[dvar])]),
  collection(
    SBOXES,  
    [int(1,S),col(K,[int]),col(K,[int_gteq(1)])]),
  get_attr1(OBJECTS,OIDS),
  get_attr2(OBJECTS,SIDS),
  get_col_attr3(OBJECTS,1,XS),
  get_attr1(SBOXES,SIDES),
  get_attr2(SBOXES,1,TS),
  get_col_attr3(SBOXES,1,TL),
  collection_increasing_seq(OBJECTS,[1]),
  geost1(OIDS,SIDS,XS,Objects),
  geost2(SIDES,TS,TL,Sboxes),
  geost_dims(1,K,DIMENSIONS),
  ctr_logic(non_overlap_sboxes,[DIMENSIONS,OIDS],Rules),
  geost(Objects,Sboxes,[overlap(true)],Rules).
B.247 nor

◊ META-DATA:

ctr_date(nor, ['20051226']).

ctr_origin(nor, 'Logic', []).

ctr_synonyms(nor, [clause]).

ctr_arguments(
    nor,
    ['VAR'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    nor,
    ['VAR'>=0, 'VAR'=<1, size('VARIABLES')>=2, required('VARIABLES',var), 'VARIABLES'\var>=0, 'VARIABLES'\var=<1]).

ctr_example(
    nor,
    [nor(1, [[var-0], [var-0]]),
     nor(0, [[var-0], [var-1]]),
     nor(0, [[var-1], [var-0]]),
     nor(0, [[var-1], [var-1]]),
     nor(0, [[var-1], [var-0], [var-1]])].

ctr_exchangeable(nor, [items('VARIABLES', all)]).

ctr_eval(nor, [automaton(nor_a)]).

ctr_pure_functional_dependency(nor, []).

ctr_functional_dependency(nor, 1, [2]).

ctr_contractible(nor, ['VAR'=1,'VARIABLES', any]).

ctr_extensible(nor, ['VAR'=0,'VARIABLES', any]).

ctr_aggregate(nor, [], ['#\union']).

nor_a(FLAG, VAR, VARIABLES) :-
check_type(dvar(0,1),VAR),
collection(VARIABLES,[dvar(0,1)]),
length(VARIABLES,L),
L>1,
get_attr1(VARIABLES,LIST),
append([VAR],LIST,LIST_VARIABLES),
AUTOMATON=
automaton(
    LIST_VARIABLES,
    _20931,
    LIST_VARIABLES,
    [source(s),sink(i),sink(k)],
    [arc(s,0,j),
     arc(s,1,i),
     arc(i,0,i),
     arc(j,0,j),
     arc(j,1,k),
     arc(k,0,k),
     arc(k,1,k)],
    [],
    [],
    []),
automaton_bool(FLAG,[0,1],AUTOMATON).
B.248  not_all_equal

◊ META-DATA:

ctr_date(not_all_equal, 

ctr_origin(not_all_equal, ’\index{CHIP|indexuse}CHIP’, []).

ctr_arguments(not_all_equal, [‘VARIABLES’–collection(var–dvar)]).

ctr_restrictions(
    not_all_equal,
    [required(‘VARIABLES’, var), size(‘VARIABLES’) > 1]).

ctr_example(
    not_all_equal,
    not_all_equal([[var-3],[var-1],[var-3],[var-3],[var-3]]).)

ctr_typical(not_all_equal, [size(‘VARIABLES’) > 2]).

ctr_exchangeable(
    not_all_equal,
    [items(‘VARIABLES’, all),
     vals([‘VARIABLES’^var], int, =\=, all, dontcare)]).

ctr_graph(
    not_all_equal,
    [‘VARIABLES’],
    2,
    [‘CLIQUE’>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    [‘NSCC’>1],
    []).

ctr_eval(
    not_all_equal,
    [checker(not_all_equal_c),
     reformulation(not_all_equal_r),
     automaton(not_all_equal_a)]).

ctr_extensible(not_all_equal, [], ‘VARIABLES’, any).

not_all_equal_c(VARIABLES) :-
    collection(VARIABLES, [int]),
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

```
length(VARIABLES,N),
N>1,
get_attr1(VARIABLES,VARS),
sort(VARS,S),
S=[_35970,_35972|_35973].

not_all_equal_r(VARIABLES) :-
collection(VARIABLES,[dvar]),
length(VARIABLES,N),
N>1,
get_attr1(VARIABLES,VARS),
NVAL in 2..N,
nvalue(NVAL,VARS).

not_all_equal_a(FLAG,VARIABLES) :-
collection(VARIABLES,[dvar]),
length(VARIABLES,N),
N>1,
not_all_equal_signature(VARIABLES,SIGNATURE),
AUTOMATON=automaton(
  SIGNATURE,
  _37475,
  SIGNATURE,
  [source(s),sink(t)],
  [arc(s,1,s),arc(s,0,t),arc(t,0,t),arc(t,1,t)],
  [],
  [],
  []),
automaton_bool(FLAG,[0,1],AUTOMATON).

not_all_equal_signature([],[]).
not_all_equal_signature([_35928],[]) :- !.
not_all_equal_signature([[var-VAR1],[var-VAR2]|VARS],[S|Ss]) :-
  VAR1#=VAR2#<==>S,
  not_all_equal_signature([[var-VAR2]|VARS],Ss).
```
B.249  not_in

◊ **META-DATA:**

```
ctr_date(not_in,['20030820','20040530','20060812']).

ctr_origin(not_in,'Derived from %c.',[in]).

ctr_arguments(not_in,['VAR'-dvar,'VALUES'-collection(val-int)]).

ctr_restrictions(
    not_in,
    [required('VALUES',val),distinct('VALUES',val)]).

ctr_example(not_in,not_in(2,[[val-1],[val-3]])).

ctr_typical(not_in,[size('VALUES')>1]).

ctr_exchangeable(
    not_in,
    [items('VALUES',all),translate(['VAR','VALUES'\$val])].

ctr_derived_collections(
    not_in,
    [col('VARIABLES'-collection(var-dvar),[item(var-'VAR')])].

ctr_graph(
    not_in,
    ['VARIABLES','VALUES'],
    2,
    ['PRODUCT'\collection(variables,values)],
    [variables\$var=values\$val],
    ['NARC'=0],
    []).

ctr_eval(not_in,[automaton(not_in_a)]).

ctr_contractible(not_in,[],'VALUES',any).
```

```
not_in_a(FLAG,VAR,VALUES) :-
    check_type(dvar,VAR),
    collection(VALUES,[int]),
    get_attr1(VALUES,VALS),
    all_different(VALS),
    not_in_signature(VALUES,SIGNALATURE,VAR),
    AUTOMATON=
```
automaton(  
    SIGNATURE, 
    _29845,  
    SIGNATURE,  
    [source(s), sink(s)],  
    [arc(s, 0, s)],  
    [],  
    [],  
    []),  
automaton_bool(FLAG, [0, 1], AUTOMATON).

not_in_signature([], [], _28219).

not_in_signature([[val-VAL]|VALs],[S|Ss],VAR) :-  
    VAR#=VAL#=S,  
    not_in_signature(VALs, Ss, VAR).


B.250 npair

◊ **META-DATA:**

ctr_date(npair, ['20030820', '20060812']).

ctr_origin(npair, 'Derived from %c.', [nvalue]).

ctr_arguments(
    npair,
    ['NPAIRS'-dvar, 'PAIRS'-collection(x-dvar, y-dvar)]).

ctr_restrictions(
    npair,
    ['NPAIRS'=\min(1, \text{size('PAIRS')})],
    'NPAIRS'=<\text{size('PAIRS')},
    \text{required('PAIRS', [x, y])}].

ctr_example(
    npair,
    npair(2,
        [[x-3, y-1], [x-1, y-5], [x-3, y-1], [x-3, y-1], [x-1, y-5]]).

ctr_typical(
    npair,
    ['NPAIRS'>1,
    'NPAIRS'\text{size('PAIRS')},
    \text{size('PAIRS')}>1,
    \text{range('PAIRS' }^x) > 1,
    \text{range('PAIRS' } ^y) > 1]).

ctr_exchangeable(
    npair,
    [\text{items('PAIRS', all)},
    \text{attrs_sync('PAIRS', [[x, y]]),
    \text{vals(['}NPAIRS'], \text{int, =\text{\}, all, dontcare}])}.

ctr_graph(
    npair,
    ['PAIRS'],
    2,
    ['\text{CLIQUE}'] >> \text{collection(pairs1, pairs2)},
    [pairs1'x=pairs2'x, pairs1'y=pairs2'y],
    ['\text{NSCC}'='NPAIRS'],
    []).
ctr_pure_functional_dependency(npair, []).  

ctr_functional_dependency(npair, 1, [2]).  

ctr_contractible(npair, ['NPAIRS'=1, size('PAIRS')>0], PAIRS, any).  

ctr_contractible(npair, ['NPAIRS'=size('PAIRS')], 'PAIRS', any).
B.251  nset_of_consecutive_values

◇ Meta-Data:

ctr_date(
    nset_of_consecutive_values,
    ['20030820','20040530','20060812']).

ctr_origin(nset_of_consecutive_values,'N.˜Beldiceanu',[]).

ctr_arguments(
    nset_of_consecutive_values,
    ['N'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    nset_of_consecutive_values,
    ['N'>=1,'N'=<size('VARIABLES'),required('VARIABLES',var)]).

ctr_example(
    nset_of_consecutive_values,
    nset_of_consecutive_values(2,
        [[var-3],
        [var-1],
        [var-7],
        [var-1],
        [var-1],
        [var-2],
        [var-8]])).

ctr_typical(
    nset_of_consecutive_values,
    ['N'>1,size('VARIABLES')>1,range('VARIABLES'`var)>1]).

ctr_exchangeable(
    nset_of_consecutive_values,
    [items('VARIABLES',all),
    vals(['VARIABLES`var',int,=\=,all,in],
    translate(['VARIABLES`var']))].

ctr_graph(
    nset_of_consecutive_values,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [abs(variables1`var-variables2`var)=<1],
    ...
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

[‘NSCC’=‘N’],
[]).

ctr_pure_functional_dependency(nset_of_consecutive_values,[]).

ctr_functional_dependency(nset_of_consecutive_values,1,[2]).
B.252  nvalue

◊ **META-DATA:**

```prolog
ctr_date(
  nvalue,
  [20000128,
   20030820,
   20040530,
   20051001,
   20060812,
   20091105]).
```

```prolog
ctr_origin(nvalue,'\cite{PachetRoy99}',[]).
```

```prolog
ctr_synonyms(nvalue,[cardinality_on_attributes_values,values]).
```

```prolog
ctr_arguments(
  nvalue,
  ['NVAL'-dvar,'VARIABLES'-collection(var-dvar)]).
```

```prolog
ctr_restrictions(
  nvalue,
  [required('VARIABLES',var),
   'NVAL'>=min(1,size('VARIABLES')),
   'NVAL'=<size('VARIABLES'),
   'NVAL'=<range('VARIABLES'\^var)]).
```

```prolog
ctr_example(
  nvalue,
  nvalue(4,[[var-3],[var-1],[var-7],[var-1],[var-6]])).
```

```prolog
ctr_typical(
  nvalue,
  ['NVAL'>1,
   'NVAL'<size('VARIABLES'),
   'NVAL'<range('VARIABLES'\^var),
   size('VARIABLES')>1,
   'NVAL'<0#'/NVAL'>1])).
```

```prolog
ctr_exchangeable(
  nvalue,
  [items('VARIABLES',all),
   vals(['VARIABLES'\^var],int,=\=,all,dontcare)]).
```

ctr_graph(}
nvalue,
   ['VARIABLES'],
2,
['CLIQUE'>>collection(variables1,variables2)],
[variables1^var=variables2^var],
['NSCC'='NVAL'],
['EQUIVALENCE']).

ctr_eval(nvalue,[builtin(nvalue_b)]).

ctr_pure_functional_dependency(nvalue,[]).

ctr_functional_dependency(nvalue,1,[2]).

ctr_contractible(
   nvalue,
   ['NVAL'=1,size('VARIABLES')>0],
   VARIABLES,
   any).

ctr_contractible(
   nvalue,
   ['NVAL'=size('VARIABLES')],
   VARIABLES,
   any).

nvalue_b(NVAL,VARIABLES) :-
   check_type(dvar,NVAL),
   collection(VARIABLES,[dvar]),
   get_attr1(VARIABLES,VARS),
   length(VARIABLES,N),
   NVAL#>=min(1,N),
   NVAL#=<N,
   list_dvar_range(VARS,R),
   NVAL#=<R,
   nvalue(NVAL,VARS).
B.253  nvalue_on_intersection

◊ **META-DATA:**

ctr_date(nvalue_on_intersection,['20040530','20060812']).

ctr_origin(
    nvalue_on_intersection,
    Derived from %c and %c.,
    [common,nvalue]).

ctr_arguments(
    nvalue_on_intersection,
    ['NVAL'-dvar,
    'VARIABLES1'-collection(var-dvar),
    'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    nvalue_on_intersection,
    [required('VARIABLES1',var),
    required('VARIABLES2',var),
    'NVAL'>=0,
    'NVAL'=<size('VARIABLES1'),
    'NVAL'=<size('VARIABLES2'),
    'NVAL'=<range('VARIABLES1'\var),
    'NVAL'=<range('VARIABLES2'\var)]).

ctr_example(
    nvalue_on_intersection,
    nvalue_on_intersection(2,
                        [[\var-1],[\var-9],[\var-1],[\var-5]],
                         [[\var-2],[\var-1],[\var-9],[\var-9],[\var-6],[\var-9]])).

ctr_typical(
    nvalue_on_intersection,
    ['NVAL']\>0,
    'NVAL'<size('VARIABLES1'),
    'NVAL'<size('VARIABLES2'),
    'NVAL'<range('VARIABLES1'\var),
    'NVAL'<range('VARIABLES2'\var),
    size('VARIABLES1')\>1,
    size('VARIABLES2')\>1]).

ctr_exchangeable(
    nvalue_on_intersection,
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

[\texttt{args([['NVAL'],['\textit{VARIABLES1}','\textit{VARIABLES2}']])},
\texttt{items('VARIABLES1',all),
items('VARIABLES2',all),
\texttt{vals(
    ['\textit{VARIABLES1}^\texttt{var}','\textit{VARIABLES2}^\texttt{var}],
    int,
    =\texttt{\not=},
    all,
    dontcare))}].

\texttt{ctr_graph(}
\texttt{nvalue_on_intersection,}
\texttt{['\textit{VARIABLES1}','\textit{VARIABLES2}'],
2,
['\texttt{PRODUCT}'>collection(variables1,variables2)],
[variables1^\texttt{var}=variables2^\texttt{var}],
['\texttt{NCC}']='\texttt{NVAL}',
[]).

\texttt{ctr_pure_functional_dependency(nvalue_on_intersection,[]).}

\texttt{ctr_functional_dependency(nvalue_on_intersection,1,[2,3]).}

\texttt{ctr_contractible(}
\texttt{nvalue_on_intersection,}
\texttt{['\texttt{NVAL}']=0],
\texttt{VARIABLES1,}
\texttt{any).}

\texttt{ctr_contractible(}
\texttt{nvalue_on_intersection,}
\texttt{['\texttt{NVAL}']=0],
\texttt{VARIABLES2,}
\texttt{any).}
B.254  nvalues

◊ **META-DATA:**

```prolog
ctr_date(nvalues,[’20030820’,’20060812’]).

ctr_origin(nvalues,’Inspired by %c and %c.’,[nvalue,count]).

ctr_arguments(nvalues,
   [’VARIABLES’-collection(var-dvar),
    ’RELOP’-atom,
    ’LIMIT’-dvar]).

ctr_restrictions(nvalues,
   [required(’VARIABLES’,var),
    in_list(’RELOP’,[=,\=,<,>,>,=<])].

ctr_example(nvalues,
   nvalues([[var-4],[var-5],[var-5],[var-4],[var-1],[var-5]],
      =,3)).

ctr_typical(nvalues,
   [size(’VARIABLES’)>1,
    ’LIMIT’>1,
    ’LIMIT’<size(’VARIABLES’),
    in_list(’RELOP’,[=,\=,<,>,>,=<])].

ctr_exchangeable(nvalues,
   [items(’VARIABLES’,all),
    vals([’VARIABLES’^var],int,\=,all,dontcare)]).

ctr_graph(nvalues,
   [’VARIABLES’],
   2,
   [’CLIQUE’>>collection(variables1:variables2)],
   [variables1^var=variables2^var],
   [’RELOP’(’NSCC’,’LIMIT’)],
   [’EQUIVALENCE’]).
```
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

\[ \text{ctr_eval}(\text{nvalues}, [\text{reformulation}(\text{nvalues}_r)])]. \]

\[ \text{ctr_pure_functional_dependency}(\text{nvalues}, [\text{in_list}('\text{RELOP}', [=]])]. \]

\[ \text{ctr_contractible}(\text{nvalues}, [\text{in_list}('\text{RELOP}', [<,=])], \text{VARIABLES}, \text{any}). \]

\[ \text{ctr_contractible}(\text{nvalues}, [\text{in_list}('\text{RELOP}', [=]), '\text{LIMIT}'=1, \text{size('VARIABLES')}>0], \text{VARIABLES}, \text{any}). \]

\[ \text{ctr_contractible}(\text{nvalues}, [\text{in_list}('\text{RELOP}', [=]), '\text{LIMIT}'=\text{size('VARIABLES')}], \text{VARIABLES}, \text{any}). \]

\[ \text{ctr_extensible}(\text{nvalues}, [\text{in_list}('\text{RELOP}', [\geq,\leq])], \text{VARIABLES}, \text{any}). \]

\[ \text{nvalues}_r(\text{VARIABLES}, \text{RELOP}, \text{LIMIT}) :- \]
\[ \text{collection}('\text{VARIABLES}', [\text{dvar}]}, \]
\[ \text{memberchk}(\text{RELOP}, [=, \leq, \geq, =, =\leq, =\geq])}, \]
\[ \text{check_type}('\text{dvar}, \text{LIMIT})}, \]
\[ \text{length}('\text{VARIABLES'}, \text{N)}, \]
\[ \text{NVAL in 0..N)}, \]
\[ \text{get_attr1}('\text{VARIABLES'}, \text{VARS})}, \]
\[ \text{nvalue}(\text{NVAL}, \text{VARS})}, \]
\[ \text{call_term_relop_value}(\text{NVAL}, \text{RELOP}, \text{LIMIT}). \]
B.255  nvalues_except_0

◊ META-DATA:

ctr_date(nvalues_except_0, ['20030820', '20060812']).

ctr_origin(nvalues_except_0, 'Derived from %c.', [nvalues]).

ctr_arguments(nvalues_except_0, nvalues_except_0, nvalues_except_0,

   ['VARIABLES' - collection(var-dvar),
     'RELOP' - atom,
     'LIMIT' - dvar]).

ncrt_restrictions(nvalues_except_0, [required('VARIABLES', var),
     in_list('RELOP', [\(=\), \(<\), \(\geq\), \(\leq\), \(\neq\)]))].

ncrt_example(nvalues_except_0, nvalues_except_0([var-4], [var-5], [var-5], [var-4], [var-0], [var-1],

   [\(\leq\),
     3])).

ncrt_typical(nvalues_except_0, [size('VARIABLES') > 1,

   'LIMIT' > 1,
   'LIMIT' < size('VARIABLES'),
   atleast(1, 'VARIABLES', 0),
   in_list('RELOP', [\(=\), \(<\), \(\geq\), \(\leq\), \(\neq\)]))].

ncrt_exchangeable(nvalues_except_0, [items('VARIABLES', all),

   vals(['VARIABLES' \(^\text{\textasciitilde}\text{\textvar}]), int(\(=\) (0)), \(\leq\), all, dontcare]]).

ncrt_graph(nvalues_except_0, ['VARIABLES'],

   2,
   ['CLIQUE' \(\triangleright\) collection(variables1, variables2)],

   [variables1 \(^\text{\textvar} =\) 0, variables1 \(^\text{\textvar} =\) variables2 \(^\text{\textvar} ),

   ['RELOP' ('NSCC', 'LIMIT')]].
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

[]).

ctr_eval(nvalues_except_0,[reformulation(nvalues_except_0_r)]).

ctr_contractible(
    nvalues_except_0,
    [in_list('RELOP',[<,=<])],
    VARIABLES,
    any).

ctr_extensible(
    nvalues_except_0,
    [in_list('RELOP',[>=,>])],
    VARIABLES,
    any).

nvalues_except_0_r(VARIABLES,RELOP,LIMIT) :-
    collection(VARIABLES,[dvar]),
    memberchk(RELOP,[=,\=,<,\>=,>,\>=,\le,\le<]),
    check_type(dvar,LIMIT),
    length(VARIABLES,N),
    N1 is N+1,
    NVAL1 in 1..N1,
    get_attr1(VARIABLES,VARS),
    append([0],VARS,VARS0),
    nvalue(NVAL1,VARS0),
    NVAL1#=NVAL+1,
    call_term_relop_value(NVAL,RELOP,LIMIT).
B.256 nvector

◊ **META-DATA:**

`ctr_date(nvector, ['20081220']).`

`ctr_origin(nvector, `
```chabert as a generalisation of %c, [nvalue]).```

`ctr_synonyms(nvector, [nvectors, npoint, npoints]).`

`ctr_types(nvector, ['VECTOR'-collection(var-dvar)]).`

`ctr_arguments(nvector, ['NVEC'-dvar, 'VECTORS'-collection(vec-'VECTOR')]).`

`ctr_restrictions(nvector, `
```size('VECTOR')>=1, 'NVEC'>=min(1,size('VECTORS')), 'NVEC'=<size('VECTORS'), required('VECTORS', vec), same_size('VECTORS', vec)).```

`ctr_example(nvector, nvector(2, `
```[[vec-[[var-5],[var-6]]], [vec-[[var-5],[var-6]]], [vec-[[var-9],[var-3]]], [vec-[[var-5],[var-6]]], [vec-[[var-9],[var-3]]])).````

`ctr_typical(nvector, `
```size('VECTOR')>1, 'NVEC'>1, 'NVEC'<size('VECTORS'), size('VECTORS')>1]).```

`ctr_exchangeable(nvector, ```
[items('VECTORS',all),
  items_sync('VECTORS'\^vec,all),
  vals(['VECTORS'\^vec],int,=\-,all,dontcare))].

ctr_graph(
  nvector,
  ['VECTORS'],
  2,
  ['CLIQUE'>>collection(vectors1,vectors2)],
  [lex_equal(vectors1\^vec,vectors2\^vec)],
  ['NSCC'='NVEC'],
  ['EQUIVALENCE']).

ctr_eval(nvector,[reformulation(nvector_r)]).

ctr_pure_functional_dependency(nvector,[]).

ctr_functional_dependency(nvector,1,[2]).

ctr_contractible(
  nvector,
  ['NVEC'=1,size('VECTORS')>0],
  VECTORS,
  any).

ctr_contractible(
  nvector,
  ['NVEC'=size('VECTORS')],
  VECTORS,
  any).

nvector_r(0,[]) :-
  !.

nvector_r(NVEC,VECTORS) :-
  check_type(dvar,NVEC),
  collection(VECTORS,[col([dvar])]),
  same_size(VECTORS),
  length(VECTORS,N),
  NVEC#>=min(1,N),
  NVEC#=<N,
  nvector_common(NVEC,VECTORS).
B.257  nvectors

◊ **META-DATA:**

```erlang
ctr_date(nvectors,['20081226']).

ctr_origin(nvectors,'Inspired by %c and %c.',[nvectors,count]).

ctr_synonyms(nvectors,[npoints]).

ctr_types(nvectors,['VECTOR'-collection(var-dvar)]).

ctr_arguments(nvectors,
               ['VECTORS'-collection(vec-'VECTOR'),
                'RELOP'-atom,
                'LIMIT'-dvar]).

ctr_restrictions(nvectors,
                 [size('VECTOR')>=1,
                  required('VECTORS',vec),
                  same_size('VECTORS',vec),
                  in_list('RELOP',[=,\=,<,>=,>,=<])]).

ctr_example(nvectors,
            nvectors(
                       [[vec-[[var-5],[var-6]]],
                        [vec-[[var-5],[var-6]]],
                        [vec-[[var-9],[var-3]]],
                        [vec-[[var-5],[var-6]]],
                        [vec-[[var-9],[var-3]]],
                        =, 2)).

ctr_typical(nvectors,
            [size('VECTOR')>1,
             size('VECTORS')>1,
             in_list('RELOP',[=,\<,\>=,\>,\=<]),
             'LIMIT'>1,
             'LIMIT'<size('VECTORS'))].

ctr_exchangeable(nvectors,
                 )
```
[items('VECTORS',all),
  items_sync('VECTORS'\vec,all),
  vals(['VECTORS'\vec],int,=\=-,all,dontcare))].

ctr_graph(
  n_vectors,
  ['VECTORS'],
  2,
  ['CLIQUE'>>collection(vectors1,vectors2)],
  [lex_equal(vectors1\vec,vectors2\vec)],
  [\textsc{relop}'('NSCC','LIMIT')],
  ['EQUIVALENCE']).

ctr_eval(n_vectors,[\text{reformulation}(n_vectors_r)]).

ctr_pure_functional_dependency(n_vectors,[in_list('RELOP',[=])]).

ctr_contractible(
  n_vectors,
  [in_list('RELOP',[<,\leq])],
  VECTORS,
  any).

ctr_extensible(
  n_vectors,
  [in_list('RELOP',[\geq,>])],
  VECTORS,
  any).

n_vectors_r(VECTORS,RELOP,LIMIT) :-
  memberchk(RELOP,[=\leq,<,\leq>,>,\leq]),
  check_type(dvar,LIMIT),
  length(VECTORS,N),
  NV in 0..N,
  eval(nvector(NV,VECTORS)),
  call_term_relop_value(NV,RELOP,LIMIT).
B.258 nvisible_from_end

◊ META-DATA:

ctr_date(nvisible_from_end,['20111228']).

ctr_origin(
    nvisible_from_end,
    Derived from %c,
    [nvisible_from_start]).

ctr_synonyms(nvisible_from_end,[nvisible,nvisible_from_right]).

ctr_arguments(
    nvisible_from_end,
    ['N'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    nvisible_from_end,
    [required('VARIABLES',var),
     'N'>=min(1,size('VARIABLES')),
     'N'=<size('VARIABLES')]).

ctr_example(
    nvisible_from_end,
    nvisible_from_end(2,
        [[var-1],
         [var-6],
         [var-2],
         [var-1],
         [var-4],
         [var-8],
         [var-2])).

ctr_typical(nvisible_from_end,[size('VARIABLES')>2]).

ctr_exchangeable(
    nvisible_from_end,
    [translate(['VARIABLES'~var])]).

ctr_eval(nvisible_from_end,[automaton(nvisible_from_end_a)]).

ctr_pure_functional_dependency(nvisible_from_end,[]).

ctr_functional_dependency(nvisible_from_end,1,[2]).
nvisible_from_end_a(FLAG,N,VARIABLES) :-
collection(VARIABLES,[dvar]),
length(VARIABLES,L),
MIN is min(1,L),
check_type(dvar(MIN,L),N),
get_attr1(VARIABLES,VARS),
reverse(VARS,RVARS),
(foreach(_13828,VARS),foreach(0,SIGNATURE)do true),
automaton(
  RVARS,
  Vi,
  SIGNATURE,
  [source(s),sink(s),sink(t)],
  [arc(s,0,t,[Vi,1]),
   arc(t,0,t,(M#<Vi->[Vi,C+1];M#>=Vi->[M,C]))],
  [M,C],
  [0,0],
  [_13917,COUNT]),
COUNT#=#<=>FLAG.
B.259  nvisible_from_start

◊ **META-DATA:**

`ctr_date(nvisible_from_start, ['20111227'])`.

`ctr_origin(nvisible_from_start, Derived from a puzzle called skyscraper, [])`.

`ctr_synonyms(nvisible_from_start, [nvisible, nvisible_from_left])`.

`ctr_arguments(nvisible_from_start, ['N'-dvar, 'VARIABLES'-collection(var-dvar)])`.

`ctr_restrictions(nvisible_from_start, [required('VARIABLES', var), 'N' ≥ min(1, size('VARIABLES')), 'N' =< size('VARIABLES')])`.

`ctr_example(nvisible_from_start, nvisible_from_start(3, [[var-1], [var-6], [var-2], [var-1], [var-4], [var-8], [var-2])))`.

`ctr_typical(nvisible_from_start, [size('VARIABLES') > 2])`.

`ctr_exchangeable(nvisible_from_start, [translate(['VARIABLES'~var])])`.

`ctr_eval(nvisible_from_start, [automaton(nvisible_from_start_a)])`.

`ctr_pure_functional_dependency(nvisible_from_start, [])`.
ctr_functional_dependency(nvisible_from_start,1,[2]).

nvisible_from_start_a(FLAG,N,VARIABLES) :-
    collection(VARIABLES,[dvar]),
    length(VARIABLES,L),
    MIN is min(1,L),
    check_type(dvar(MIN,L),N),
    get_attr1(VARIABLES,VARS),
    (foreach(_13684,VARS),foreach(0,SIGNATURE)do true),
    automaton(
        VARS,
        Vi,
        SIGNATURE,
        [source(s),sink(s),sink(t)],
        [arc(s,0,t,[Vi,1]),
         arc(t,0,t,(M#<Vi->[Vi,C+1];M#>=Vi->[M,C]))],
        [M,C],
        [0,0],
        [_13773,COUNT]),
    COUNT#=N#<=>FLAG.
**B.260 open_alldifferent**

◊ **Meta-Data:**

ctr_date(open_alldifferent,[‘20060824’,’20090524’]).

ctr_origin(open_alldifferent,’\cite{HoeveRegin06}’,[]).

ctr_synonyms(
    open_alldifferent,
    [open_alldiff,open_alldistinct,open_distinct]).

ctr_arguments(
    open_alldifferent,
    [‘S’-svar,’VARIABLES’-collection(var-dvar)]).

ctr_restrictions(
    open_alldifferent,
    [‘S’>=1,’S’=<size(‘VARIABLES’),required(‘VARIABLES’,var)]).

ctr_example(
    open_alldifferent,
    open_alldifferent(
        {2,3,4},
        [[var-9],[var-1],[var-9],[var-3]]).

ctr_typical(open_alldifferent,[size(‘VARIABLES’)>2]).

ctr_exchangeable(
    open_alldifferent,
    [vals([‘VARIABLES’ˆvar],int,\=,all,dontcare)]).

ctr_graph(
    open_alldifferent,
    [‘VARIABLES’],
    2,
    [‘CLIQUE’>collection(variables1,variables2)],
    [variables1ˆvar=variables2ˆvar,
     variables1ˆkey in set ’S’,
     variables2ˆkey in set ’S’],
    [‘MAX_NSCC’=<1],
    [‘ONE_SUCC’]).

ctr_contractible(open_alldifferent,[],’VARIABLES’,suffix).
B.261  open_among

♦ Meta-Data:

ctr_date(open_among, ['20060824']).

ctr_origin(
    open_among,
    Derived from %c and %c.,
    [among, open_global_cardinality]).

ctr_arguments(
    open_among,
    ['S'-svar,
     'NVAR'-dvar,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int)]).

ctr_restrictions(
    open_among,
    ['S'>=1,
     'S'=<size('VARIABLES'),
     'NVAR'>=0,
     'NVAR'=<size('VARIABLES'),
     required('VARIABLES',var),
     required('VALUES',val),
     distinct('VALUES', val)]).

ctr_example(
    open_among,
    open_among(
        {2,3,4,5},
        3,
        [[var-8],[var-5],[var-5],[var-4],[var-1]],
        [[val-1],[val-5],[val-8]]).

ctr_typical(
    open_among,
    ['NVAR'>0,
     'NVAR'<size('VARIABLES'),
     size('VARIABLES')>1,
     size('VALUES')>1,
     size('VARIABLES')>size('VALUES')]).

ctr_exchangeable(
    open_among,
[items('VALUES',all),
vals(
    ['VARIABLES'\textasciitilde{}var],
    \texttt{comp('VALUES'\textasciitilde{}val),
    =,}
    dontcare,
    dontcare))].

\texttt{ctr\_graph(}
    open\_among,
    ['VARIABLES'],
    1,
    ['SELF'\textasciitilde{}collection(variables)],
    [variables\textasciitilde{}var in 'VALUES',variables\textasciitilde{}key in\_set 'S'],
    ['NARC'='NVAR'],
    []).

\texttt{ctr\_functional\_dependency(open\_among,2,[1,3,4]).}

\texttt{ctr\_contractible(open\_among,['NVAR']=0,['VARIABLES',suffix]).}
B.262  open_atleast

◊ **Meta-Data:**

\[
\text{ctr\_date(open\_atleast,}[\text{[20060824]})].
\]

\[
\text{ctr\_origin(}
\begin{align*}
\text{open\_atleast,} \\
\text{Derived from %c and %c.,} \\
\text{[atleast,open\_global\_cardinality].}
\end{align*}
\]

\[
\text{ctr\_arguments(}
\begin{align*}
\text{open\_atleast,} \\
\text{[}S\text{-svar,} \\
\text{N-int,} \\
\text{VARIABLES\text{-collection(var-dvar),} } \\
\text{VALUE\text{-int}]}. \\
\end{align*}
\]

\[
\text{ctr\_restrictions(}
\begin{align*}
\text{open\_atleast,} \\
\text{[}S\text{>=1,} \\
\text{S=<size(VARIABLES),} \\
\text{N>=0,} \\
\text{N=<size(VARIABLES),} \\
\text{required(VARIABLES,} \text{var}])}. \\
\end{align*}
\]

\[
\text{ctr\_example(}
\begin{align*}
\text{open\_atleast,} \\
\text{open\_atleast(} \\
\text{[2,3,4],} \\
2, \\
\text{[[var-4],[var-2],[var-4],[var-4]],} \\
4\text{)}.} \\
\end{align*}
\]

\[
\text{ctr\_typical(}
\begin{align*}
\text{open\_atleast,} \\
\text{[}N\text{>0,'}N'\text{<size(VARIABLES),size(VARIABLES)>1]}. \\
\end{align*}
\]

\[
\text{ctr\_exchangeable(}
\begin{align*}
\text{open\_atleast,} \\
\text{vals([}'N'\text{],int(>=(0))>,} \text{dontcare,dontcare),} \\
\text{vals(} \\
\text{[}'VARIABLES'\text{^}var], \\
\text{comp('VALUE'),} \\
\text{>=,} \\
\text{dontcare,} \\
\end{align*}
\]


dontcare).}

ctr_graph{
    open_atleast,
    ['VARIABLES'],
    1,
    ['SELF']>>collection(variables],
    [variables^var='VALUE',variables^key in_set 'S'],
    ['NARC'>'N'],
    []).

ctr_extensible(open_atleast,[],'VARIABLES',suffix).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.263 open_atmost

◊ Meta-Data:

ctr_date(open_atmost, ['20060824']).

ctr_origin(
    open_atmost,
    Derived from %c and %c.,
    [atmost, open_global_cardinality]).

ctr_arguments(
    open_atmost,
    ['S'-svar, 
     'N'-int, 
     'VARIABLES'-collection(var-dvar),
     'VALUE'-int]).

ctr_restrictions(
    open_atmost,
    ['S'>=1,
     'S'=<size('VARIABLES'),
     'N'>=0,
     required('VARIABLES',var))).

ctr_example(
    open_atmost,
    open_atmost({2,3,4},1,[[var-2],[var-2],[var-4],[var-5]],2)).

ctr_typical(
    open_atmost,
    ['N']>0,'N'<size('VARIABLES'),size('VARIABLES')>1]).

ctr_exchangeable(
    open_atmost,
    [vals(['N'],int,<,dontcare,dontcare),
     vals(
      ['VARIABLES'\^var],
      comp('VALUE'),
      =<,
      dontcare, 
      dontcare)]).

ctr_graph(
    open_atmost,
    ['VARIABLES'],
    ...)
1,
['SELF'>>collection(variables)],
[variables^var='VALUE',variables^key in_set 'S'],
['NARC'='N'],
[]).

ctr_contractible(open_atmost,[],'VARIABLES',suffix).
B.264 open_global_cardinality

◊ Meta-Data:

ctr_date(open_global_cardinality,['20060824']).

ctr_origin(open_global_cardinality,\cite{HoeveReg in06},[]).

ctr_synonyms(open_global_cardinality,[open_gcc,ogcc]).

ctr_arguments(
  open_global_cardinality,
  ['S'-svar,
   'VARIABLES'-collection(var-dvar),
   'VALUES'-collection(val-int,noccurrence-dvar)]).

ctr_restrictions(
  open_global_cardinality,
  ['S'>=1,
   'S'=<size('VARIABLES'),
   required('VARIABLES',var),
   required('VALUES',[val,noccurrence]),
   distinct('VALUES',val),
   'VALUES'\noccurrence>=0,
   'VALUES'\noccurrence=<size('VARIABLES'))).

ctr_example(
  open_global_cardinality,
  open_global_cardinality(2,3,4),
  [[var-3],[var-3],[var-8],[var-6]],
  [[val-3,noccurrence-1],
   [val-5,noccurrence-0],
   [val-6,noccurrence-1]])).

ctr_typical(
  open_global_cardinality,
  [size('VARIABLES')>1,
   range('VARIABLES'\var)>1,
   size('VALUES')>1,
   range('VALUES'\noccurrence)>1,
   size('VARIABLES')>size('VALUES'))).

ctr_exchangeable(
  open_global_cardinality,
  [items('VALUES',all),
   ...]}).
vals(
    [‘VARIABLES’\^\text{\textasciitilde}var],
    all(notin(‘VALUES’\^\text{\textasciitilde}val)),
    =,
    dontcare,
    dontcare))).

ctr_graph(
    open_global_cardinality,
    [‘VARIABLES’],
    1,
    foreach(‘VALUES’,[‘SELF’\text{\textasciitilde}collection(variables)]),
    [variables\text{\textasciitilde}var=‘VALUES’\text{\textasciitilde}val,variables\text{\textasciitilde}key in_set ‘S’],
    [‘NVERTEX’=‘VALUES’\text{\textasciitilde}noccurrence],
    []).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.265 open_global_cardinality_low_up

◊ Meta-Data:

ctr_date(open_global_cardinality_low_up,['20060824']).

ctr_origin(
    open_global_cardinality_low_up,
    \cite{HoeveRegin06},
    []).

ctr_arguments(
    open_global_cardinality_low_up,
    ['S'-svar,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int,omin-int,omax-int)]).

ctr_restrictions(
    open_global_cardinality_low_up,
    ['S'>=1,
     'S'=<size('VARIABLES'),
     required('VARIABLES',var),
     size('VALUES')>0,
     required('VALUES',[val,omin,omax]),
     distinct('VALUES',val),
     'VALUES'ˆomin>=0,
     'VALUES'ˆomax=<size('VARIABLES'),
     'VALUES'ˆomin=<'VALUES'ˆomax]).

ctr_example(
    open_global_cardinality_low_up,
    open_global_cardinality_low_up(
        {2,3,4},
        [[var-3],[var-3],[var-8],[var-6]],
        [[val-3,omin-1,omax-3],
         [val-5,omin-0,omax-1],
         [val-6,omin-1,omax-2]]).

ctr_typical(
    open_global_cardinality_low_up,
    [size('VARIABLES')>1,
     range('VARIABLES'ˆvar)>1,
     size('VALUES')>1,
     'VALUES'ˆomin=<size('VARIABLES'),
     'VALUES'ˆomax>0,
     'VALUES'ˆomax=<size('VARIABLES'),
     ...)
size('VARIABLES') > size('VALUES'))).

ctr_exchangeable(
    open_global_cardinality_low_up,
    [items('VALUES', all),
     vals(
         ['VARIABLES'~var],
         all(notin('VALUES'~val)),
         =,
         dontcare,
         dontcare))].

ctr_graph(
    open_global_cardinality_low_up,
    ['VARIABLES'],
    1,
    foreach('VALUES', ['SELF'>collection(variables))],
    [variables~var='VALUES'~val, variables~key in_set 'S'],
    ['NVERTEX'>='VALUES'~omin, 'NVERTEX'='VALUES'~omax],
    []).
B.266 open\_maximum

\textbf{Meta-Data:}

c\_r\_date(open\_maximum,[‘20090507’]).

c\_r\_origin(open\_maximum,’Derived from %c’,[maximum]).

c\_r\_arguments(
    open\_maximum,
    [‘MAX’-dvar,’VARIABLES’-collection(var-dvar,bool-dvar)]).

c\_r\_restrictions(
    open\_maximum,
    [size(‘VARIABLES’)\textgreater{}0,
    required(‘VARIABLES’,[var,bool]),
    ‘VARIABLES’\textasciicircum{}bool\textgreater{}=0,
    ‘VARIABLES’\textasciicircum{}bool\textless{}=1]).

c\_r\_example(
    open\_maximum,
    open\_maximum(
        5,
        [[var-3,bool-1],
        [var-1,bool-0],
        [var-7,bool-0],
        [var-5,bool-1],
        [var-5,bool-1]])).

c\_r\_typical(
    open\_maximum,
    [size(‘VARIABLES’)\textgreater{}1,range(‘VARIABLES’\textasciicircum{}var)>1]).

c\_r\_exchangeable(
    open\_maximum,
    [items(‘VARIABLES’,all),
    translate([‘MAX’,‘VARIABLES’\textasciicircum{}var])]).

c\_r\_eval(open\_maximum,[automaton(open\_maximum\_a)]).

open\_maximum\_a(FLAG,MAX,VARIABLES) :-
    check\_type(dvar,MAX),
    collection(VARIABLES,[dvar,dvar(0,1)]),
    length(VARIABLES,N),
    N\textgreater{}0,
    open\_maximum\_signature(VARIABLES,SIGNATURE,MAX),

AUTOMATON=
automaton(
    SIGNATURE,
    _17774,
    SIGNATURE,
    [source(s), sink(t)],
    [arc(s,0,s),
     arc(s,1,t),
     arc(s,3,s),
     arc(s,4,s),
     arc(s,5,s),
     arc(t,1,t),
     arc(t,0,t),
     arc(t,3,t),
     arc(t,4,t),
     arc(t,5,t)],
    [],
    [],
    []),
automaton_bool(FLAG,[0,1,2,3,4,5],AUTOMATON).

open_maximum_signature([],[],_15873).

open_maximum_signature([|VAR| bool-B]|VARs],[S|Ss],MAX) :-
  S in 0..5,
  B#=1/\MAX#>VAR#<=>S#=0,
  B#=1/\MAX=#VAR#<=>S#=1,
  B#=1/\MAX#<VAR#<=>S#=2,
  B#=0/\MAX#>VAR#<=>S#=3,
  B#=0/\MAX=#VAR#<=>S#=4,
  B#=0/\MAX#<VAR#<=>S#=5,
  open_maximum_signature(VARs,Ss,MAX).
B.267 open_minimum

◊ **Meta-Data:**

ctr_date(open_minimum,['20090506']).

ctr_origin(open_minimum,'Derived from %c',[minimum]).

ctr_arguments(
    open_minimum,
    ['MIN'-dvar,'VARIABLES'-collection(var-dvar,bool-dvar)]).

ctr_restrictions(
    open_minimum,
    [size('VARIABLES')>0,
     required('VARIABLES',[var,bool]),
     'VARIABLES'\'bool>=0,
     'VARIABLES'\'bool=<1]).

ctr_example(
    open_minimum,
    open_minimum(3,
        [[var-3,bool-1],
         [var-1,bool-0],
         [var-7,bool-0],
         [var-5,bool-1],
         [var-5,bool-1]]).

ctr_typical(
    open_minimum,
    [size('VARIABLES')>1,range('VARIABLES'\'var)>1]).

ctr_exchangeable(
    open_minimum,
    [items('VARIABLES',all),
     translate(['MIN','VARIABLES'\'var])].

ctr_eval(open_minimum,[automaton(open_minimum_a)]).

open_minimum_a(FLAG,MIN,VARIABLES) :-
    check_type(dvar,MIN),
    collection(VARIABLES,[dvar,dvar(0,1)]),
    length(VARIABLES,N),
    N>0,
    open_minimum_signature(VARIABLES,SIGNALATURE,MIN),
AUTOMATON =
  automaton(  
    SIGNATURE,  
    _19018,  
    SIGNATURE,  
    [source(s), sink(t)],  
    [arc(s,0,s),  
      arc(s,1,t),  
      arc(s,3,s),  
      arc(s,4,s),  
      arc(s,5,s),  
      arc(t,1,t),  
      arc(t,0,t),  
      arc(t,3,t),  
      arc(t,4,t),  
      arc(t,5,t)],  
    [],  
    [],  
    []),
  automaton_bool(FLAG,[0,1,2,3,4,5],AUTOMATON).

open_minimum_signature([],[],_17117).

open_minimum_signature([ [var-VAR,bool-B]|VARs],[S|Ss],MIN) :-
  S in 0..5,
  B#=1#/\MIN#<VAR#<=S#<=>S#=0,
  B#=1#/\MIN#=VAR#<=>S#=1,
  B#=1#/\MIN#>VAR#<=>S#=2,
  B#=0#/\MIN#<VAR#<=S#<=>S#=3,
  B#=0#/\MIN#=VAR#<=>S#=4,
  B#=0#/\MIN#>VAR#<=>S#=5,
  open_minimum_signature(VARs,Ss,MIN).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.268 opposite_sign

◊ **Meta-Data:**

```prolog
ctr_predefined(opposite_sign).
ctr_date(opposite_sign,['20100821']).
ctr_origin(opposite_sign,'Arithmetic.',[]).
ctr_arguments(opposite_sign,['VAR1'-dvar,'VAR2'-dvar]).
ctr_restrictions(opposite_sign,[]).
ctr_example(opposite_sign,opposite_sign(6,-3)).
ctr_typical(opposite_sign,['VAR1'=\=-0]).
ctr_exchangeable(opposite_sign,[args([['VAR1','VAR2']]])].
ctr_eval(opposite_sign,[builtin(opposite_sign_b)]).

opposite_sign_b(VAR1,VAR2) :-
  check_type(dvar,VAR1),
  check_type(dvar,VAR2),
  VAR1#>=0#/VAR2#=<0#/VAR2#=0#/VAR1#=<0.
```

```prolog
```
B.269 or

◊ Meta-Data:

\[
\text{ctr_date(or, ['20051226'])}.
\]

\[
\text{ctr_origin(or, 'Logic', [])}.
\]

\[
\text{ctr_synonyms(or, [rel])}.
\]

\[
\text{ctr_arguments(or, ['VAR'-dvar,'VARIABLES'-collection(var-dvar)])}.
\]

\[
\text{ctr_restrictions(or,}
\]

\[
\text{'VAR'=0,}
\]

\[
\text{'VAR'=<1,}
\]

\[
\text{size('VARIABLES')>=2,}
\]

\[
\text{required('VARIABLES',var),}
\]

\[
\text{'VARIABLES'\text{-}var>=0,}
\]

\[
\text{'VARIABLES'\text{-}var=<1}].
\]

\[
\text{ctr_example(or,}
\]

\[
\text{[or(0,[[var-0],[var-0]]),}
\]

\[
\text{or(1,[[var-0],[var-1]]),}
\]

\[
\text{or(1,[[var-0],[var-0]]),}
\]

\[
\text{or(1,[[var-1],[var-1]]),}
\]

\[
\text{or(1,[[var-1],[var-0],[var-1]])}].
\]

\[
\text{ctr_exchangeable(or, [items('VARIABLES',all)])}.
\]

\[
\text{ctr_eval(or, [automaton(or_a)])}.
\]

\[
\text{ctr_pure_functional_dependency(or, [])}.
\]

\[
\text{ctr_functional_dependency(or, 1, [2])}.
\]

\[
\text{ctr_contractible(or, ['VAR'=0],'VARIABLES',any)}.
\]

\[
\text{ctr_extensible(or, ['VAR'=1],'VARIABLES',any)}.
\]

\[
\text{ctr_aggregate(or, [], [#\/,union])}.
\]

\[
\text{or_a(FLAG,VAR,VARIABLES) :-}
\]

\[
\text{check_type(dvar(0,1),VAR),}
\]

\[
\text{collection(VARIABLES, [dvar(0,1)])},
\]

\[
\text{collection(VARIABLES, [dvar(0,1)])}.
\]
length(VARIABLES, L),
L>1,
get_attr1(VARIABLES, LIST),
append([VAR], LIST, LIST_VARIABLES),
AUTOMATON=
automaton(
    LIST_VARIABLES,
    _21931,
    LIST_VARIABLES,
    [source(s), sink(i), sink(k)],
    [arc(s,0,i),
     arc(s,1,j),
     arc(i,0,i),
     arc(j,0,j),
     arc(j,1,k),
     arc(k,0,k),
     arc(k,1,k)],
    [],
    [],
    []),
automaton_bool(FLAG,[0,1],AUTOMATON).
B.270 orchard

◊ **META-DATA:**

ctr_date(orchard, ['20000128', '20030820']).

ctr_origin(orchard, '\cite{Jackson1821}', []).

ctr_arguments(
  orchard,
  ['NROW'-dvar, 'TREES'-collection(index-int, x-dvar, y-dvar)]).

ctr_restrictions(
  orchard,
  ['NROW'>=0,
   'TREES'\textasciitilde index>=1,
   'TREES'\textasciitilde index=size('TREES'),
   required('TREES', [index, x, y]),
   distinct('TREES', index),
   'TREES'\textasciitilde x>=0,
   'TREES'\textasciitilde y>=0]).

ctr_example(
  orchard,
  orchard(10, [[index-1,x-0,y-0],
   [index-2,x-4,y-0],
   [index-3,x-8,y-0],
   [index-4,x-2,y-4],
   [index-5,x-4,y-4],
   [index-6,x-6,y-4],
   [index-7,x-0,y-8],
   [index-8,x-4,y-8],
   [index-9,x-8,y-8]]).

ctr_typical(orchard, ['NROW'>0, size('TREES')>3]).

ctr_exchangeable(
  orchard,
  [items('TREES', all),
   attrs_sync('TREES', [[index],[x,y]]),
   translate(['TREES'\textasciitilde x]),
   translate(['TREES'\textasciitilde y])].

ctr_graph
orchard,
[‘TREES’],
3,
[‘CLIQUE’(<>>>collection(trees1,trees2,trees3)],
[trees1^x*trees2^y-trees1^x*trees3^y+
trees1^y*trees3^x-trees1^y*trees2^x+
trees2^x*trees3^y-trees2^y*trees3^x=
0],
[‘NARC’='NROW’],
[]).

ctr_pure_functional_dependency(orchard,[]).

ctr_functional_dependency(orchard,1,[2]).
B.271  ordered_atleast_nvector

◊ **META-DATA:**

```prolog
ctr_date(ordered_atleast_nvector,'20080921').
```

```prolog
ctr_origin(
    ordered_atleast_nvector,
    Conjoin %c and %c.,
    [atleast_nvector,lex_chain_leseq]).
```

```prolog
ctr_synonyms(
    ordered_atleast_nvector,
    [ordered_atleast_nvectors,
    ordered_atleast_npoint,
    ordered_atleast_npoints]).
```

```prolog
ctr_types(
    ordered_atleast_nvector,
    ['VECTOR'-collection(var-dvar)]).
```

```prolog
ctr_arguments(
    ordered_atleast_nvector,
    ['NVEC'-dvar,'VECTORS'-collection(vec-'VECTOR')]).
```

```prolog
ctr_restrictions(
    ordered_atleast_nvector,
    [size('VECTOR')>=1,
    'NVEC'>=0,
    'NVEC'=<size('VECTORS'),
    required('VECTORS',vec),
    same_size('VECTORS',vec)]).
```

```prolog
ctr_example(
    ordered_atleast_nvector,
    ordered_atleast_nvector(2,
        [[vec-[[var-5],[var-6]]],
        [vec-[[var-5],[var-6]]],
        [vec-[[var-5],[var-6]]],
        [vec-[[var-9],[var-3]]],
        [vec-[[var-9],[var-4]]])).
```

```prolog
ctr_typical(
    ordered_atleast_nvector,
    [size('VECTOR')>=1,
    ]).
```
'NVEC'>0,
'NVEC'<size('VECTORS'),
size('VECTORS')>1].

ctr_exchangeable(
  ordered_atleast_nvector,
  [vals([’NVEC’],int(=(0)),>,dontcare,dontcare)]).

ctr_graph(
  ordered_atleast_nvector,
  [’VECTORS’],
  2,
  [’PATH’>>collection(vectors1,vectors2)],
  [lex_lesseq(vectors1^vec,vectors2^vec)],
  [’NARC’=size(’VECTORS’)-1],
  []).

ctr_graph(
  ordered_atleast_nvector,
  [’VECTORS’],
  2,
  [’PATH’>>collection(vectors1,vectors2)],
  [lex_less(vectors1^vec,vectors2^vec)],
  [’NCC’>’NVEC’],
  []).

ctr_eval(
  ordered_atleast_nvector,
  [reformulation(ordered_atleast_nvector_r)]).

ordered_atleast_nvector_r(0,[]) :- !.

ordered_atleast_nvector_r(NVEC,VECTORS) :-
  eval(atleast_nvector(NVEC,VECTORS)),
  eval(lex_chain_lesseq(VECTORS)).
B.272 ordered_atmost_nvector

◇ Meta-Data:

ctr_date(ordered_atmost_nvector, ['20080921']).

ctr_origin(
    ordered_atmost_nvector,
    Conjoin %c and %c.,
    [atmost_nvector, lex_chain_lesseq]).

ctr_synonyms(
    ordered_atmost_nvector,
    [ordered_atmost_nvectors,
      ordered_atmost_npoint,
      ordered_atmost_npoints]).

ctr_types(
    ordered_atmost_nvector,
    ['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    ordered_atmost_nvector,
    ['NVEC'-dvar,'VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    ordered_atmost_nvector,
    [size('VECTOR')>=1,
     'NVEC'>=min(1,size('VECTORS')),
     required('VECTORS',vec),
     same_size('VECTORS',vec)]).

ctr_example(
    ordered_atmost_nvector,
    ordered_atmost_nvector(
        3,
        [[vec-[[var-5],[var-6]]],
         [vec-[[var-5],[var-6]]],
         [vec-[[var-5],[var-6]]],
         [vec-[[var-9],[var-3]]],
         [vec-[[var-9],[var-3]]]))).

ctr_typical(
    ordered_atmost_nvector,
    [size('VECTOR')>1,
     'NVEC'>1,
`NVEC'<size('VECTORS'),
size('VECTORS')>1]).

ctr_exchangeable(
ordered_atmost_nvvector,
[vals(['NVEC'],int,<,dontcare,dontcare)]).

ctr_graph(
ordered_atmost_nvvector,
['VECTORS'],
2,
['PATH'>>collection(vectors1,vectors2)],
[lex_lesseq(vectors1.vec,vectors2.vec)],
['NARC'=size('VECTORS')-1],
[]).

ctr_graph(
ordered_atmost_nvvector,
['VECTORS'],
2,
['PATH'>>collection(vectors1,vectors2)],
[lex_less(vectors1.vec,vectors2.vec)],
['NCC'='NVEC'],
[]).

ctr_eval(
ordered_atmost_nvvector,
[reformulation(ordered_atmost_nvvector_r)]).

ctr_contractible(ordered_atmost_nvvector,[],'VECTORS',any).

ordered_atmost_nvvector_r(0,[]) :-
!.

ordered_atmost_nvvector_r(NVEC,VECTORS) :-
eval(atmost_nvvector(NVEC,VECTORS)),
eval(lex_chain_lesseq(VECTORS)).
B.273 ordered_global_cardinality

♦ Meta-Data:

ctr_date(ordered_global_cardinality, ['20090911']).

ctr_origin
  ordered_global_cardinality, \cite{PetitRegin09}, []).

ctr_usual_name(ordered_global_cardinality, ordgcc).

ctr_synonyms(ordered_global_cardinality, [ordered_gc c]).

ctr_arguments
  ordered_global_cardinality,
  [‘VARIABLES’-collection(var-dvar),
   ‘VALUES’-collection(val-int,omax-int)]).

ctr_restrictions
  ordered_global_cardinality,
  [required(‘VARIABLES’,var),
   size(‘VALUES’) > 0, required(‘VALUES’,[val,omax]),
   increasing_seq(‘VALUES’,[val]),
   ‘VALUES’‘omax>=0, ‘VALUES’‘omax=<size(‘VARIABLES’)].

ctr_example
  ordered_global_cardinality,
  ordered_global_cardinality(
    [[var-2],[var-0],[var-1],[var-0],[var-0]],
    [[val-0,omax-5],[val-1,omax-3],[val-2,omax-1]]).

ctr_exchangeable
  ordered_global_cardinality,
  [items(‘VARIABLES’,all)].

ctr_graph
  ordered_global_cardinality,
  [‘VARIABLES’],
  1,
  foreach(‘VALUES’,[‘SELF’>>collection(variables)]),
  [variables‘var>=‘VALUES’‘val],
  [‘NVERTEX’=<‘VALUES’‘omax],
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

\[
\text{ctr_eval(}
\quad \text{ordered_global_cardinality,}
\quad [\text{reformulation(ordered_global_cardinality_r)}]).
\]

\[
\text{ctr_contractible(ordered_global_cardinality,[],'VALUES',any}).
\]

\[
\text{ordered_global_cardinality_r(VARIABLES,VALUES)} :-
\text{length(VARIABLES,N)},
\text{collection(VARIABLES,[dvar])},
\text{collection(VALUES,[int,int(0,N)])},
\text{length(VALUES,M)},
\text{M>0},
\text{collection_increasing_seq(VALUES,[1])},
\begin{cases}
N=0 & \rightarrow \\
\text{true} & ;
\end{cases}
\text{get_attr1(VALUES,VALS)},
\text{get_attr2(VALUES,OMAXS)},
\text{length(OCCS,M)},
\text{domain(OCCS,0,N)},
\text{create_collection(}
\quad \text{VALS,}
\quad \text{OCCS,}
\quad \text{val,}
\quad \text{nooccurrence,}
\quad \text{VALUES_GC),}
\text{eval(global_cardinality(VARIABLES,VALUES_GC))},
\text{reverse(OCCS,ROCCS)},
\text{build_sliding_sums(ROCCS,0,SUMS)},
\text{reverse(OMAXS,ROMAXS)},
\text{ordered_global_cardinality1(SUMS,ROMAXS)}
\end{cases}.
\]

\[
\text{ordered_global_cardinality1([],[]}).
\]

\[
\text{ordered_global_cardinality1([V|R],[L|S]) :-}
\quad V#=<L,
\quad \text{ordered_global_cardinality1(R,S)}.
\]
B.274 ordered_nvectors

◊ META-DATA:

ctr_date(ordered_nvectors,['20080919']).

ctr_origin(ordered_nvectors,'Derived from %c.',[nvectoretor]).

ctr_synonyms( 
    ordered_nvectors,
    [ordered_nvectors,ordered_npoint,ordered_npoints]).

ctr_types(ordered_nvectors,['VECTOR'-collection(var-dvar)]).

ctr_arguments( 
    ordered_nvectors,
    ['NVEC'-dvar,'VECTORS'-collection(vec:'VECTOR')]).

ctr_restrictions( 
    ordered_nvectors,
    [size('VECTOR')>=1, 
    'NVEC'=>min(1,size('VECTORS')), 
    'NVEC'=<size('VECTORS'), 
    required('VECTORS',vec), 
    same_size('VECTORS',vec)]).

ctr_example( 
    ordered_nvectors,
    ordered_nvectors( 
        2, 
        [[vec-[[var-5],[var-6]]], 
        [vec-[[var-5],[var-6]]], 
        [vec-[[var-5],[var-6]]], 
        [vec-[[var-9],[var-3]]], 
        [vec-[[var-9],[var-3]]]]).

ctr_typical( 
    ordered_nvectors, 
    [size('VECTOR')>1, 
    'NVEC'>1, 
    'NVEC'<size('VECTORS'), 
    size('VECTORS')>1]).

ctr_graph( 
    ordered_nvectors, 
    ['VECTORS'], 
    [...])
2,
['PATH'>>collection(vectors1,vectors2)],
[lex_lesseq(vectors1^vec,vectors2^vec)],
['NARC'=size('VECTORS')-1],
[]).
ctr_graph(
    ordered_nvector,
    ['VECTORS'],
    2,
    ['PATH'>>collection(vectors1,vectors2)],
    [lex_less(vectors1^vec,vectors2^vec)],
    ['NCC'='NVEC'],
    []).
ctr_eval(ordered_nvector,[reformulation(ordered_nvector_r)]).
ctr_functional_dependency(ordered_nvector,1,[2]).
ctr_contractible(
    ordered_nvector,
    ['NVEC'=1,size('VECTORS')>0],
    VECTORS,
    any).
ctr_contractible(
    ordered_nvector,
    ['NVEC'=size('VECTORS')],
    VECTORS,
    any).

ordered_nvector_r(0,[]) :- !.

ordered_nvector_r(NVEC,VECTORS) :-
    eval(nvector(NVEC,VECTORS)),
    eval(lex_chain_lesseq(VECTORS)).
B.275 orth_link_ori_siz_end

♦ META-DATA:

ctr_date(orth_link_ori_siz_end,['20030820','20060812']).

ctr_origin(
  orth_link_ori_siz_end,
  Used by several constraints between orthotopes, []).

ctr_arguments(
  orth_link_ori_siz_end,
  ['ORTHOTOPE'-collection(ori-dvar,siz-dvar,end-dvar)]).

ctr_restrictions(
  orth_link_ori_siz_end,
  [size('ORTHOTOPE')>0,
   require_at_least(2,'ORTHOTOPE',[ori,siz,end]),
   'ORTHOTOPE'`siz>=0,
   'ORTHOTOPE'`ori=<'ORTHOTOPE'`end]).

ctr_example(
  orth_link_ori_siz_end,
  orth_link_ori_siz_end([[ori-2,siz-2,end-4],[ori-1,siz-3,end-4]])).

ctr_typical(
  orth_link_ori_siz_end,
  [size('ORTHOTOPE')>1,'ORTHOTOPE'`siz>0]).

ctr_exchangeable(
  orth_link_ori_siz_end,
  [items('ORTHOTOPE',all),
   translate(['ORTHOTOPE`ori','ORTHOTOPE`end]),
   translate(['ORTHOTOPE`siz','ORTHOTOPE`end])].

ctr_graph(
  orth_link_ori_siz_end,
  ['ORTHOTOPE'],
  1,
  ['SELF'=>collection(orthotope)],
  [orthotope`ori+orthotope`siz=orthotope`end],
  ['NARC'=size('ORTHOTOPE')],
  []).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

```prolog
ctr_eval(
    orth_link_ori_siz_end,
    [reformulation(orth_link_ori_siz_end_r)]).

ctr_pure_functional_dependency(orth_link_ori_siz_end, []).

ctr_functional_dependency(orth_link_ori_siz_end, 1-1, [1-2, 1-3]).
ctr_functional_dependency(orth_link_ori_siz_end, 1-2, [1-1, 1-3]).
ctr_functional_dependency(orth_link_ori_siz_end, 1-3, [1-1, 1-2]).
ctr_contractible(orth_link_ori_siz_end, [], 'ORTHOTOPE', any).

orth_link_ori_siz_end_r(ORTHOTOPE) :-
    collection(ORTHOTOPE, [dvar, dvar_gteq(0), dvar]),
    length(ORTHOTOPE, N),
    N>0,
    get_attr1(ORTHOTOPE, ORIGINS),
    get_attr2(ORTHOTOPE, SIZES),
    get_attr3(ORTHOTOPE, ENDS),
    gen_varcst(ORIGINS, SIZES, ENDS).
```
B.276  orth_on_the_ground

◊ **META-DATA:**

ctr_date(orth_on_the_ground, ['20030820', '20040726', '20060812']).

ctr_origin(orth_on_the_ground, Used for defining %c., [place_in_pyramid]).

ctr_arguments(orth_on_the_ground, ['ORTHOTOPE'-collection(ori-dvar, siz-dvar, end-dvar), 'VERTICAL_DIM'-int]).

ctr_restrictions(orth_on_the_ground, [size('ORTHOTOPE')>0, require_at_least(2,'ORTHOTOPE',[ori,siz,end]), 'ORTHOTOPE'ˆsiz>=0, 'ORTHOTOPE'ˆori=<'ORTHOTOPE'ˆend, 'VERTICAL_DIM'>=1, 'VERTICAL_DIM'=<size('ORTHOTOPE'), orth_link_ori_siz_end('ORTHOTOPE')].)

ctr_example(orth_on_the_ground, orth_on_the_ground([ [ori-1,siz-2,end-3],[ori-2,siz-3,end-5]], 1)).

ctr_typical(orth_on_the_ground, [size('ORTHOTOPE')>1,'ORTHOTOPE'ˆsiz>0]).

ctr_graph(orth_on_the_ground, ['ORTHOTOPE'], 1, ['SELF']>>collection(orthotope), [orthotopeˆkey='VERTICAL_DIM',orthotopeˆori=1], ['NARC'=1], []).
B.277 orth_on_top_of_orth

◊ **Meta-Data:**

**ctr_date:**

```
ctr_date(
    orth_on_top_of_orth,
    ['20030820','20040726','20060812']).
```

**ctr_origin:**

```
ctr_origin(
    orth_on_top_of_orth,
    Used for defining %c.,
    [place_in_pyramid]).
```

**ctr_types:**

```
ctr_types(
    orth_on_top_of_orth,
    ['ORTHOTOPE'-collection(ori-dvar,siz-dvar,end-dvar)]).
```

**ctr_arguments:**

```
ctr_arguments(
    orth_on_top_of_orth,
    ['ORTHOTOPE1'-'ORTHOTOPE',
     'ORTHOTOPE2'-'ORTHOTOPE',
     'VERTICAL_DIM'-int]).
```

**ctr_restrictions:**

```
ctr_restrictions(
    orth_on_top_of_orth,
    [size('ORTHOTOPE')>0,
     require_at_least(2,'ORTHOTOPE',[ori,siz,end]),
     'ORTHOTOPE'\$siz>=0,
     'ORTHOTOPE'\$ori='ORTHOTOPE'\$end,
     size('ORTHOTOPE1')=size('ORTHOTOPE2'),
     'VERTICAL_DIM'\$>=1,
     'VERTICAL_DIM'\$size('ORTHOTOPE1'),
     orth_link_ori_siz_end('ORTHOTOPE1'),
     orth_link_ori_siz_end('ORTHOTOPE2')).
```

**ctr_example:**

```
ctr_example(
    orth_on_top_of_orth,
    orth_on_top_of_orth(
        [[ori-5,siz-2,end-7],[ori-3,siz-3,end-6]],
        [[ori-3,siz-5,end-8],[ori-1,siz-2,end-3]],
        2)).
```

**ctr_typical:**

```
ctr_typical(
    orth_on_top_of_orth,
    [size('ORTHOTOPE')>1,'ORTHOTOPE'\$siz>0]).
```
ctr_graph{
  orth_on_top_of_orth,
  ['ORTHOTOPE1','ORTHOTOPE2'],
  2,
  ['PRODUCT'(=)>>collection(orthotope1,orthotope2)],
  [orthotope1ˆkey='VERTICAL_DIM',
   orthotope2^ori=orthotope1^ori,
   orthotope1^end=orthotope2^end],
  ['NARC'=size('ORTHOTOPE1')-1],
  []).}

ctr_graph{
  orth_on_top_of_orth,
  ['ORTHOTOPE1','ORTHOTOPE2'],
  2,
  ['PRODUCT'(=)>>collection(orthotope1,orthotope2)],
  [orthotope1ˆkey='VERTICAL_DIM',
   orthotope1^ori=orthotope2^end],
  ['NARC'=1],
  []).}.
B.278 orths_are_connected

◊ Meta-Data:

ctr_date(orths_are_connected,['20000128', '20030820', '20060812']).

ctr_origin(orths_are_connected, 'N. Beldiceanu', []).

ctr_types(orths_are_connected, ['ORTHOTOPE'-collection(ori-dvar, siz-dvar, end-dvar)]).

ctr_arguments(orths_are_connected, ['ORTHOTOPES'-collection(orth-‘ORTHOTOPE’)]).

ctr_restrictions(orths_are_connected, [size('ORTHOTOPE')>0, require_at_least(2,'ORTHOTOPE',[ori,siz,end]), 'ORTHOTOPE'˜siz>0, 'ORTHOTOPE'˜ori='ORTHOTOPE'˜end, required('ORTHOTOPES',orth), same_size('ORTHOTOPES',orth)]).

ctr_example(orths_are_connected, orths_are_connected([orth-[[ori-2,siz-4,end-6],[ori-2,siz-2,end-4]]], [orth-[[ori-1,siz-2,end-3],[ori-4,siz-3,end-7]]], [orth-[[ori-7,siz-4,end-11],[ori-1,siz-2,end-3]]], [orth-[[ori-6,siz-2,end-8],[ori-3,siz-2,end-5]]])).

ctr_typical(orths_are_connected, [size('ORTHOTOPE')>1, size('ORTHOTOPES')>1]).

ctr_exchangeable(orths_are_connected, [items('ORTHOTOPES', all), items_sync('ORTHOTOPES'˜orth, all), translate(['ORTHOTOPES'˜orth'ori,'ORTHOTOPES'˜orth'end])]).

ctr_graph(}
orths_are_connected,
[‘ORTHOTOPES’],
1,
[‘SELF’>>collection(orthotopes)],
[orth_link_ori_siz_end(orthotopes^orth)],
[‘NARC’=size(‘ORTHOTOPES’)],
[[]].

ctr_graph{
  orths_are_connected,
  [‘ORTHOTOPES’],
  2,
  [‘CLIQUE’(\=\=)>>collection(orthotopes1, orthotopes2)],
  [two_orth_are_in_contact(
    orthotopes1^orth,
    orthotopes2^orth)],
  [‘NVERTEX’=size(‘ORTHOTOPES’), ‘NCC’=1],
  [[]].
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.279 overlap_sboxes

◊ **Meta-Data:**

```prolog
ctr_date(overlap_sboxes,['20070622','20090725']).
```

```prolog
ctr_origin(overlap_sboxes,
    Geometry, derived from \cite{RandellCuiCohn92}, []).
```

```prolog
ctr_synonyms(overlap_sboxes,[overlap]).
```

```prolog
ctr_types(overlap_sboxes,
    ['VARIABLES'-collection(v-dvar),
     'INTEGERS'-collection(v-int),
     'POSITIVES'-collection(v-int)]).
```

```prolog
ctr_arguments(overlap_sboxes,
    ['K'-int,
     'DIMS'-sint,
     'OBJECTS'-collection(oid-int,sid-int,x-'VARIABLES'),
     'SBOXES'-collection(sid-int,t-'INTEGERS',l-'POSITIVES')]).
```

```prolog
ctr_restrictions(overlap_sboxes,
    [size('VARIABLES')>=1,
     size('INTEGERS')>=1,
     size('POSITIVES')>=1,
     required('VARIABLES',v),
     size('VARIABLES')='K',
     required('INTEGERS',v),
     size('INTEGERS')='K',
     required('POSITIVES',v),
     size('POSITIVES')='K',
     'POSITIVES'~v>0,
     'K'>0,
     'DIMS'>=0,
     'DIMS'<'K',
     increasing_seq('OBJECTS',[oid]),
     required('OBJECTS',[oid,sid,x]),
     'OBJECTS'~oid>=1,
     'OBJECTS'~oid=<size('OBJECTS'),
     'OBJECTS'~sid>=1,
```
'OBJECTS' \( \# \text{sid} < \text{size('SBOXES')}, \text{size('SBOXES')} \geq 1, \text{required('SBOXES', [sid,t,l]), 'SBOXES' \# \text{sid} = 1, 'SBOXES' \# \text{sid} < \text{size('SBOXES')}, \text{do_not_overlap('SBOXES')}.\)

\[\text{ctr_example(\text{overlap_sboxes), overlap_sboxes(2, \{0,1\}, [[oid-1,sid-1,x-[[v-1],[v-1]]], [oid-2,sid-2,x-[[v-3],[v-2]]], [oid-3,sid-3,x-[[v-2],[v-4]]], [sid-1,t-[[v-0],[v-0]],l-[[v-4],[v-5]]], [sid-2,t-[[v-0],[v-0]],l-[[v-3],[v-3]]], [sid-3,t-[[v-0],[v-0]],l-[[v-2],[v-1]]]).}\]

\[\text{ctr_typical(overlap_sboxes, [size('OBJECTS') \# 1]).}\]

\[\text{ctr_exchangeable(overlap_sboxes, [items('OBJECTS',all), items('SBOXES',all), items_sync('OBJECTS' \# x,'SBOXES' \# t,'SBOXES' \# l,all), vals([\text{\# SBOXES' \# l \# v}], \text{int}, <, \text{dontcare}, \text{dontcare})).}\]

\[\text{ctr_eval(overlap_sboxes, [logic(overlap_sboxes_g)]).}\]

\[\text{ctr_logic(overlap_sboxes, [DIMENSIONS,OIDS], (\text{origin(O1,S1,D)---\# O1'\# x(D)\# + S1'\# t(D))}, (\text{end(O1,S1,D)---\# O1'\# x(D)\# + S1'\# t(D)\# + S1'\# l(D))}, (\text{overlap_sboxes(Dims, O1,S1,O2,S2)---\# forall(D, Dims, end(O1,S1,D)\# \# origin(O2,S2,D)\# /\ end(O2,S2,D)\# \# origin(O1,S1,D))}, (\text{overlap_objects(Dims, O1,O2)---\# forall(S1, sboxes([O1'\# sid]), exists(\})}}\]
\[
\begin{align*}
S_2, \\
\text{sboxes([O2^sid]),} \\
\text{overlap_sboxes(Dims,O1,S1,O2,S2))}), \\
\text{(all_overlap(Dims,OIDS)---+)} \\
\text{forall(} \\
\text{ O1,} \\
\text{ objects(OIDS),} \\
\text{forall(} \\
\text{ O2,} \\
\text{ objects(OIDS),} \\
\text{ O1^oid#<O2^oid#=>overlap_objects(Dims,O1,O2))),} \\
\text{all_overlap(DIMENSIONS,OIDS)).}
\end{align*}
\]

ctr_contractible(overlap_sboxes,[],'OBJECTS',suffix).

\text{overlap_sboxes_g(K,_28765,[],_28767) :-} \\
\text{!,} \\
\text{check_type(int_gteq(1),K).}

\text{overlap_sboxes_g(K,_DIMS,OBJECTS,SBOXES) :-} \\
\text{length(OBJECTS,O),} \\
\text{length(SBOXES,S),} \\
\text{O>0,} \\
\text{S>0,} \\
\text{check_type(int_gteq(1),K),} \\
\text{collection(OBJECTS,[int(1,O),dvar(1,S),col(K,[dvar])]},) \\
\text{collection(} \\
\text{ SBOXES,} \\
\text{ [int(1,S),col(K,[int]),col(K,[int_gteq(1)]))]))) \\
\text{get_attr1(OBJECTS,OIDS),} \\
\text{get_attr2(OBJECTS,SID),} \\
\text{get_col_attr3(OBJECTS,1,XS),} \\
\text{get_attr1(SBOXES,SID),} \\
\text{get_col_attr2(SBOXES,1,TS),} \\
\text{get_col_attr3(SBOXES,1,TL),} \\
\text{collection_increasing_seq(OBJECTS,[1]),} \\
\text{geost1(OIDS,SID,XS,Objects),} \\
\text{geost2(SID,S,TL,SBboxes),} \\
\text{geost_dims(1,K,DIMENSIONS),} \\
\text{ctr_logic(overlap_sboxes,[DIMENSIONS,OIDS],Rules),} \\
\text{geost(Objects,SBboxes,[overlap(true)],Rules).}
\]
B.280  path

◊ META-DATA:

ctr_date(path, [‘20090101’]).

ctr_origin(path, ‘Derived from %c.’, [binary_tree]).

ctr_arguments(
    path,
    [‘NPATH’-dvar, ‘NODES’-collection(index-int, succ-dvar)]).

ctr_restrictions(
    path,
    [‘NPATH’>=1,
     ‘NPATH’=<size(‘NODES’),
     required(‘NODES’, [index, succ]),
     size(‘NODES’) > 0,
     ‘NODES’^index>=1,
     ‘NODES’^index=<size(‘NODES’),
     distinct(‘NODES’, index),
     ‘NODES’^succ>=1,
     ‘NODES’^succ=<size(‘NODES’)]).

ctr_example(
    path,
    path( 3,
        [[index-1, succ-1],
         [index-2, succ-3],
         [index-3, succ-5],
         [index-4, succ-7],
         [index-5, succ-1],
         [index-6, succ-6],
         [index-7, succ-7],
         [index-8, succ-6]])).

ctr_typical(path, [‘NPATH’<size(‘NODES’), size(‘NODES’)>1]).

ctr_exchangeable(path, [items(‘NODES’, all)]).

ctr_graph(
    path,
    [‘NODES’],
    2,
    [‘CLIQUE’>>collection(nodes1, nodes2)]),
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

[nodes1\textasciitilde succ=nodes2\textasciitilde index],
[\textquoteleft\textquoteleft MAX_NSCC\textquoteright\textquoteright<1,\textquoteleft\textquoteleft NCC\textquoteright\textquoteright=\textquoteleft\textquoteleft NPATH\textquoteright\textquoteright,\textquoteleft\textquoteleft MAX_ID\textquoteright\textquoteright<1],
[\textquoteleft\textquoteleft ONE_SUCC\textquoteright\textquoteright]).

\texttt{ctr\_eval(path, [reformulation(path\_r)])}.

\texttt{ctr\_functional\_dependency(path, 1, [2])}.

\texttt{path\_r(NPATH, NODES) :-
  eval(tree(NPATH, NODES)),
  get\_attr1(NODES, INDEXES),
  get\_attr2(NODES, SUCCS),
  k\_ary\_tree(INDEXES, INDEXES, SUCCS, 1).}
B.281 path_from_to

◊ META-DATA:

ctr_date(path_from_to,['20030820','20040530','20060812']).

ctr_origin(
    path_from_to,
    \cite{AlthausBockmayrElfKasperJungerMehlhorn02}, [1]).

ctr_usual_name(path_from_to,path).

ctr_arguments(
    path_from_to,
    ['FROM'-int,
     'TO'-int,
     'NODES'-collection(index-int,succ-svar)]).

ctr_restrictions(
    path_from_to,
    ['FROM'>=1,
     'FROM'=<size('NODES'),
     'TO'>=1,
     'TO'=<size('NODES'),
     required('NODES',{index,succ}),
     'NODES'\^index>=1,
     'NODES'\^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'\^succ>=1,
     'NODES'\^succ=<size('NODES'))].

ctr_example(
    path_from_to,
    path_from_to(
        4,
        3,
        [[index-1,succ-{}],
         [index-2,succ-{}],
         [index-3,succ-{5}],
         [index-4,succ-{5}],
         [index-5,succ-{2,3}]]).

ctr_typical(path_from_to,['FROM'\='TO',size('NODES')>2]).

ctr_exchangeable(path_from_to,[items('NODES',all)]).
ctr_graph(
    path_from_to,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes2`index in_set nodes1`succ],
    ['PATH_FROM_TO'(index,'FROM','TO')=1],
    []).

\subsection{pattern}

\textbf{\textit{Meta-Data:}}

\begin{verbatim}
ctr_date(pattern, ['20031008', '20090717']).
ctr_origin(pattern, \cite{BourdaisGalinierPesant03}, []).
ctr_types(pattern, ['PATTERN'-collection(var-int)]).

ctr_arguments(pattern, ['VARIABLES'-collection(var-dvar),
                      'PATTERNS'-collection(pat-'PATTERN')]).

ctr_restrictions(pattern, [required('PATTERN', var),
                           'PATTERN'\^var>=0,
                           change(0,'PATTERN',=),
                           size('PATTERN')>1,
                           required('VARIABLES', var),
                           required('PATTERNS', pat),
                           size('PATTERNS')>0,
                           same_size('PATTERNS', pat)]).

ctr_example(pattern, pattern(
                  [[var-1],
                   [var-1],
                   [var-2],
                   [var-2],
                   [var-2],
                   [var-1],
                   [var-3],
                   [var-3]],
                  [[pat-[[var-1], [var-2], [var-1]]],
                   [pat-[[var-1], [var-2], [var-3]]],
                   [pat-[[var-2], [var-1], [var-3]]])).

ctr_typical(pattern, [size('VARIABLES')>2, range('VARIABLES'\^var)>1]).

ctr_exchangeable(
\end{verbatim}
pattern,
[items('PATTERNS',all),
  items_sync('VARIABLES','PATTERNS'\^pat,reverse),
  vals(
    ['VARIABLES'\^var,'PATTERNS'\^pat\^var],
    int,
    =\=,
    all,
    dontcare))).

ctr_eval(pattern,[automaton(pattern_a)]).

ctr_contractible(pattern,[],'VARIABLES',prefix).

ctr_contractible(pattern,[],'VARIABLES',suffix).

pattern_a(FLAG,VARIABLES,PATTERNS) :-
collection(VARIABLES,[dvar]),
collection(PATTERNS,[col([int_gteq(0)])]),
same_size(PATTERNS),
length(PATTERNS,NPATTERNS),
NPATTERNS>0,
get_attr1(VARIABLES,VARS),
get_col_attr1(PATTERNS,1,PATTS),
PATTS=[PATT|_22270],
length(PATT,K),
K>1,
pattern_change(PATTERNS),
remove_duplicates(PATTS,PATTS_NO_DUPLICATES),
pattern_build_tree(
  PATTS_NO_DUPLICATES,
  ID_PATTS,
  node(-1-0,[]),
  1,
  _25109,
  TREE),
flattern(PATTS_NO_DUPLICATES,FLAT_PATTS),
remove_duplicates(FLAT_PATTS,VALUES),
pattern_next(
  ID_PATTS,
  ID_PATTS,
  VALUES,
  ADDITIONAL_TRANSITIONS),
pattern_gen_states(TREE,STATES,TRANSITIONS),
append( TRANSITIONS,
ADDITIONAL_TRANSITIONS, ALL_TRANSITIONS),
AUTOMATON=
automaton(
  VARS, _27181, VARS, STATES, ALL_TRANSITIONS, []),
automaton_bool(FLAG, VALUES, AUTOMATON).

pattern_gen_states(
  node(-1-0, LIST_SUNS), [source(NAME), sink(NAME) | R],
  TRANSITIONS) :-
  !,
  number_codes(-1, CODE),
  atom_codes(ATOM, CODE),
  atom_concat(s, ATOM, NAME),
  pattern_gen_states1(LIST_SUNS, -1, R, TRANSITIONS).

pattern_gen_states(
  node(ID-_VAL, LIST_SUNS), [sink(NAME) | R],
  TRANSITIONS) :-
  ID>=0,
  number_codes(ID, IDCODE),
  atom_codes(IDATOM, IDCODE),
  atom_concat(s, IDATOM, NAME),
  pattern_gen_states1(LIST_SUNS, ID, R, TRANSITIONS).

pattern_gen_states1([], _22194, [], []).

pattern_gen_states1([N|R], ID1, ST, TRANSITIONS) :-
  N=node(ID2-VAL2, _22220),
  pattern_gen_states(N, S, TRANSITIONS1),
  pattern_gen_states1(R, ID1, T, TRANSITIONS2),
  append(S, T, ST),
  number_codes(ID1, IDCODE1),
  atom_codes(IDATOM1, IDCODE1),
  atom_concat(s, IDATOM1, IDNAME1),
  number_codes(ID2, IDCODE2),
  atom_codes(IDATOM2, IDCODE2),
atom_concat(s,IDATOM2,IDNAME2),
append(
    [arc(IDNAME1,VAL2,IDNAME2),
     arc(IDNAME2,VAL2,IDNAME2)],
    TRANSITIONS1,
    T1),
append(T1,TRANSITIONS2,TRANSITIONS).

pattern_change([]).

pattern_change([[]?22202-P|R]) :-
    eval(change(0,P,=)),
    pattern_change(R).

pattern_next([],_22194,_22195,[]).

pattern_next([PID-P|R],ID_PATTS,VALUES,TRANSITIONS) :-
    P=[_22223|RP],
    pattern_next1(VALUES,PID,RP,ID_PATTS,TRANSITIONS1),
    pattern_next(R,ID_PATTS,VALUES,TRANSITIONS2),
    append(TRANSITIONS1,TRANSITIONS2,TRANSITIONS).

pattern_next1([],_22194,_22195,_22196,[]).

pattern_next1([V|R],PID,RP,ID_PATTS, [arc(PIDNAME,V,NEWPIDNAME)|S]) :-
    append(RP,[V],NEWP),
    pattern_search(ID_PATTS,NEWP,NEWPID),
    number_codes(PID,PIDCODE),
    atom_codes(PIDATOM,PIDCODE),
    atom_concat(s,PIDATOM,PIDNAME),
    number_codes(NEWPID,NEWPIDCODE),
    atom_codes(NEWPIDATOM,NEWPIDCODE),
    atom_concat(s,NEWPIDATOM,NEWPIDNAME),
    !,
    pattern_next1(R,PID,RP,ID_PATTS,S).

pattern_next1([_22201|R],PID,RP,ID_PATTS, S) :-
    pattern_next1(R,PID,RP,ID_PATTS, S).

pattern_search([ID-PAT|_22200],PAT,ID) :-
!.
pattern_search([R|PAT], PAT, ID) :-
    pattern_search(R, PAT, ID).

pattern_build_tree([], [], TREE, NODE_ID, NODE_ID, TREE).

pattern_build_tree(
    [PATTERN|R],
    [PATTERN_ID-PATTERN|S],
    OLD_TREE,
    OLD_NODE_ID,
    NEW_NODE_ID,
    NEW_TREE) :-
    pattern_insert(
        PATTERN,
        OLD_TREE,
        OLD_NODE_ID,
        CUR_NODE_ID,
        CUR_TREE,
        PATTERN_ID),
    pattern_build_tree(
        R,
        S,
        CUR_TREE,
        CUR_NODE_ID,
        NEW_NODE_ID,
        NEW_TREE).

pattern_insert([], TREE, NODE_ID, NODE_ID, TREE, _22198).

pattern_insert(
    [I|R],
    OLD_TREE,
    OLD_NODE_ID,
    NEW_NODE_ID,
    node(LABEL, NEW_TREE),
    PATTERN_ID) :-
    OLD_TREE=node(LABEL, LIST_NODES),
    pattern_occurs(I, LIST_NODES, [], BEFORE, SUBTREE, AFTER), !,
    pattern_insert(
        R,
        SUBTREE,
        OLD_NODE_ID,
        NEW_NODE_ID,
        NEW_SUBTREE,
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

\[
\text{pattern_insert}( \\
\qquad [I|R], \\
\quad \text{node}((\text{LABEL}, \text{LIST\_NODES}), \\
\quad \text{OLD\_NODE\_ID}, \\
\quad \text{NEW\_NODE\_ID}, \\
\quad \text{node}((\text{LABEL}, [\text{BRANCH}\mid \text{LIST\_NODES}]), \\
\quad \text{PATTERN\_ID}) : - \\
\quad \text{pattern\_create\_branch}( \\
\qquad [I|R], \\
\qquad \text{OLD\_NODE\_ID}, \\
\qquad \text{NEW\_NODE\_ID}, \\
\qquad \text{BRANCH}, \\
\qquad \text{PATTERN\_ID}). \\
\]

\[
\text{pattern\_create\_branch}( \\
\qquad [I], \\
\quad \text{OLD\_NODE\_ID}, \\
\quad \text{NEW\_NODE\_ID}, \\
\quad \text{node}((\text{OLD\_NODE\_ID}-I, []), \\
\quad \text{OLD\_NODE\_ID}) : - \\
\quad \text{NEW\_NODE\_ID} \text{ is } \text{OLD\_NODE\_ID}+1. \\
\]

\[
\text{pattern\_create\_branch}( \\
\qquad [I,J|R], \\
\quad \text{OLD\_NODE\_ID}, \\
\quad \text{NEW\_NODE\_ID}, \\
\quad \text{node}((\text{OLD\_NODE\_ID}-I, [S]), \\
\quad \text{OLD\_NODE\_ID}) : - \\
\quad \text{CUR\_NODE\_ID} \text{ is } \text{OLD\_NODE\_ID}+1, \\
\quad \text{pattern\_create\_branch}( \\
\qquad [J|R], \\
\qquad \text{CUR\_NODE\_ID}, \\
\qquad \text{NEW\_NODE\_ID}, \\
\qquad \text{S}, \\
\qquad \text{PATTERN\_ID}). \\
\]

\[
\text{pattern\_occurs}( \\
\qquad I, \\
\quad [\text{node}(\text{Id}-I, L) \mid \text{AFTER}], \\
\quad \text{BEFORE}, \\
\quad \text{BEFORE}, \\
\quad \text{BEFORE}, \\
\quad \text{BEFORE}). \\
\]

\[
\text{append}((\text{BEFORE}, [\text{NEW\_SUBTREE}], \text{TEMPO\_TREE}), \\
\text{append}(\text{TEMPO\_TREE}, \text{AFTER}, \text{NEW\_TREE})). \\
\]
node(Id-I,L),
AFTER) :- !.

pattern_occurs(I,
  [NODE|AFTER_CUR],
  BEFORE_CUR,
  BEFORE,
  NODE_FOUND,
  AFTER) :-
  pattern_occurs(I,
      AFTER_CUR,
      [NODE|BEFORE_CUR],
      BEFORE,
      NODE_FOUND,
      AFTER).
B.283 peak

◊ **Meta-Data:**

```
ctr_date(peak, ['20040530']).

ctr_origin(peak, 'Derived from %c.', [inflexion]).

ctr_arguments(peak, ['N'-dvar, 'VARIABLES'-collection (var-dvar)]).

ctr_restrictions(
    peak,
    ['N'>=0,
     2*N'=<max(size('VARIABLES')-1,0),
     required('VARIABLES', var))).

ctr_example(
    peak,
    peak(
        2,
        [[var-1],
         [var-1],
         [var-4],
         [var-8],
         [var-6],
         [var-2],
         [var-7],
         [var-1]]).

ctr_typical(
    peak,
    [size('VARIABLES')>2, range('VARIABLES'~var)>1]).

ctr_exchangeable(
    peak,
    [items('VARIABLES', reverse), translate([VARIABLES~var])]).

ctr_eval(peak, [automaton(peak_a)]).

ctr_contractible(peak, ['N'=0], 'VARIABLES', any).
```

```
peak_a(FLAG,N,VARIABLES) :-
    check_type(dvar_gteq(0),N),
    collection(VARIABLES, [dvar]),
    length(VARIABLES, L),
    MAX is max(L-1, 0),
```
\[ 2 \cdot N^# \leq \text{MAX}, \]
\[ \text{peak_signature} (\text{VARIABLES}, \text{SIGNATURE}), \]
\[ \text{automaton} ( \]
\[ \text{SIGNATURE}, \]
\[ _19173, \]
\[ \text{SIGNATURE}, \]
\[ [\text{source}(s), \text{sink}(u), \text{sink}(s)], \]
\[ [\text{arc}(s, 0, s), \]
\[ \text{arc}(s, 1, s), \]
\[ \text{arc}(s, 2, u), \]
\[ \text{arc}(u, 0, s, [C+1]), \]
\[ \text{arc}(u, 1, u), \]
\[ \text{arc}(u, 2, u)], \]
\[ [C], \]
\[ [0], \]
\[ \text{COUNT}]), \]
\[ \text{COUNT}^# = N^# \iff \text{FLAG}. \]

\text{peak_signature}([], []).

\text{peak_signature}([_17407], []) :-
  !.

\text{peak_signature}([[\text{var-VAR1}], [\text{var-VAR2}] | \text{VARs}], [S | \text{Ss}]) :-
  S \text{ in } 0..2,
  \text{VAR1}^# > \text{VAR2}^# \iff S^# = 0,
  \text{VAR1}^# = \text{VAR2}^# \iff S^# = 1,
  \text{VAR1}^# < \text{VAR2}^# \iff S^# = 2,
  \text{peak_signature}([[\text{var-VAR2}] | \text{VARs}], \text{Ss}).
B.284  period

◊ **META-DATA:**

```prolog
ctr_predefined(period).

ctr_date(period, ['20000128', '20030820', '20040530', '20060812']).

ctr_origin(period, 'N.˘Beldiceanu', []).

ctr_arguments(
  period,
  ['PERIOD'-dvar,
   'VARIABLES'-collection(var-dvar),
   'CTR'-atom]).

ctr_restrictions(
  period,
  ['PERIOD'>=1,
   'PERIOD'=<size('VARIABLES'),
   required('VARIABLES', var),
   in_list('CTR', [=, =\=, <, >, >=, =<])).

ctr_example(
  period,
  period(3,
    [[var-1],
     [var-1],
     [var-4],
     [var-1],
     [var-1],
     [var-4],
     [var-1],
     [var-1],
     [var-1]],
    =)).

ctr_typical(
  period,
  ['PERIOD'>1,
   'PERIOD'<size('VARIABLES'),
   size('VARIABLES')>2,
   range('VARIABLES'^var)>1,
   in_list('CTR', [=])).

ctr_exchangeable(
```
period,
  [items('VARIABLES',reverse),
  items('VARIABLES',shift),
  vals(['VARIABLES'~var],int,=\=,all,dontcare)]).

ctr_eval(period,[reformulation(period_r)]).

ctr_pure_functional_dependency(period,[]).

ctr_functional_dependency(period,1,[2,3]).

ctr_contractible(
  period,
  [in_list('CTR',[=]),'PERIOD'=1],
  VARIABLES,
  any).

ctr_contractible(period,[],'VARIABLES',prefix).

ctr_contractible(period,[],'VARIABLES',suffix).

period_r(PERIOD,VARIABLES,CTR) :-
  check_type(dvar,PERIOD),
  collection(VARIABLES,[dvar]),
  memberchk(CTR,[=,=\=,<,>=,>,=<]),
  length(VARIABLES,N),
  PERIOD#=\=1,
  PERIOD#=<N,
  get_attr1(VARIABLES,VARS),
  period1(N,VARS,LISTS),
  period4(LISTS,1,CTR,BOOLES),
  reverse(BOOLES,RBOOLES),
  period7(RBOOLES,1,PERIOD,1,EXPR),
  call(EXPR).
B.285  period_except_0

◊ Meta-Data:

ctr_predefined(period_except_0).

ctr_date(period_except_0, ['20030820', '20040530', '20060813']).

ctr_origin(period_except_0, 'Derived from %c.', [period]).

ctr_arguments( period_except_0, ['PERIOD'-dvar, 'VARIABLES'-collection(var-dvar), 'CTR'-atom]).

ctr_restrictions( period_except_0, ['PERIOD'>=1, 'PERIOD'=<size('VARIABLES'), required('VARIABLES', var), in_list('CTR', [=, =\=, <, >, >=, =\<])).

ctr_example( period_except_0, period_except_0( 3, [[var-1], [var-1], [var-4], [var-1], [var-1], [var-0], [var-1], [var-1], [var-1]], =)).

ctr_typical( period_except_0, ['PERIOD'>1, 'PERIOD'<size('VARIABLES'), size('VARIABLES')>2, range('VARIABLES'\^var)>1, atleast(1, 'VARIABLES', 0), in_list('CTR', [=])).
ctr_exchangeable(
    period_except_0,
    [items('VARIABLES', reverse),
      items('VARIABLES', shift),
      vals(['VARIABLES'\textsuperscript{\textasciitilde}var], int(\textasciitilde\textasciitilde0), =\textasciitilde\textasciitilde, all, dontcare)]).

ctr_eval(period_except_0, [reformulation(period_except_0_r)]).

ctr_pure_functional_dependency(period_except_0, []).

ctr_functional_dependency(period_except_0, 1, [2, 3]).

ctr_contractible(
    period_except_0,
    [in_list('CTR', [=]), 'PERIOD'=1],
    VARIABLES, any).

ctr_contractible(period_except_0, [], 'VARIABLES', prefix).

ctr_contractible(period_except_0, [], 'VARIABLES', suffix).

\texttt{period\_except\_0\_r(PERIOD, VARIABLES, CTR) :-}
    check_type(dvar, PERIOD),
    collection(VARIABLES, [dvar]),
    memberchk(CTR, [=, \textasciitilde\textasciitilde, <, \textasciitilde\textasciitilde, \textasciitilde\textasciitilde, \textasciitilde\textasciitilde, \textasciitilde\textasciitilde]),
    length(VARIABLES, N),
    PERIOD\#\textasciitilde=1,
    PERIOD\#\textasciitilde=N,
    get_attr1(VARIABLES, VARS),
    period1(N, VARS, LISTS),
    period4(LISTS, 0, CTR, BOOLS),
    reverse(BOOLS, RBOOLS),
    period7(RBOOLS, 1, PERIOD, 1, EXPR),
    call(EXPR).}
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.286 period_vectors

◊ **Meta-Data:**

```prolog
ctr_predefined(period_vectors).
ctr_date(period_vectors, [’20110614’]).
ctr_origin(period_vectors, ’Derived from %c’, [period]).
ctr_types(
    period_vectors,
    [’VECTOR’-collection(var-dvar), ’CTR’-atom]).
ctr_arguments(
    period_vectors,
    [’PERIOD’-dvar,
     ’VECTORS’-collection(vec-’VECTOR’),
     ’CTRS’-collection(ctr-’CTR’)]).
ctr_restrictions(
    period_vectors,
    [size(’VECTOR’)>=1,
     required(’VECTOR’, var),
     in_list(’CTR’, [=, \=, <, >=, >, <=]),
     ’PERIOD’>=1,
     ’PERIOD’=<size(’VECTORS’),
     required(’VECTORS’, vec),
     same_size(’VECTORS’, vec),
     required(’CTRS’, ctr),
     size(’CTRS’)=size(’VECTOR’)]).
ctr_example(
    period_vectors,
    period_vectors(3,
        [[[vec-[[var-1], [var-0]]],
          [vec-[[var-1], [var-5]]],
          [vec-[[var-4], [var-4]]],
          [vec-[[var-1], [var-0]]],
          [vec-[[var-1], [var-5]]],
          [vec-[[var-4], [var-4]]],
          [vec-[[var-1], [var-0]]],
          [vec-[[var-1], [var-5]]],
          [[ctr- =], [ctr- =]]))).
```
ctr_typical(period_vectors,
    [in_list('CTR',[=]),
      size('VECTOR')>1,
      'PERIOD'>1,
      'PERIOD'<size('VECTORS'),
      size('VECTORS')>2]).

ctr_exchangeable(period_vectors,[items('VECTORS',reverse)]).

ctr_eval(period_vectors,[reformulation(period_vectors_r)]).

ctr_pure_functional_dependency(period_vectors,[]).

ctr_functional_dependency(period_vectors,1,[2,3]).

ctr_contractible(period_vectors,[],'VECTORS',prefix).

ctr_contractible(period_vectors,[],'VECTORS',suffix).

period_vectors_r(PERIOD,VECTORS,CTRS) :-
    check_type(dvar,PERIOD),
    collection(VECTORS,[col([dvar])]),
    collection(CTRS,[atom([=,\=,<,>,>=,<=])]),
    length(VECTORS,N),
    PERIOD#>=1,
    PERIOD#=<N,
    get_attr1(VECTORS,VECTS),
    get_attr1(CTRS,LCTRS),
    period1(N,VECTS,LISTS),
    period4(LISTS,2,LCTRS,BOOLES),
    reverse(BOOLES,RBOOLES),
    period7(RBOOLES,1,PERIOD,1,EXPR),
    call(EXPR).
B.287 permutation

◊ Meta-Data:

```prolog
ctr_date(permutation,['20111210']).

ctr_origin(
    permutation,
    Derived from %c.,
    [alldifferent_consecutive_values]).

ctr_arguments(permutation,['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    permutation,
    [required('VARIABLES',var),
    minval('VARIABLES'\^var)=1,
    maxval('VARIABLES'\^var)=size('VARIABLES')]).

ctr_example(
    permutation,
    permutation([[var-3],[var-2],[var-1],[var-4]])).

ctr_typical(permutation,[size('VARIABLES')>2]).

ctr_exchangeable(
    permutation,
    [items('VARIABLES',all),
    vals(['VARIABLES'\^var],int,\=,all,in)]).

ctr_graph(
    permutation,
    ['VARIABLES'],
    2,
    ['CLIQUE'\>collection(variables1,variables2)],
    [variables1\^var=variables2\^var],
    ['MAX_NSCC'=<1],
    ['ONE_SUCC']).

ctr_eval(
    permutation,
    [checker(permutation_c),reformulation(permutation_r)]).

permutation_c([]) :-
!.
```
permutation_c(VARIABLES) :-
    collection(VARIABLES,[int]),
    get_attr1(VARIABLES,VARS),
    min_member(MIN,VARS),
    MIN=1,
    length(VARS,N),
    max_member(MAX,VARS),
    MAX=N,
    sort(VARS,SVARS),
    length(SVARS,N).

permutation_r([]) :-
    !.

permutation_r(VARIABLES) :-
    collection(VARIABLES,[dvar]),
    get.attr1(VARIABLES,VARS),
    all_different(VARS),
    minimum(1,VARS),
    length(VARIABLES,N),
    maximum(N,VARS).
B.288 place_in_pyramid

◊ Meta-Data:

\[
\begin{align*}
\text{ctr}_\text{date}( & \text{place_in_pyramid}, \\
& \text{[''20000128'', ''20030820'', ''20041230'', ''20060813'']].
\end{align*}
\]

\[
\begin{align*}
\text{ctr}_\text{origin}( & \text{place_in_pyramid}, 'N.\text{'}\text{Beldiceanu}', []).
\end{align*}
\]

\[
\begin{align*}
\text{ctr}_\text{types}( & \text{place_in_pyramid}, \\
& \text{[''ORTHOTOPE''-collection(ori-dvar,siz-dvar,end-dvar)']}).
\end{align*}
\]

\[
\begin{align*}
\text{ctr}_\text{arguments}( & \text{place_in_pyramid}, \\
& \text{[''ORTHOTOPE''-collection(orth-'ORTHOTOPE'),} \\
& \text{'VERTICAL\_DIM'-int}]).
\end{align*}
\]

\[
\begin{align*}
\text{ctr}_\text{restrictions}( & \text{place_in_pyramid}, \\
& \text{[size('ORTHOTOPE')}>0,} \\
& \text{require_at_least(2,'ORTHOTOPE',[ori,siz,end]),} \\
& \text{'ORTHOTOPE''siz}>0,} \\
& \text{'ORTHOTOPE''ori=''} \text{<'} \text{ORTHOTOPE''end,} \\
& \text{required('ORTHOTOPE',orth),} \\
& \text{same_size('ORTHOTOPE',orth),} \\
& \text{'VERTICAL\_DIM'}=1,} \\
& \text{diffn('ORTHOTOPE')}]).
\end{align*}
\]

\[
\begin{align*}
\text{ctr}_\text{example}( & \text{place_in_pyramid,} \\
& \text{place_in_pyramid(} \\
& \text{[[orth-[[ori-1,siz-3,end-4],[ori-1,siz-2,end-3]]],} \\
& \text{[orth-[[ori-1,siz-2,end-3],[ori-3,siz-3,end-6]]],} \\
& \text{[orth-[[ori-5,siz-6,end-11],[ori-1,siz-2,end-3]]],} \\
& \text{[orth-[[ori-5,siz-2,end-7],[ori-3,siz-2,end-5]]],} \\
& \text{[orth-[[ori-8,siz-3,end-11],[ori-3,siz-2,end-5]]],} \\
& \text{[orth-[[ori-8,siz-2,end-10],[ori-5,siz-2,end-7]]]]],} \\
& \text{2})].
\end{align*}
\]

\[
\begin{align*}
\text{ctr}_\text{typical}( & \text{place_in_pyramid,} \\
& \text{[size('ORTHOTOPE')}>1,} \\
& \text{'ORTHOTOPE''siz}>0,} \\
& \text{size('ORTHOTOPE')}>1]).
\end{align*}
\]
2701

ctr_exchangeable(place_in_pyramid, [items('ORTHOTOPES', all)]).

ctr_graph(
    place_in_pyramid,
    ['ORTHOTOPES'],
    2,
    ['CLIQUE'>>collection(orthotopes1, orthotopes2)],
    [orthotopes1^key=orthotopes2^key\/
     orth_on_the_ground(orthotopes1^orth, 'VERTICAL_DIM')\/
     orthotopes1^key=\=orthotopes2^key\/
     orth_on_top_of_orth(
         orthotopes1^orth,
         orthotopes2^orth,
         VERTICAL_DIM)],
    ['NARC'=size('ORTHOTOPES')],
    []).
B.289  polyomino

◊ Meta-Data:

`ctr_date(polyomino,[‘20000128’,‘20030820’,‘20060813’]).`

`ctr_origin(polyomino,’Inspired by \cite{Golomb65}.’,[]).`

`ctr_arguments( polyomino, [CELLS-
    collection(   index-int,   right-dvar,   left-dvar,   up-dvar,   down-dvar)])).`

`ctr_restrictions( polyomino, [‘CELLS’ˆindex>=1,   ‘CELLS’ˆindex=<size(‘CELLS’),   size(‘CELLS’)>=1,   required(‘CELLS’,[index,right,left,up,down]),   distinct(‘CELLS’,index),   ‘CELLS’ˆright>=0,   ‘CELLS’ˆright=<size(‘CELLS’),   ‘CELLS’ˆleft>=0,   ‘CELLS’ˆleft=<size(‘CELLS’),   ‘CELLS’ˆup>=0,   ‘CELLS’ˆup=<size(‘CELLS’),   ‘CELLS’ˆdown>=0,   ‘CELLS’ˆdown=<size(‘CELLS’)])].)

`ctr_example( polyomino, polyomino(   [[index-1,right-0,left-0,up-2,down-0],   [index-2,right-3,left-0,up-0,down-1],   [index-3,right-0,left-2,up-4,down-0],   [index-4,right-5,left-0,up-0,down-3],   [index-5,right-0,left-4,up-0,down-0]]).`

`ctr_exchangeable( polyomino, [items(‘CELLS’,all),]`
attrs_sync('CELLS',[[index],[right, left], [up], [down]]),
attrs_sync('CELLS',[[index],[right],[left],[up, down]]),
attrs_sync('CELLS',[[index],[up, left, down, right]]).
B.290  power

◊ Meta-Data:

ctr_predefined(power).

ctr_date(power,['20070930']).

ctr_origin(power,\cite{DenmatGotliebDucasse07},[]).

ctr_synonyms(power,[xexpyeqz]).

ctr_arguments(power,['X'-dvar,'N'-dvar,'Y'-dvar]).

ctr_restrictions(power,['X'>=0,'N'>=0,'Y'>=0]).

ctr_example(power,power(2,3,8)).

ctr_typical(power,['X'>1,'N'>1,'Y'>1]).

ctr_eval(power,[reformulation(power_r)]).

ctr_pure_functional_dependency(power,[]).

ctr_functional_dependency(power,3,[1,2]).

\begin{verbatim}

\texttt{power_r(_14329,0,Y) :- !, Y=1.}
\texttt{power_r(X,N,Y) :-
  check_type(dvar_gteq(0),X),
  check_type(dvar_gteq(0),N),
  check_type(dvar_gteq(0),Y),
  fd_min(N,Min),
  fd_max(N,Max),
  Min1 is max(1,Min),
  power1(0,Min1,Max,1,X,Y,Disj),
  call(Disj).}
\texttt{power1(I,_14330,Max,_14332,_14333,_14334,_14335,0) :- I>Max, !.}
\texttt{power1(I,Min,Max,P,X,Y,N,R) :- I<Min,}
\end{verbatim}

\end{verbatim}
\text{!,}
\text{Il is I+1,}
\text{power1(Il,Min,Max,P*X,X,Y,N,R).}

\text{power1(I,Min,Max,P,X,Y,N,P#=Y#/N#=I#/R) :-}
\text{I>=Min,}
\text{I=<Max,}
\text{Il is I+1,}
\text{power1(Il,Min,Max,P*X,X,Y,N,R).}
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.291  precedence

◊ **Meta-Data:**

```prolog
ctr_date(precedence,[\'20111015\']).

ctr_origin(precedence,’Scheduling’,[]).

ctr_arguments(
  precedence,
  [\'TASKS'-collection(origin-dvar,duration-dvar)]).

ctr_restrictions(
  precedence,
  [required(\'TASKS',[origin,duration]),\'TASKS'\^duration>=0]).

ctr_example(
  precedence,
  precedence([[origin-1,duration-3],
                [origin-4,duration-0],
                [origin-5,duration-2],
                [origin-8,duration-1]]).

ctr_typical(precedence,[size(\'TASKS\')>1,\'TASKS'\^duration>=1]).

ctr_exchangeable(
  precedence,
  [vals([\'TASKS'\^duration],int(=0)),>,dontcare,dontcare),
   translate([\'TASKS'\^origin])]).

ctr_graph(
  precedence,
  [\'TASKS\'],
  2,
  [\'PATH'\>collection(tasks1,tasks2)],
   [tasks1\^origin+tasks1\^duration<tasks2\^origin],
   [\'NARC'=size(\'TASKS\')-1],
   []).

ctr_eval(precedence,[reformulation(precedence_r)]).

ctr_contractible(precedence,[],\'TASKS\',any).

precedence_r(TASKS) :-
  length(TASKS,N),
```
N>1,
collection(TASKS, [dvar, dvar_gteq(0)]),
get_attr1(TASKS, ORIGINS),
get_attr2(TASKS, DURATIONS),
gen_precedences(ORIGINS, DURATIONS).

gen_precedences([_23799], [_23801]) :- !.

gen_precedences([O1, O2 | R], [D1, D2 | S]) :-
O1+D1#=<O2,
gen_precedences([O2 | R], [D2 | S]).
B.292  product_ctr

◊ **Meta-Data:**

```
ctr_date(product_ctr,['20030820','20060813','20070902']).
ctr_origin(product_ctr,'Arithmetic constraint.',[]).
ctr_arguments(  
    product_ctr,  
    ['VARIABLES'-collection(var-dvar),'CTR'-atom,'VAR'-dvar]).
ctr_restrictions(  
    product_ctr,  
    [required('VARIABLES',var),  
      in_list('CTR',[=,\=<,>,>=,<=])]).
ctr_example(  
    product_ctr,  
    product_ctr([[var-2],[var-1],[var-4]],=,8)).
ctr_typical(  
    product_ctr,  
    [size('VARIABLES')\>1,  
      size('VARIABLES')\<10,  
      range('VARIABLES'\^\^var)\>1,  
      'VARIABLES'\^\^var\=\=0,  
      in_list('CTR',[=,<,\>=,\>=,\<=]),  
      'VAR'\=\=0]).
ctr_exchangeable(product_ctr,[items('VARIABLES',all)]).
ctr_graph(  
    product_ctr,  
    ['VARIABLES'],  
    1,  
    ['SELF'\=>collection(variables)],  
    ['TRUE'],  
    ['CTR'('PROD'('VARIABLES',var),'VAR')],  
    []).
ctr_eval(product_ctr,[reformulation(product_ctr_r)]).
ctr_pure_functional_dependency(  
    product_ctr,  
    [in_list('CTR',[=])]).
```
ctr_contractible(
    product_ctr,
    \[\text{in\_list}'\text{CTR}',\text{[<,\leq]}',\text{minval}'\text{VARIABLES'}^\text{\textasciitilde}\text{\text{var}}\geq\text{0}],
    \text{VARIABLES},
    \text{any}).

ctr_aggregate(product_ctr,\text{[in\_list}'\text{CTR}',\text{=}'],[\text{union},\text{id},\text{\text{*}}]).

\text{product\_ctr\_r}(\text{VARIABLES},\text{CTR},\text{VAR}) :-
    \text{collection}(\text{VARIABLES},[\text{dvar}]),
    \text{memberchk}(\text{CTR},[\text{=,\leq,<,\geq,\geq,\leq}]),
    \text{check\_type}(\text{dvar},\text{VAR}),
    \text{get\_attr}(\text{VARIABLES},\text{VARS}),
    \text{build\_prod\_var}(\text{VARS},\text{PROD}),
    \text{call\_term\_relop\_value}(\text{PROD},\text{CTR},\text{VAR}).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.293 proper_forest

◊ META-DATA:

ctr_date(proper_forest,[‘20050604’,‘20060813’]).

ctr_origin(
    proper_forest,
    Derived from %c, \cite{BeldiceanuKatrielLorca06}., [tree]).

ctr_arguments(
    proper_forest,
    [’NTREES’-dvar,
     ’NODES’-collection(index-int,neighbour-svar)]).

ctr_restrictions(
    proper_forest,
    [’NTREES’>=0,
     required(’NODES’,[index,neighbour]),
     size(’NODES’)mod 2=0,
     ’NODES’^index>=1,
     ’NODES’^index=<size(’NODES’),
     distinct(’NODES’,index),
     ’NODES’^neighbour>=1,
     ’NODES’^neighbour=<size(’NODES’),
     ’NODES’^neighbour\='NODES’^index]).

ctr_example(
    proper_forest,
    proper_forest(3,
        [[index-1,neighbour-{3,6}],
         [index-2,neighbour-{9}],
         [index-3,neighbour-{1,5,7}],
         [index-4,neighbour-{9}],
         [index-5,neighbour-{3}],
         [index-6,neighbour-{1}],
         [index-7,neighbour-{3}],
         [index-8,neighbour-{10}],
         [index-9,neighbour-{2,4}],
         [index-10,neighbour-{8}]]).

ctr_typical(proper_forest,[’NTREES’>0,size(’NODES’)>1]).

ctr_exchangeable(proper_forest,[items(’NODES’,all)]).
ctr_graph(
    proper_forest,
    ['NODES'],
    2,
    ['CLIQUE' \(\subseteq\)] collection(nodes1, nodes2),
    [nodes2\text{\textasciitilde}index in_set nodes1\text{\textasciitilde}neighbour],
    ['N_VERTEX' = ('NARC' + 2 * 'N_TREES') / 2,
     'N_CC' = 'N_TREES',
     'N_VERTEX' = size('NODES'),
     ['SYMMETRIC']].

ctr_functional_dependency(proper_forest, 1, [2]).
B.294  range_ctr

◊ **Meta-Data:**

\[
\text{ctr\_date(range\_ctr,\[['20030820', '20060813']\]).}
\]

\[
\text{ctr\_origin(range\_ctr,'Arithmetic constraint.',\[]\}).}
\]

\[
\text{ctr\_arguments(}
\quad \text{range\_ctr,}
\quad \text{['VARIABLES'-collection(var-dvar),'CTR'-atom,'R'-dvar]).}
\]

\[
\text{ctr\_restrictions(}
\quad \text{range\_ctr,}
\quad \text{[size('VARIABLES')>0,}
\quad \quad \text{required('VARIABLES',var),}
\quad \quad \text{in_list('CTR',[=,\|=,<,\>=,>,\=<])).}}
\]

\[
\text{ctr\_example(range\_ctr,range\_ctr([[var-1],[var-9],[var-4]],=,9)).}
\]

\[
\text{ctr\_typical(}
\quad \text{range\_ctr,}
\quad \text{[size('VARIABLES')>1,}
\quad \quad \text{range('VARIABLES'\^var)>1,}
\quad \quad \text{in_list('CTR',[=,<,\>=,>,\=<])).}}
\]

\[
\text{ctr\_exchangeable(}
\quad \text{range\_ctr,}
\quad \text{[items('VARIABLES',all),}
\quad \quad \text{vals(['VARIABLES'\^var],int,\|=,all,in),}
\quad \quad \text{translate(['VARIABLES'\^var])).}}
\]

\[
\text{ctr\_graph(}
\quad \text{range\_ctr,}
\quad \text{['VARIABLES'],}
\quad \text{1,}
\quad \text{['SELF']>>collection(variables)],}
\quad \text{['TRUE'],}
\quad \text{['CTR'('RANGE'('VARIABLES',var),'R')],}
\quad \text{[}).}
\]

\[
\text{ctr\_eval(range\_ctr,[reformulation(range\_ctr\_r)]).}
\]

\[
\text{ctr\_pure\_functional\_dependency(range\_ctr,[in\_list('CTR',[=])).}
\]

\[
\text{ctr\_contractible(}
\]
range_ctr,
  [in_list(’CTR’,[<,=])],
  VARIABLES,
  any).

ctr_extensible(  
  range_ctr,
  [in_list(’CTR’,[>=,>])],
  VARIABLES,
  any).

range_ctr_r(VARIABLES,CTR,R) :-
  collection(VARIABLES,[dvar]),
  memberchk(CTR,[=,\=,\<,\>,\>=,\=<]),
  check_type(dvar,R),
  length(VARIABLES,N),
  N>0,
  get_attr1(VARIABLES,VARS),
  minimum(MIN,VARS),
  maximum(MAX,VARS),
  call_term_relop_value(MAX-MIN+1,CTR,R).
B.295  relaxed_sliding_sum

◇ **META-DATA:**

```
ctr_date(
    relaxed_sliding_sum,
    ['20000128','20030820','20060813']).

ctr_origin(relaxed_sliding_sum,'\index{CHIP|indexuse}CHIP',[]).

ctr_arguments(
    relaxed_sliding_sum,
    ['ATLEAST'-int,
     'ATMOST'-int,
     'LOW'-int,
     'UP'-int,
     'SEQ'-int,
     'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    relaxed_sliding_sum,
    ['ATLEAST'>=0,
     'ATMOST'>='ATLEAST',
     'ATMOST'=<size('VARIABLES')-'SEQ'+1,
     'UP'>='LOW',
     'SEQ'>0,
     'SEQ'=<size('VARIABLES'),
     required('VARIABLES',var))).

ctr_example(
    relaxed_sliding_sum,
    relaxed_sliding_sum(
        3,
        4,
        3,
        7,
        4,
        [[var-2],
         [var-4],
         [var-2],
         [var-0],
         [var-0],
         [var-3],
         [var-4]])).

ctr_typical(
```
relaxed_sliding_sum,
[‘SEQ’>1,
 ‘SEQ’<size(‘VARIABLES’),
 range(‘VARIABLES’\^\text{\textasciitilde}var)>1,
 ‘ATLEAST’>0\#\‘ATMOST’<size(‘VARIABLES’)-‘SEQ’+1]).

ctr_exchangeable(
 relaxed_sliding_sum,
 [vals([‘ATLEAST’],int(\geq(0)),>,dontcare,dontcare),
 vals(
   [‘ATMOST’],
   int(\leq(size(‘VARIABLES’)-‘SEQ’+1)),
   <, dontcare, dontcare),
 items(‘VARIABLES’,reverse)]).

ctr_graph(
 relaxed_sliding_sum,
 [‘VARIABLES’],
 SEQ, 
 [‘PATH’\gg\text{\textasciitilde}collection],
 [sum_ctr(collection,\geq,’LOW’),sum_ctr(collection,\leq,’UP’)],
 [‘NARC’\geq‘ATLEAST’,‘NARC’\leq‘ATMOST’],
 []).

ctr_eval(
 relaxed_sliding_sum,
 [reformulation(relaxed_sliding_sum_r)]).

relaxed_sliding_sum_r(\text{\textasciitilde}ATLEAST,\text{\textasciitilde}ATMOST,LOW,UP,SEQ,\text{\textasciitilde}VARIABLES) :-
 integer(\text{\textasciitilde}ATLEAST),
 integer(\text{\textasciitilde}ATMOST),
 integer(LOW),
 integer(UP),
 integer(SEQ),
 collection(\text{\textasciitilde}VARIABLES,[\text{\textasciitilde}dvar]),
 length(\text{\textasciitilde}VARIABLES,N),
 LIMIT is N-SEQ+1,
 ATLEAST\geq0,
 ATMOST\geq\text{\textasciitilde}ATLEAST,
 ATMOST\leq\text{\textasciitilde}LIMIT,
 UP\geq\text{\textasciitilde}LOW,
 SEQ\geq0,
 SEQ<\text{\textasciitilde}N,
 get_attr1(\text{\textasciitilde}VARIABLES,\text{\textasciitilde}VARS),
relaxed_sliding_sum1(VARS, [], LOW, UP, SEQ, SUMB),
call(SUMB#>=ATLEAST),
call(SUMB#=<ATMOST).

relaxed_sliding_sum1([], _20633, _20634, _20635, _20636, 0).

relaxed_sliding_sum1([Last | R], Seq, LOW, UP, SEQ, B+RB) :-
append(Seq, [Last], Sequence),
length(Sequence, L),
(L > SEQ ->
Sequence=[_20696|SeqCur],
build_sum_var(SeqCur, SumVar),
B in 0..1,
call(SumVar#>=LOW#/\SumVar#=<UP#<=>B),
relaxed_sliding_sum1(R, SeqCur, LOW, UP, SEQ, RB)
; L = SEQ ->
build_sum_var(Sequence, SumVar),
B in 0..1,
call(SumVar#>=LOW#/\SumVar#=<UP#<=>B),
relaxed_sliding_sum1(R, Sequence, LOW, UP, SEQ, RB)
; relaxed_sliding_sum1(R, Sequence, LOW, UP, SEQ, RB)
).

APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE
B.296  remainder

◊ Meta-Data:

ctr_predefined(remainder).

ctr_date(remainder, [‘20110612’]).

ctr_origin(remainder, ‘Arithmetic.’, []).

ctr_synonyms(remainder, [modulo, mod]).

ctr_arguments(remainder, ['Q'-dvar,'D'-dvar,'R'-dvar]).

ctr_restrictions(remainder, ['Q'='#>=0', 'D'='#>0', 'R'='#>=0', 'R','#<D']).

ctr_example(remainder, remainder(15,2,1)).

ctr_eval(remainder, [builtin(remainder_b)]).

ctr_pure_functional_dependency(remainder, []).

ctr_functional_dependency(remainder, 3, [1,2]).

remainder_b(Q,D,R) :-
    check_type(dvar,Q),
    check_type(dvar,D),
    check_type(dvar,R),
    Q#>=0,
    D#>0,
    R#>=0,
    R#<D,
    Q mod D#=R.
B.297 roots

◊ **Meta-Data:**

```prolog
ctr_date(roots,['20070620']).

ctr_origin(roots, \cite{BessiereHebrardHnichKiziltanWalsh05IJCAI}, []).

ctr_arguments(roots, ['S'-svar,'T'-svar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(roots, ['S'=<size('VARIABLES'),required('VARIABLES',var)]).

ctr_example(roots, roots([\{2,4,5\},\{2,3,8\},\[[var-1],[var-3],[var-1],[var-2],[var-3]\]]).

ctr_typical(roots, [size('VARIABLES')>1,range('VARIABLES'\^var)>1]).

ctr-derived_collections(roots, [col('SETS'-collection(s-svar,t-svar), \item(s-'S',t-'T'))]).

ctr_graph(roots, ['SETS','VARIABLES'], 2, ['PRODUCT'>>collection(sets,variables)], [variables^key in_set sets^s#<= variables^var in_set sets^t], ['NARC'=size('VARIABLES')], []).
B.298 same

◊ Meta-Data:

ctr_date(same,["20000128","20030820","20040530","20060813"]).

ctr_origin(same,"N.˘Beldiceanu",[]).

ctr_arguments(same,
        ['VARIABLES1'-collection(var-dvar),
         'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(same,
        [size('VARIABLES1')=size('VARIABLES2'),
         required('VARIABLES1',var),
         required('VARIABLES2',var)]).

ctr_example(same,
        same(same(
            [[var-1],[var-9],[var-1],[var-5],[var-2],[var-1]],
             [[var-9],[var-1],[var-1],[var-1],[var-2],[var-5]])).

ctr_typical(same,
        [size('VARIABLES1')>1,
         range('VARIABLES1'\^var)>1,
         range('VARIABLES2'\^var)>1]).

ctr_exchangeable(same,
        [args([[VARIABLES1','VARIABLES2']]),
         items('VARIABLES1',all),
         items('VARIABLES2',all),
         vals(
             ['VARIABLES1'\^var,'VARIABLES2'\^var],
             int,
             =\=,
             all,
             dontcare)]).

ctr_graph(same,
        ['VARIABLES1','VARIABLES2'],
        [\=\=,
         all,
         dontcare]).
2, ['PRODUCT'>>collection(variables1,variables2)],
[variables1^var=variables2^var],
[for_all('CC','NSOURCE'='NSINK'),
 'NSOURCE'=size('VARIABLES1'),
 'NSINK'=size('VARIABLES2')],
[]).

ctr_eval(same,[reformulation(same_r)]).

ctr_aggregate(same,[],[union,union]).

same_r(VARIABLES1,VARIABLES2) :-
collection(VARIABLES1,[dvar]),
collection(VARIABLES2,[dvar]),
length(VARIABLES1,N1),
length(VARIABLES2,N2),
N1=N2,
get_attr1(VARIABLES1,VARS1),
get_attr1(VARIABLES2,VARS2),
same1(VARS1,VARS2).

same1(VARS1,VARS2) :-
length(VARS1,N),
length(PERMUTATION1,N),
domain(PERMUTATION1,1,N),
length(PERMUTATION2,N),
domain(PERMUTATION2,1,N),
length(SVARS,N),
get_minimum(VARS1,MIN1),
get_maximum(VARS1,MAX1),
domain(SVARS,MIN1,MAX1),
sorting(VARS1,PERMUTATION1,SVARS),
sorting(VARS2,PERMUTATION2,SVARS).
B.299  same_and_global_cardinality

◊ **META-DATA:**

ctr_date(same_and_global_cardinality,['20040530','20060813']).

ctr_origin(same_and_global_cardinality, Conjoin %c and %c, [same,global_cardinality]).

ctr_synonyms(same_and_global_cardinality, [sgcc,same_gcc,same_and_gcc,swc,same_with_cardinalities]).

ctr_arguments(same_and_global_cardinality, ['VARIABLES1'-collection(var-dvar), 'VARIABLES2'-collection(var-dvar), 'VALUES'-collection(val-int,noccurrence-dvar)]).

ctr_restrictions(same_and_global_cardinality, [size('VARIABLES1')=size('VARIABLES2'), required('VARIABLES1',var), required('VARIABLES2',var), required('VALUES',val), 'VALUES'~noccurrence>=0, 'VALUES'~noccurrence=<size('VARIABLES1'))].

ctr_example(same_and_global_cardinality, same_and_global_cardinality([ [var-1],[var-9],[var-1],[var-5],[var-2],[var-1] ], [ [var-9],[var-1],[var-1],[var-1],[var-2],[var-5] ], [ [val-1,noccurrence-3], [val-2,noccurrence-1], [val-5,noccurrence-1], [val-7,noccurrence-0], [val-9,noccurrence-1]]).

ctr_typical(same_and_global_cardinality, [size('VARIABLES1')>1, range('VARIABLES1'~var)>1, ...]}.
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

\[
\text{range('VARIABLES2' \cdot \text{var}) > 1,} \\
\text{size('VALUES') > 1,} \\
\text{range('VALUES' \cdot \text{noccurrence}) > 1,} \\
\text{size('VARIABLES1') > size('VALUES'))}.
\]

\[
\text{\textit{ctr\_exchangeable}}(
\text{\hspace{1em}same\_and\_global\_cardinality,} \\
\text{\hspace{1em}[\text{args}([['VARIABLES1','VARIABLES2'], ['VALUES']]),} \\
\text{\hspace{2em}items('VARIABLES1',\text{all}),} \\
\text{\hspace{2em}items('VARIABLES2',\text{all}),} \\
\text{\hspace{2em}items('VALUES',\text{all}),} \\
\text{\hspace{2em}vals(} \\
\text{\hspace{3em}['VARIABLES1' \cdot \text{var}, 'VARIABLES2' \cdot \text{var}],} \\
\text{\hspace{3em}all(\text{notin('VALUES' \cdot \text{val})},} \\
\text{\hspace{3em}=} \\
\text{\hspace{3em}dontcare,} \\
\text{\hspace{3em}dontcare),} \\
\text{\hspace{2em}vals(} \\
\text{\hspace{3em}['VARIABLES1' \cdot \text{var}, 'VARIABLES2' \cdot \text{var}, 'VALUES' \cdot \text{val}],} \\
\text{\hspace{3em}int,} \\
\text{\hspace{3em}=} \text{=} \\
\text{\hspace{3em}all,} \\
\text{\hspace{3em}dontcare})}).
\]

\[
\text{\textit{ctr\_graph}}(
\text{\hspace{1em}same\_and\_global\_cardinality,} \\
\text{\hspace{1em}['VARIABLES1','VARIABLES2'],} \\
\text{\hspace{1em}2,} \\
\text{\hspace{2em}['PRODUCT' >> \text{collection}(variables1,variables2)],} \\
\text{\hspace{2em}variables1' \cdot \text{var}=variables2' \cdot \text{var],} \\
\text{\hspace{2em}[\text{for\_all('CC','NSOURCE'='NSINK'),} \\
\text{\hspace{3em}NSOURCE=size('VARIABLES1'),} \\
\text{\hspace{3em}NSINK=size('VARIABLES2'))}],} \\
\text{[}).}
\]

\[
\text{\textit{ctr\_graph}}(
\text{\hspace{1em}same\_and\_global\_cardinality,} \\
\text{\hspace{1em}['VARIABLES1'],} \\
\text{\hspace{1em}1,} \\
\text{\hspace{2em}foreach('VALUES',[['SELF' >> \text{collection}(variables)]]),} \\
\text{\hspace{2em}variables' \cdot \text{var}=VALUES' \cdot \text{val],} \\
\text{\hspace{2em}['NVERTEX'='VALUES' \cdot \text{noccurrence}],} \\
\text{[}).}
\]

\[
\text{\textit{ctr\_eval}}(
\text{\hspace{1em}same\_and\_global\_cardinality,}
\]
[reformulation(same_and_global_cardinality_r)].

ctr_contractible(same_and_global_cardinality,[],'VALUES',any).

same_and_global_cardinality_r(VARIABLES1,VARIABLES2,VALUES) :-
  eval(same(VARIABLES1,VARIABLES2)),
  eval(global_cardinality(VARIABLES1,VALUES)).
B.300 same_and_global_cardinality_low_up

◊ **Meta-Data:**

```prolog
ctr_date( 
    same_and_global_cardinality_low_up, 
    ['20051104','20060813']).

ctr_origin( 
    same_and_global_cardinality_low_up, 
    Derived from %c and %c, 
    [same,global_cardinality_low_up]).

ctr_arguments( 
    same_and_global_cardinality_low_up, 
    ['VARIABLES1'-collection(var-dvar), 
     'VARIABLES2'-collection(var-dvar), 
     'VALUES'-collection(val-int,omin-int,omax-int)]).

ctr_restrictions( 
    same_and_global_cardinality_low_up, 
    [size('VARIABLES1')=size('VARIABLES2'), 
     required('VARIABLES1',var), 
     required('VARIABLES2',var), 
     required('VALUES',[val,omin,omax]), 
     distinct('VALUES',val), 
     'VALUES'¨omin>=0, 
     'VALUES'¨omax=<size('VARIABLES1'), 
     'VALUES'¨omin=<'VALUES'¨omax]).

ctr_example( 
    same_and_global_cardinality_low_up, 
    same_and_global_cardinality_low_up( 
        [[var-1],[var-9],[var-1],[var-5],[var-2],[var-1]], 
        [[var-9],[var-1],[var-1],[var-1],[var-2],[var-5]], 
        [[val-1,omin-2,omax-3], 
         [val-2,omin-1,omax-1], 
         [val-5,omin-1,omax-1], 
         [val-7,omin-0,omax-2], 
         [val-9,omin-1,omax-1]])).

ctr_typical( 
    same_and_global_cardinality_low_up, 
    [size('VARIABLES1')>1, 
     range('VARIABLES1'¨var)>1, 
     range('VARIABLES2'¨var)>1, 
     
```
size('VALUES') \geq 1,
'VALUES'omin='VALUES'\omin=<=\text{size('VARIABLES1')},
'VALUES'omax='VALUES'\omax=0,
'VALUES'omax='VALUES'\omax=\text{size('VARIABLES1')},
size('VARIABLES1')=\text{size('VALUES')}].

\text{ctr\_exchangeable(}
\text{same\_and\_global\_cardinality\_low\_up,}
[\text{args}([['VARIABLES1','VARIABLES2'],[\text{VALUES}]]),
\text{items('VARIABLES1'),all},
\text{items('VARIABLES2'),all},
\text{vals(}
\text{[','VARIABLES1''var','VARIABLES2''var]},
\text{all(notin('VALUES''val'))),}
\text{=,}
\text{dontcare,}
\text{dontcare)},
\text{items('VALUES'),all},
\text{vals([','VALUES''omin],int(>=(0)),>,dontcare,dontcare)},
\text{vals(}
\text{[','VALUES''omax],}
\text{int(=<(\text{size('VARIABLES1')})),}
\text{<,}
\text{dontcare,}
\text{dontcare),}
\text{vals(}
\text{[','VARIABLES1''var','VARIABLES2''var','VALUES''val]},
\text{int,}
\text{=\=,}
\text{all,}
\text{dontcare})].

\text{ctr\_graph(}
\text{same\_and\_global\_cardinality\_low\_up,}
['VARIABLES1','VARIABLES2'],
2,
['PRODUCT'\=>\text{collection(variables1,variables2)],
[variables1'var=variables2'var],
[\text{for\_all('CC','NSOURCE'='NSINK'),}
\text{NSOURCE}=\text{size('VARIABLES1')},
\text{NSINK}=\text{size('VARIABLES2')}],
[]).

\text{ctr\_graph(}
\text{same\_and\_global\_cardinality\_low\_up,}
['VARIABLES1'],
foreach('VALUES', ['SELF'>>collection(variables)],
[variables^var='VALUES'^val],
['NVERTEX'^'=='VALUES'^omin,'NVERTEX'^'='VALUES'^omax],
[]).

ctr_eval(
    same_and_global_cardinality_low_up,
    [reformulation(same_and_global_cardinality_low_up_r)]).

ctr_contractible(
    same_and_global_cardinality_low_up,
    [],
    VALUES,
    any).

same_and_global_cardinality_low_up_r(
    VARIABLES1,
    VARIABLES2,
    VALUES) :-
    eval(same(VARIABLES1,VARIABLES2)),
    eval(global_cardinality_low_up(VARIABLES1,VALUES)).
B.301  same_intersection

◊ **META-DATA:**

\[
\text{ctr\_date(same\_intersection, ['20040530', '20060814'])}.
\]

\[
\text{ctr\_origin(same\_intersection,)}
\]
\[
\text{Derived from \%c and \%c.,}
\]
\[
\text{[same, common]}.
\]

\[
\text{ctr\_arguments(same\_intersection,)}
\]
\[
\text{['VARIABLES1'\-collection(var-dvar),}
\]
\[
\text{'VARIABLES2'\-collection(var-dvar)].}
\]

\[
\text{ctr\_restrictions(same\_intersection,)}
\]
\[
\text{[required('VARIABLES1', var), required('VARIABLES2', var)].}
\]

\[
\text{ctr\_example(same\_intersection,)}
\]
\[
\text{same\_intersection(}
\]
\[
\text{[[var-1],[var-9],[var-1],[var-5],[var-2],[var-1]],}
\]
\[
\text{[[var-9],}
\]
\[
\text{[var-1],}
\]
\[
\text{[var-1],}
\]
\[
\text{[var-3],}
\]
\[
\text{[var-5],}
\]
\[
\text{[var-8]]].}
\]

\[
\text{ctr\_typical(same\_intersection,)}
\]
\[
\text{[size('VARIABLES1')>1,}
\]
\[
\text{range('VARIABLES1'\^var)>1,}
\]
\[
\text{size('VARIABLES2')>1,}
\]
\[
\text{range('VARIABLES2'\^var)>1]).}
\]

\[
\text{ctr\_exchangeable(same\_intersection,)}
\]
\[
\text{[args([['VARIABLES1', 'VARIABLES2']]),}
\]
\[
\text{items('VARIABLES1', all),}
\]
\[
\text{items('VARIABLES2', all),}
\]
\[
\text{vals(
\]
\[
\text{['VARIABLES1'\^var,'VARIABLES2'\^var],}
\]
\[
\text{[['VARIABLES1', 'VARIABLES2']}])].}
\]
\text{int,}
\text{\textbackslash =,}
\text{all,}
\text{dontcare}).}

c_{\text{tr\_graph}}(\text{same\_intersection,}
\text{[}'VARIABLES1','VARIABLES2'\text{],}
\text{2,}
\text{[}'PRODUCT'>>\text{collection(v1,v2)],}
\text{[v1\_\text{\textbackslash var}=v2\_\text{\textbackslash var}],}
\text{[\text{for\_all}('CC','NSOURCE'='NSINK')],}
\text{[]}).}

c_{\text{tr\_eval}}(\text{same\_intersection,}
\text{[reformulation(same\_intersection\_r)]}).

\text{same\_intersection\_r(VARIABLES1,VARIABLES2) :-}
\text{\text{collection(v1,[dvar]),}
\text{collection(v2,[dvar]),}
\text{length(v1,N1),}
\text{length(v2,N2),}
\text{\{ N1=0 ->
\text{true
\text{; N2=0 ->
\text{true
\text{; get\_attr1(v1,v11),}
\text{get\_attr1(v2,v21),}
\text{get\_minimum(v1,v11),}
\text{get\_minimum(v2,v21),}
\text{get\_maximum(v1,v11),}
\text{get\_maximum(v2,v21),}
\text{\text{MIN is min(v11,v21),
\text{MAX is max(v11,v21),
\text{complete\_card(MIN,MAX,N1,[],[],VN1),
\text{complete\_card(MIN,MAX,N2,[],[],VN2),
\text{global\_cardinality(v1,VN1),
\text{global\_cardinality(v2,VN2),
\text{same\_intersection1(VN1,VN2)
\text{).}
\text{same\_intersection1([],[]).}
\text{same\_intersection1([v-O1|R],[v-O2|S]) :-
\text{O1#>0#/\02#>0#=>O2#=O1,}
same_intersection1(R,S).
B.302  same_interval

◇ Meta-Data:

ctr_date(same_interval, [‘20030820’, ‘20060814’]).

ctr_origin(same_interval, ‘Derived from %c.’, [same]).

ctr_arguments(
    same_interval,
    [‘VARIABLES1’-collection(var-dvar),
     ‘VARIABLES2’-collection(var-dvar),
     ‘SIZE_INTERVAL’-int]).

ctr_restrictions(
    same_interval,
    [size(‘VARIABLES1’) = size(‘VARIABLES2’),
     required(‘VARIABLES1’, var),
     required(‘VARIABLES2’, var),
     ‘SIZE_INTERVAL’ > 0]).

ctr_example(
    same_interval,
    same_interval(
        same_interval{
            [[var-1], [var-7], [var-6], [var-0], [var-1], [var-7]],
            [[var-8], [var-8], [var-8], [var-0], [var-1], [var-2]],
            3}).

ctr_typical(
    same_interval,
    [size(‘VARIABLES1’)>1,
     range(‘VARIABLES1’ ^ var)>1,
     range(‘VARIABLES2’ ^ var)>1,
     ‘SIZE_INTERVAL’>1,
     ‘SIZE_INTERVAL’<range(‘VARIABLES1’ ^ var),
     ‘SIZE_INTERVAL’<range(‘VARIABLES2’ ^ var)]).

dontcare,
dontcare).

ctr_graph(
  same_interval,
  [‘VARIABLES1’,'VARIABLES2'],
  2,
  ['PRODUCT'>>collection(variables1,variables2)],
  [variables1\$var/'SIZE_INTERVAL'=variables2\$var/'SIZE_INTERVAL'],
  [for_all('CC', 'NSOURCE'='NSINK'),
   'NSOURCE'=size('VARIABLES1'),
   'NSINK'=size('VARIABLES2')],
  []).

ctr_eval(same_interval,[reformulation(same_interval_r)]).

ctr_aggregate(same_interval,[],[union,union,id]).

same_interval_r(VARIABLES1,VARIABLES2,SIZE_INTERVAL) :-
  collection(VARIABLES1,[dvar]),
  collection(VARIABLES2,[dvar]),
  length(VARIABLES1,N1),
  length(VARIABLES2,N2),
  N1=N2,
  integer(SIZE_INTERVAL),
  SIZE_INTERVAL>0,
  get_attr1(VARIABLES1,VARS1),
  get_attr1(VARIABLES2,VARS2),
  gen_quotient(VARS1,SIZE_INTERVAL,QUOTVARS1),
  gen_quotient(VARS2,SIZE_INTERVAL,QUOTVARS2),
  same1(QUOTVARS1,QUOTVARS2).
B.303 same_modulo

◊ **Meta-Data:**

```prolog
ctr_date(same_modulo,['20030820','20060814']).

ctr_origin(same_modulo,'Derived from %c.',[same]).

ctr_arguments(same_modulo,
  [{'VARIABLES1'}-collection(var-dvar),
   'VARIABLES2'–collection(var-dvar),
   'M'–int}).

ctr_restrictions(same_modulo,
  [size('VARIABLES1')=size('VARIABLES2'),
   required('VARIABLES1',var),
   required('VARIABLES2',var),
   'M'>0]).

ctr_example(same_modulo,
  same_modulo([[(var-1),[var-9],[var-1],[var-5],[var-2],[var-1]],
                [[var-6],[var-4],[var-1],[var-1],[var-5],[var-5]],
                3]).

ctr_typical(same_modulo,
  [size('VARIABLES1')>1,
   range('VARIABLES1'~var)>1,
   range('VARIABLES2'~var)>1,
   'M'>1,
   'M'<maxval('VARIABLES1'~var),
   'M'<maxval('VARIABLES2'~var)]).

ctr_exchangeable(same_modulo,
  [args([[VARIABLES1'],[VARIABLES2'],['M']]),
   items('VARIABLES1',all),
   items('VARIABLES2',all),
   vals(['VARIABLES'~var],mod('M'),=,dontcare,dontcare)]).

ctr_graph(same_modulo,
  ..).
```
['VARIABLES1', 'VARIABLES2'],
2,
['PRODUCT'>>collection(variables1,variables2)],
[variables1\mod 'M'=variables2\mod 'M'],
[for_all('CC', 'NSOURCE'='NSINK'),
 'NSOURCE'=size('VARIABLES1'),
 'NSINK'=size('VARIABLES2')],
[]).

ctr_eval(same_modulo,[reformulation(same_modulo_r)]).

ctr_aggregate(same_modulo, [], [union, union, id]).

same_modulo_r(VARIABLES1, VARIABLES2, M) :-
collection(VARIABLES1, [dvar]),
collection(VARIABLES2, [dvar]),
length(VARIABLES1, N1),
length(VARIABLES2, N2),
N1=N2,
integer(M),
M>0,
get_attr1(VARIABLES1, VARS1),
get_attr1(VARIABLES2, VARS2),
gen_remainder(VARS1, M, REMVARS1),
gen_remainder(VARS2, M, REMVARS2),
samel(REMVARS1, REMVARS2).
B.304  same_partition

◊ **Meta-Data:**

ctr_date(same_partition, ['20030820', '20060814']).

ctr_origin(same_partition, 'Derived from %c.', [same]).

ctr_types(same_partition, ['VALUES'-collection(val-int)]).

ctr_arguments(same_partition, ['VARIABLES1'-collection(var-dvar), 'VARIABLES2'-collection(var-dvar), 'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(same_partition, [size('VALUES')>=1, required('VALUES', val), distinct('VALUES', val), size('VARIABLES1')=size('VARIABLES2'), required('VARIABLES1', var), required('VARIABLES2', var), required('PARTITIONS', p), size('PARTITIONS')>=2]).

ctr_example(same_partition, same_partition([[var-1], [var-2], [var-6], [var-3], [var-1], [var-2]], [[var-6], [var-6], [var-2], [var-3], [var-1], [var-3]], [[p-[[val-1], [val-3]], [p-[[val-4]]], [p-[[val-2], [val-6]]]]).

ctr_typical(same_partition, [size('VARIABLES1')>1, range('VARIABLES1'`var)>1, range('VARIABLES2'`var)>1, size('VARIABLES1')>size('PARTITIONS'), size('VARIABLES2')>size('PARTITIONS')]).

ctr_exchangeable(same_partition,
[args([[‘VARIABLES1’,‘VARIABLES2’],[‘PARTITIONS’]]),
  items(‘VARIABLES1’,all),
  items(‘VARIABLES2’,all),
  items(‘PARTITIONS’,all),
  items(‘PARTITIONS’^p,all),
  vals(
    [‘VARIABLES’^var],
    part(‘PARTITIONS’),
    =, 
    dontcare, 
    dontcare))).

ctr_graph(
  same_partition,
  [‘VARIABLES1’,‘VARIABLES2’],
  2,
  [‘PRODUCT’>>collection(variables1,variables2)],
  [in_same_partition(
    variables1^var, 
    variables2^var, 
    PARTITIONS)],
  [for_all(‘CC’,‘NSOURCE’=’NSINK’),
   ‘NSOURCE’=size(‘VARIABLES1’),
   ‘NSINK’=size(‘VARIABLES2’)],
  []).

ctr_eval(same_partition,[reformulation(same_partition_r)]).

ctr_aggregate(same_partition,[],[union,union,id]).

same_partition_r(VARIABLES1,VARIABLES2,PARTITIONS) :-
  collection(VARIABLES1,[dvar]),
  collection(VARIABLES2,[dvar]),
  length(VARIABLES1,N1),
  length(VARIABLES2,N2),
  N1=N2,
  get_attr1(VARIABLES1,VARS1),
  get_attr1(VARIABLES2,VARS2),
  get_col_attr1(PARTITIONS,1,PVALS),
  flattern(PVALS,VALS),
  all_different(VALS),
  length(PARTITIONS,P),
  P>1,
  length(PVALS,LPVALS),
  LPVALS1 is LPVALS+1,
  get_partition_var(VARS1,PVALS,PVARS1,LPVALS1,0),
get_partition_var(VARS2,PVALS,PVARS2,LPVALS1,0),
same1(PVARS1,PVARS2).
B.305 same_sign

◊ Meta-Data:

ctr_predefined(same_sign).

ctr_date(same_sign,['20100821']).

ctr_origin(same_sign,'Arithmetic.',[]).

ctr_arguments(same_sign,['VAR1'-dvar,'VAR2'-dvar]).

ctr_restrictions(same_sign,[]).

ctr_example(same_sign,same_sign(7,1)).

ctr_typical(same_sign,['VAR1'='\=0','VAR2'='\=0]).

ctr_exchangeable(same_sign,[args([['VAR1','VAR2']])]).

ctr_eval(same_sign,[builtin(same_sign_b)]).

same_sign_b(VAR1,VAR2) :-
    check_type(dvar,VAR1),
    check_type(dvar,VAR2),
    VAR1#\>=0\/
    VAR2#\>=0\/
    VAR1#\=<0\/
    VAR2#\=<0.
B.306 scalar_product

◊ Meta-Data:

ctr_predefined(scalar_product).

ctr_date(scalar_product, ['20090415']).

ctr_origin(scalar_product, 'Arithmetic constraint.', []).

ctr_synonyms(scalar_product, [equation, linear, sum_weight, weightedSum]).

ctr_arguments(scalar_product, ['LINEARTERM'-collection(coeff-int, var-dvar), 'CTR'-atom, 'VAL'-dvar]).

ctr_restrictions(scalar_product, [required('LINEARTERM', [coeff, var]), in_list('CTR', [=, !=, <, >, >=, <=])]).

ctr_example(scalar_product, scalar_product([[[coeff-1, var-1], [coeff-3, var-1], [coeff-1, var-4]], =, 8]).

ctr_typical(scalar_product, [size('LINEARTERM') > 1, range('LINEARTERM' ^ coeff) > 1, range('LINEARTERM' ^ var) > 1, in_list('CTR', [=, <, >, =, <=])]).

ctr_exchangeable(scalar_product, [items('LINEARTERM', all), attrs('LINEARTERM', [[coeff, var]])]).

ctr_eval(scalar_product, [builtin(scalar_product_b)]).
ctr_pure_functional_dependency(
    scalar_product,
    [in_list('CTR',[=])]).

ctr_contractible(
    scalar_product,
    [in_list('CTR',[<=,=<]),
     minval('LINEARTERM'`coeff)>=0,
     minval('LINEARTERM'`var)>=0],
    LINEARTERM,
    any).

ctr_extensible(
    scalar_product,
    [in_list('CTR',[>=,>]),
     minval('LINEARTERM'`coeff)>=0,
     minval('LINEARTERM'`var)>=0],
    LINEARTERM,
    any).

ctr_aggregate(scalar_product,[],[union,id,+]).

scalar_product_b(LINEARTERM,=,VAR) :- !,
    collection(LINEARTERM,[int,dvar]),
    check_type(dvar,VAR),
    get_attr1(LINEARTERM,COEFFS),
    get_attr2(LINEARTERM,VARS),
    scalar_product(COEFFS,VARS,#=,VAR).

scalar_product_b(LINEARTERM,\=,VAR) :- !,
    collection(LINEARTERM,[int,dvar]),
    check_type(dvar,VAR),
    get_attr1(LINEARTERM,COEFFS),
    get_attr2(LINEARTERM,VARS),
    scalar_product(COEFFS,VARS,#\=,VAR).

scalar_product_b(LINEARTERM,\<,VAR) :- !,
    collection(LINEARTERM,[int,dvar]),
    check_type(dvar,VAR),
    get_attr1(LINEARTERM,COEFFS),
    get_attr2(LINEARTERM,VARS),
    scalar_product(COEFFS,VARS,#\<,VAR).
scalar_product_b(LINEARTERM,>=,VAR) :-
  !,
  collection(LINEARTERM,[int,dvar]),
  check_type(dvar,VAR),
  get_attr1(LINEARTERM,COEFFS),
  get_attr2(LINEARTERM,VARS),
  scalar_product(COEFFS,VARS,#>=,VAR).

scalar_product_b(LINEARTERM,>,VAR) :-
  !,
  collection(LINEARTERM,[int,dvar]),
  check_type(dvar,VAR),
  get_attr1(LINEARTERM,COEFFS),
  get_attr2(LINEARTERM,VARS),
  scalar_product(COEFFS,VARS,#>,VAR).

scalar_product_b(LINEARTERM,=<,VAR) :-
  collection(LINEARTERM,[int,dvar]),
  check_type(dvar,VAR),
  get_attr1(LINEARTERM,COEFFS),
  get_attr2(LINEARTERM,VARS),
  scalar_product(COEFFS,VARS,#=<,VAR).
B.307 sequence_folding

◊ Meta-Data:

ctr_date(sequence_folding, ['20030820', '20040530', '20060814']).

ctr_origin(sequence_folding, 'J. Pearson', []).

ctr_arguments(
    sequence_folding,
    ['LETTERS'-collection(index-int, next-dvar)]).

ctr_restrictions(
    sequence_folding,
    [size('LETTERS')>=1, 
     required('LETTERS', [index, next]), 
     'LETTERS'~index>=1, 
     'LETTERS'~index=<size('LETTERS'), 
     increasing_seq('LETTERS', index), 
     'LETTERS'~next>=1, 
     'LETTERS'~next=<size('LETTERS')]).

ctr_example(
    sequence_folding,
    sequence_folding(
        [[index-1, next-1], 
         [index-2, next-8], 
         [index-3, next-3], 
         [index-4, next-5], 
         [index-5, next-5], 
         [index-6, next-7], 
         [index-7, next-7], 
         [index-8, next-8], 
         [index-9, next-9]])).

ctr_typical(
    sequence_folding,
    [size('LETTERS')>2, range('LETTERS'~next)>1]).

ctr_graph(
    sequence_folding,
    ['LETTERS'],
    1,
    ['SELF']>>collection(letters),
    [letters~next>=letters~index],
    ['NARC'=size('LETTERS')],
    )
ctr_graph(
    sequence_folding,
    ['LETTERS'],
    2,
    ['CLIQUE'(<)\ collection(letters1, letters2)],
    letters2\next=\next<\next,\n    ['NARC'=\size('LETTERS')*\size('LETTERS')-1)/2],
    []).

ctr_eval(sequence_folding, [automaton(sequence_folding_a)]).

sequence_folding_a(FLAG, LETTERS) :-
    length(LETTERS, N),
    N>=1,
    collection(LETTERS, [int(1,N), dvar(1,N)]),
    collection_increasing_seq(LETTERS, [1]),
    sequence_folding_signature(LETTERS, SIGNATURE),
    AUTOMATON =
        automaton(  
            SIGNATURE,  
            _46743,  
            SIGNATURE,  
            [source(s), sink(s)],  
            [arc(s,0,s), arc(s,1,s)],
            [],
            [],
            []),
    automaton_bool(FLAG, [0,1,2], AUTOMATON).

sequence_folding_signature([], []).

sequence_folding_signature([_45054], []) :- !.

sequence_folding_signature([L1, L2 | R], S) :-
    sequence_folding_signature([L2 | R], L1, S1),
    sequence_folding_signature([L2 | R], S2),
    append(S1, S2, S).

sequence_folding_signature([], _45050, []).

sequence_folding_signature([L2 | R], L1, [S | Ss]) :-
    L1=[index-INDEX1, next-NEXT1],
    ...
L2=[index-INDEX2,next-NEXT2],
INDEX1#=<NEXT1#/INDEX2#=<NEXT2#/\NEXT1#=<INDEX2#><=>
S#=0,
INDEX1#=<NEXT1#/INDEX2#=<NEXT2#/\NEXT1#=<INDEX2#/
NEXT2#=<NEXT1#><=>
S#=1,
sequence_folding_signature(R,L1,Ss).
B.308 set_value_precede

◊ Meta-Data:

ctr_predefined(set_value_precede).

ctr_date(set_value_precede, ['20041003']).

ctr_origin(set_value_precede, '\cite{YatChiuLawJimmyLee04}', []).

ctr_arguments(
    set_value_precede,
    ['S'-int,'T'-int,'VARIABLES'-collection(var-svar)]).

ctr_restrictions(
    set_value_precede,
    ['S'='T', required('VARIABLES', var)]).

ctr_example(
    set_value_precede,
    [set_value_precede(2, 1, [[var-{0,2}], [var-{0,1}], [var-{}], [var-{1}]]),
     set_value_precede(0, 1, [[var-{0,2}], [var-{0,1}], [var-{}], [var-{1}]]),
     set_value_precede(0, 2, [[var-{0,2}], [var-{0,1}], [var-{}], [var-{1}]]),
     set_value_precede(0, 4, [[var-{0,2}], [var-{0,1}], [var-{}], [var-{1}]]))].

ctr_typical(set_value_precede, ['S'<'T', size('VARIABLES')>1]).

ctr_contractible(set_value_precede, [], 'VARIABLES', suffix).
B.309  shift

◊ **META-DATA:**

```prolog
CTR_DATE(shift, ['20030820', '20060814', '20090531']).
CTR_ORIGIN(shift, 'N.˘Beldiceanu', []).
CTR_ARGUMENTS(
    shift,
    ['MIN_BREAK'-int,
     'MAX_RANGE'-int,
     'TASKS'-collection(origin-dvar, end-dvar)]).
CTR_RESTRICTIONS(
    shift,
    ['MIN_BREAK'>0,
     'MAX_RANGE'>0,
     required('TASKS', [origin, end]),
     'TASKS'起源<TASKS'末尾]).
CTR_EXAMPLE(
    shift,
    shift(6, 8,
           [[origin-17, end-20],
            [origin-7, end-10],
            [origin-2, end-4],
            [origin-21, end-22],
            [origin-5, end-6]])).
CTR_TYPICAL(
    shift,
    ['MIN_BREAK'>1,
     'MAX_RANGE'>1,
     'MIN_BREAK'<MAX_RANGE',
     size('TASKS')>2)).
CTR.Exchangeable(
    shift,
    [items('TASKS', all), translate(['TASKS'起源])]).
CTR_GRAPH(
    shift,
    ['TASKS'],
    ...)```
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

1,
[‘SELF’>>collection(tasks)],
[tasks^end>=tasks^origin,
  tasks^end-tasks^origin=<’MAX_RANGE’],
[‘NARC’=size(‘TASKS’)],
[]).

ctr_graph(
  shift,
  [‘TASKS’],
  2,
  [‘CLIQUE’>>collection(tasks1,tasks2)],
  [tasks2^origin>=tasks1^end#/\
    tasks2^origin-tasks1^end=<’MIN_BREAK’#/\
    tasks1^origin>=tasks2^end#/\
    tasks1^origin-tasks2^end=<’MIN_BREAK’#/\
    tasks2^origin<tasks1^end#/\tasks1^origin<tasks2^end],
  [],
  [],
  [CC>>
    col(‘VARIABLES’-collection(var-dvar),
      [item(var-‘TASKS’^origin),item(var-‘TASKS’^end)]),
    [range_ctr(variables,=<,’MAX_RANGE’)])].

ctr_eval(shift,[reformulation(shift_r)]).

shift_r(MIN_BREAK,MAX_RANGE,TASKS) :-
  integer(MIN_BREAK),
  MIN_BREAK>0,
  integer(MAX_RANGE),
  MAX_RANGE>0,
  collection(TASKS,[dvar,dvar]),
  get_attr1(TASKS,ORIGINS),
  get_attr2(TASKS,ENDS),
  get_minimum(ORIGINS,MINO),
  get_maximum(ORIGINS,MNO),
  get_minimum(ENDS,MINE),
  get_maximum(ENDS,MAXE),
  shift1{
    ORIGINS,
    ENDS,
    ORIGINS,
    ENDS,
    MINO,
    MAXO,
shift1([], [], _36999, _36945, _36991, _37037, _37083, _37129, _37175, _37221).

shift1([O|RO], [E|RE], ORIGINS, ENDS, MINO, MAXO, MINE, MAXE, MIN_BREAK, MAX_RANGE) :-
    shift2(ORIGINS, ENDS, O, E, MIN_BREAK, MAX_RANGE, ORIBOOLS, ENDBOOLS),
    MIN in MINO..MAXO,
    MAX in MINE..MAXE,
    eval(open_minimum(MIN,ORIBOOLS)),
    eval(open_maximum(MAX,ENDBOOLS)),
    MAX-MIN#=<MAX_RANGE,
    shift1(RO, RE, ORIGINS,
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

shift2([],[],_36566,_36567,_36568,_36569,[],[]).

shift2([Oj|RO],[Ej|RE],Oi,Ei,MIN_BREAK,MAX_RANGE,[[var-Oj,bool-Bij]|ROB],
        [[var-Ej,bool-Bij]|REB]) :-
        Oi#<Ei,
        Bij#<=>
        Oj#>=Ei#/
        Oi#>=Ej#/
        Oj#<Ei#/
        shift2(RO,RE,Oi,Ei,MIN_BREAK,MAX_RANGE,ROB,REB).
B.310  **sign_of**

◊ **Meta-Data:**

```prolog
ctr_predefined(sign_of).
ctr_date(sign_of, ['20110612']).
ctr_origin(sign_of, 'Arithmetic.', []).
ctr_usual_name(sign_of, sign).
ctr_arguments(sign_of, ['S'-dvar, 'X'-dvar]).
ctr_restrictions(sign_of, ['S' >= -1, 'S' =< 1]).
ctr_example(sign_of, [sign_of(-1,-8), sign_of(0,0), sign_of(1,8)]).
ctr_typical(sign_of, ['S' =\= 0, 'X' =\= 0]).
ctr_eval(sign_of, [builtin(sign_of_b)]).
ctr_pure_functional_dependency(sign_of, []).
ctr_functional_dependency(sign_of, 1, [2]).
```

```prolog
sign_of_b(S,X) :-
    check_type(dvar,S),
    check_type(dvar,X),
    S#>= -1,
    S#=<1,
    X#<0#/
    S# = -1#
    /
    S#=0#
    /
    X#>0#
    /
    S#=1.
```
B.311  \texttt{size\_max\_seq\_alldifferent}

\textbf{Meta-Data:}

\begin{verbatim}
ctr_date(size_max_seq_alldifferent,['20030820','20060814']).

ctr_origin(size_max_seq_alldifferent,'N.˘Beldiceanu',[]).

ctr_synonyms(
  size_max_seq_alldifferent,
  [size_maximal_sequence_alldiff,
   size_maximal_sequence_alldistinct,
   size_maximal_sequence_alldifferent]).

ctr_arguments(
  size_max_seq_alldifferent,
  ['SIZE'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
  size_max_seq_alldifferent,
  ['SIZE']>=0,
  'SIZE'=<size('VARIABLES'),
  required('VARIABLES',var))).

ctr_example(
  size_max_seq_alldifferent,
  size_max_seq_alldifferent(4,
    [[var-2],
     [var-2],
     [var-4],
     [var-5],
     [var-2],
     [var-7],
     [var-4]]).

ctr_typical(
  size_max_seq_alldifferent,
  ['SIZE']>=2,
  'SIZE'=<size('VARIABLES'),
      range('VARIABLES'¯var)>1]).

ctr_exchangeable(
  size_max_seq_alldifferent,
  [translate(['VARIABLES'¯var])]).
\end{verbatim}
ctr_graph(
  size_max_seq_alldifferent,
  ['VARIABLES'],
  '*',
  ['PATH_N'>>collection],
  [alldifferent(collection)],
  ['NARC'='SIZE'],
  []).

ctr_eval(
  size_max_seq_alldifferent,
  [reformulation(size_max_seq_alldifferent_r)]).

ctr_pure_functional_dependency(size_max_seq_alldifferent,[]).

ctr_functional_dependency(size_max_seq_alldifferent,1,[2]).

size_max_seq_alldifferent_r(SIZE,VARIABLES) :-
  length(VARIABLES,N),
  check_type(dvar(0,N),SIZE),
  collection(VARIABLES,[dvar]),
  size_max_seq_alldifferent1(VARIABLES,N,SIZES),
  eval(maximum(SIZE,SIZES)).

size_max_seq_alldifferent1([],_21684,[]).

size_max_seq_alldifferent1([AV|R],N,[[var-SIZE]|S]) :-
  SIZE in 0..N,
  eval(size_max_starting_seq_alldifferent(SIZE,[AV|R])),
  N1 is N-1,
  size_max_seq_alldifferent1(R,N1,S).
B.312 size_max_starting_seq_alldifferent

◊ Meta-Data:

ctr_date(size_max_starting_seq_alldifferent, ['20030820','20060814','20090524']).

ctr_origin(size_max_starting_seq_alldifferent, Inspired by %c., [size_max_seq_alldifferent]).

ctr_synonyms(size_max_starting_seq_alldifferent, [size_maximal_starting_sequence_alldiff, size_maximal_starting_sequence_alldistinct, size_maximal_starting_sequence_alldifferent]).

ctr_arguments(size_max_starting_seq_alldifferent, ['SIZE'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(size_max_starting_seq_alldifferent, ['SIZE'>=0, 'SIZE'=<size('VARIABLES'), required('VARIABLES',var)]).

ctr_example(size_max_starting_seq_alldifferent, size_max_starting_seq_alldifferent(4, [[var-9], [var-2], [var-4], [var-5], [var-2], [var-7], [var-4]])).

ctr_typical(size_max_starting_seq_alldifferent, ['SIZE']>=2, 'SIZE'<size('VARIABLES'), range('VARIABLES'`var)>1]).
ctr_exchangeable(
    size_max_starting_seq_alldifferent,
    [translate(['VARIABLES'='VAR'])]).

ctr_graph(
    size_max_starting_seq_alldifferent,
    ['VARIABLES'],
    ['PATH_1'>>collection],
    [alldifferent(collection)],
    ['NARC'='SIZE'],
    []).

ctr_eval(
    size_max_starting_seq_alldifferent,
    [reformulation(size_max_starting_seq_alldifferent_r)]).

ctr_pure_functional_dependency(
    size_max_starting_seq_alldifferent,
    []).

ctr_functional_dependency(
    size_max_starting_seq_alldifferent,
    1,
    [2]).

size_max_starting_seq_alldifferent_r(SIZE,VARIABLES) :-
    length(VARIABLES,N),
    check_type(dvar(0,N),SIZE),
    collection(VARIABLES,[dvar]),
    get_attr1(VARIABLES,VARS),
    size_max_starting_seq_alldifferent1(VARS,[],1,SUMB),
    call(SIZE#=SUMB).

size_max_starting_seq_alldifferent1([],_23436,_23437,0).

size_max_starting_seq_alldifferent1([VAR|RVARS],L,BPREV,B+SUM) :-
    size_max_starting_seq_alldifferent1(L,VAR,BPREV,CONJ),
    call(B#<=CONJ),
    size_max_starting_seq_alldifferent1( 
        RVARS, 
        [VAR|L],
        B, 
        SUM).
size_max_starting_seq_alldifferent2([],_23436,BPREV,BPREV).

size_max_starting_seq_alldifferent2([VAR2|RVARS], VAR1, BPREV, VAR1#=VAR2#/\R) :-
    size_max_starting_seq_alldifferent2(RVARS,VAR1,BPREV,R).
B.313 sliding_card_skip0

◊ META-DATA:

ctr_date(
  sliding_card_skip0,
  [˜'20000128', '20030820', '20040530', '20060815']).

ctr_origin(sliding_card_skip0, 'N. Beldiceanu', []).

ctr_arguments(
  sliding_card_skip0,
  ['ATLEAST'-int,
   'ATMOST'-int,
   'VARIABLES'-collection(var-dvar),
   'VALUES'-collection(val-int)]).

ctr_restrictions(
  sliding_card_skip0,
  ['ATLEAST'>=0,
   'ATLEAST'=<size('VARIABLES'),
   'ATMOST'>=0,
   'ATMOST'=<size('VARIABLES'),
   'ATMOST'>='ATLEAST',
   required('VARIABLES', var),
   required('VALUES', val),
   distinct('VALUES', val),
   'VALUES'\val\=\=0]).

ctr_example(
  sliding_card_skip0,
  sliding_card_skip0(2, 3,
  [[var-0],
   [var-7],
   [var-2],
   [var-9],
   [var-0],
   [var-0],
   [var-9],
   [var-4],
   [var-9]],
  [[val-7],[val-9]])).

ctr_typical(}
sliding_card_skip0, 
[size('VARIABLES')]>1, 
size('VALUES')>0, 
size('VARIABLES')>size('VALUES'), 
atleast(1,'VARIABLES',0), 
'ATLEAST'>`0\/'ATMOST'<size('VARIABLES')]).

ctr_exchangeable(
  sliding_card_skip0,
  [vals(['ATLEAST'],int(>=0),>,dontcare,dontcare),
   vals([['ATMOST'],
       int(=<size('VARIABLES'))),
       <,
       dontcare,
       dontcare),
   items('VARIABLES',reverse),
   vals([['VARIABLES'\var],
       comp_diff('VALUES'\val,\=0),
       =,
       dontcare,
       dontcare])].

ctr_graph(
  sliding_card_skip0,
  ['VARIABLES'],
  2,
  ['PATH'>>collection(variables1,variables2),
   'LOOP'>>collection(variables1,variables2)],
  [variables1\var=\=0,variables2\var=\=0],
  [],
  [],
  ['CC'>>[variables]],
  [among_low_up('ATLEAST','ATMOST',variables,'VALUES')]).

ctr_eval(sliding_card_skip0,[automaton(sliding_card_skip0_a)]).

sliding_card_skip0_a(FLAG,ATLEAST,ATMOST,VARIABLES,VALUES) :-
  collection(VARIABLES,[dvar]),
  length(VARIABLES,N),
  check_type(int(0,N),ATLEAST),
  check_type(int(0,N),ATMOST),
  ATMOST>=ATLEAST,
  collection(VALUES,[int_diff(0)]),
  get_attr1(VALUES,LIST_VALUES),
all_different(LIST_VALUES),
list_to_fdset(LIST_VALUES,SET_OF_VALUES),
sliding_card_skip0_signature(
  VARIABLES,
  SIGNATURE,
  SET_OF_VALUES),
automaton(
  SIGNATURE,
  _41100,
  SIGNATURE,
  [source(s),sink(i),sink(s)],
  [arc(s,0,s),
   arc(s,1,i,[0,L,U]),
   arc(s,2,i,[1,L,U]),
   arc(i,0,s,[C,min(L,C),max(U,C)]),
   arc(i,1,i),
   arc(i,2,i,[C+1,L,U])],
  [C,L,U],
  [ATLEAST,ATLEAST,ATMOST],
  [C1,L1,U1]),
min(C1,L1)#>=ATLEAST#
\max(C1,U1)#=<ATMOST#<=>FLAG.

sliding_card_skip0_signature([],[],_38290).

sliding_card_skip0_signature(
  [[var-VAR]|VARs],
  [S|Ss],
  SET_OF_VALUES) :-
  VAR#=0#<=>NZ,
  VAR in_set SET_OF_VALUES#<=>In,
  S in 0..2,
  S#max(2*NZ+In-1,0),
  sliding_card_skip0_signature(VARs,Ss,SET_OF_VALUES).
B.314  sliding_distribution

◊ Meta-Data:

$$\text{ctr\_date}(\text{sliding\_distribution},\text{\['20031008', '20060815', '20090524']\}).$$

$$\text{ctr\_origin}(\text{sliding\_distribution},'\text{\cite{ReginPuget97}},[],[]).$$

$$\text{ctr\_arguments}(\text{sliding\_distribution},\text{\['SEQ\'-int, 'VARIABLES'\'-collection(var-dvar), 'VALUES'\'-collection(val-int,omin-int,omax-int)\}]).$$

$$\text{ctr\_restrictions}(\text{sliding\_distribution},\text{\['SEQ'>0, 'SEQ'=<size('VARIABLES'), \text{required('VARIABLES',var), size('VALUES')}>0, \text{required('VALUES',[val,omin,omax]), distinct('VALUES',val), 'VALUES'\`omin>=0, 'VALUES'\`omax='SEQ', 'VALUES'\`omin='VALUES'\`omax\}]).$$

$$\text{ctr\_example}(\text{sliding\_distribution},\text{sliding\_distribution}(\text{4},\text{[[var-0], [var-5], [var-0], [var-6], [var-5], [var-0], [var-0]], [[val-0,omin-1,omax-2], [val-1,omin-0,omax-4], [val-4,omin-0,omax-4], [val-5,omin-1,omax-2], [val-6,omin-0,omax-2]])).}$$

$$\text{ctr\_typical}$$
sliding_distribution,
['SEQ'>1,
'SEQ'<size('VARIABLES'),
size('VARIABLES')>size('VALUES'))).

ctr_exchangeable(
    sliding_distribution,
    [items('VARIABLES',reverse),
     vals{
         ['VARIABLES'\var],
         all(notin('VALUES'\val)),
         =,
         dontcare, 
dontcare),
     items('VALUES',all),
     vals(['VALUES'\omin],int(>=0),>,dontcare,dontcare),
     vals(['VALUES'\omax],int(<=('SEQ')),<,dontcare,dontcare),
     vals{
         ['VARIABLES'\var,'VALUES'\val],
         int,
         =\=,
         all,
         dontcare}]).

ctr_graph(
    sliding_distribution,
    ['VARIABLES'],
    SEQ,
    ['PATH'>>collection],
    [global_cardinality_low_up(collection,'VALUES')],
    ['NARC'=size('VARIABLES')-'SEQ'+1],
    []).

ctr_eval(
    sliding_distribution,
    [reformulation(sliding_distribution_r)]).

ctr_contractible(
    sliding_distribution,
    ['SEQ'=1],
    VARIABLES,
    any).

ctr_contractible(sliding_distribution,[],'VARIABLES',prefix).

ctr_contractible(sliding_distribution,[],'VARIABLES',suffix).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

```prolog
ctr_contractible(sliding_distribution,[],'VALUES',any).

sliding_distribution_r(SEQ,VARIABLES,VALUES) :-
    integer(SEQ),
    collection(VARIABLES,[dvar]),
    length(VARIABLES,L),
    SEQ>0,
    SEQ=<L,
    collection(VALUES,[int,int(0,L),int(0,L)]),
    length(VALUES,M),
    M>0,
    sliding_distribution1(VARIABLES,[],VALUES,SEQ).

sliding_distribution1([],_24062,_24063,_24064).

sliding_distribution1([Last|R],Seq,VALUES,SEQ) :-
    append(Seq,[Last],Sequence),
    length(Sequence,L),
    ( L>SEQ ->
        Sequence=[_24114|SeqCur],
        eval(global_cardinality_low_up(SeqCur,VALUES)),
        sliding_distribution1(R,SeqCur,VALUES,SEQ)
    ; L=SEQ ->
        eval(global_cardinality_low_up(Sequence,VALUES)),
        sliding_distribution1(R,Sequence,VALUES,SEQ)
    ; sliding_distribution1(R,Sequence,VALUES,SEQ)
).
```
B.315  sliding_sum

◊ META-DATA:

ctr_date(sliding_sum, []).

ctr_origin(sliding_sum, []).

ctr_synonyms(sliding_sum, []).

ctr_arguments(sliding_sum, []).

ctr_restrictions(sliding_sum, []).

ctr_example(sliding_sum, []).

ctr_typical(sliding_sum, []).
ctr_exchangeable(sliding_sum,[items('VARIABLES',reverse)]).

ctr_graph(
    sliding_sum,
    ['VARIABLES'],
    SEQ,
    ['PATH'>>collection],
    [sum_ctr(collection,>=,'LOW'),sum_ctr(collection,=<,'UP')],
    ['NARC'=size('VARIABLES')-'SEQ'+1],
    []).

ctr_eval(sliding_sum,[reformulation(sliding_sum_r)]).

ctr_contractible(sliding_sum,[SEQ=1],'VARIABLES',any).

ctr_contractible(sliding_sum,[],'VARIABLES',prefix).

ctr_contractible(sliding_sum,[],'VARIABLES',suffix).

sliding_sum_r(LOW,UP,SEQ,VARIABLES) :-
    integer(LOW),
    integer(UP),
    integer(SEQ),
    collection(VARIABLES,[dvar]),
    length(VARIABLES,L),
    UP>=LOW,
    SEQ>0,
    SEQ=<L,
    sliding_sum1(VARIABLES,[],LOW,UP,SEQ).

sliding_sum1([],_24744,_24745,_24746,_24747).

sliding_sum1([Last|R],Seq,LOW,UP,SEQ) :-
    append(Seq,[Last],Sequence),
    length(Sequence,L),
    ( L>SEQ ->
        Sequence=[_24799|SeqCur],
        eval(sum_ctr(SeqCur,>=,LOW)),
        eval(sum_ctr(SeqCur,=<,UP)),
        sliding_sum1(R,SeqCur,LOW,UP,SEQ)
    ; L=SEQ ->
        eval(sum_ctr(Sequence,>=,LOW)),
        eval(sum_ctr(Sequence,=<,UP)),
        sliding_sum1(R,Sequence,LOW,UP,SEQ)
    ; sliding_sum1(R,Sequence,LOW,UP,SEQ)
).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.316 sliding_time_window

◊ **Meta-Data:**

```prolog
ctr_date(
    sliding_time_window,
    ['20030820','20060815','20090530']).

ctr_origin(sliding_time_window,'N.˘Beldiceanu',[]).

ctr_arguments(
    sliding_time_window,
    ['WINDOW_SIZE'-int,
     'LIMIT'-int,
     'TASKS'-'collection(origin-dvar,duration-dvar)]).

ctr_restrictions(
    sliding_time_window,
    ['WINDOW_SIZE'>0,
     'LIMIT'>=0,
     required('TASKS',[origin,duration]),
     'TASKS' `duration>=0]).

ctr_example(
    sliding_time_window,
    sliding_time_window(9,
    6,
    [[origin-10,duration-3],
     [origin-5,duration-1],
     [origin-6,duration-2],
     [origin-14,duration-2],
     [origin-2,duration-2]]).

ctr_typical(
    sliding_time_window,
    ['WINDOW_SIZE'>1,
     'LIMIT'>0,
     'LIMIT'<sum('TASKS' `duration),
     size('TASKS')>1,
     'TASKS' `duration>0]).

ctr_exchangeable(
    sliding_time_window,
    [vals(['WINDOW_SIZE'],int,>,dontcare,dontcare),
     vals(['LIMIT'],int,<,dontcare,dontcare),
```
items('TASKS',all),
translate([['TASKS'~origin]),
vals([['TASKS'~duration],int(>=0),>,dontcare,dontcare])].

ctr_graph(
    sliding_time_window,
    ['TASKS'],
    2,
    ['CLIQUE'>>collection(tasks1,tasks2)],
    [tasks1~origin=<tasks2~origin,
      tasks2~origin-tasks1~origin<'WINDOW_SIZE'],
    [],
    [],
    ['SUCC'>>[source,tasks]],
    [sliding_time_window_from_start(
        WINDOW_SIZE,
        LIMIT,
        tasks,
        source~origin)]).

ctr_eval(
    sliding_time_window,
    [reformulation(sliding_time_window_r)]).

ctr_contractible(sliding_time_window,[],'TASKS',any).

sliding_time_window_r(WINDOW_SIZE,LIMIT,TASKS) :-
    integer(WINDOW_SIZE),
    WINDOW_SIZE>0,
    integer(LIMIT),
    LIMIT>=0,
    collection(TASKS,[dvar,dvar_gteq(0)]),
    get_attr1(TASKS,ORIGINS),
    get_attr2(TASKS,DURATIONS),
    sliding_time_window1(
        ORIGINS,
        DURATIONS,
        1,
        ORIGINS,
        DURATIONS,
        WINDOW_SIZE,
        LIMIT).
B.317  sliding_time_window_from_start

◊  Meta-Data:

ctr_date(
   sliding_time_window_from_start,
   ['20030820','20060815','20090530']).

ctr_origin(
   sliding_time_window_from_start,
   Used for defining %c.,
   [sliding_time_window]).

ctr_arguments(
   sliding_time_window_from_start,
   ['WINDOW_SIZE'-int,
    'LIMIT'-int,
    'TASKS'-collection(origin-dvar,duration-dvar),
    'START'-dvar]).

ctr_restrictions(
   sliding_time_window_from_start,
   ['WINDOW_SIZE'>0,
    'LIMIT'>=0,
    required('TASKS',[origin,duration]),
    'TASKS'`duration>=0]).

ctr_example(
   sliding_time_window_from_start,
   sliding_time_window_from_start(
      9,
      6,
      [[origin-10,duration-3],
       [origin-5,duration-1],
       [origin-6,duration-2]],
      5)).

ctr_typical(
   sliding_time_window_from_start,
   ['WINDOW_SIZE'>1,
    'LIMIT'>0,
    'LIMIT'<'WINDOW_SIZE',
    size('TASKS')>1,
    'TASKS'`duration>0]).

ctr_exchangeable(
sliding_time_window_from_start,
[vals(["WINDOW_SIZE"],int,>,dontcare,dontcare),
 vals(["LIMIT"],int,<,dontcare,dontcare),
 items(\('TASKS\',all),
 vals([\('TASKS'\^\text{duration}\],int(>=0)),>,dontcare,dontcare),
 translate([\('START','TASKS'\^\text{origin}]])].

ctr_derived_collections(
  sliding_time_window_from_start,
  [col('S'-collection(var-dvar),[item(var-'START')])]).

ctr_graph(
  sliding_time_window_from_start,
  ['S','TASKS'],
  2,
  ['PRODUCT']>>collection(s,tasks]),
  ['TRUE'],
  ['SUM_WEIGHT_ARC'](
    max(0,
     min(s\_var+'WINDOW_SIZE',
       tasks\_origin+tasks\_duration)-
     max(s\_var,tasks\_origin)))=<
    LIMIT],
  []).

ctr_eval(
  sliding_time_window_from_start,
  [reformulation(sliding_time_window_from_start_r)]).

ctr_contractible(sliding_time_window_from_start,[],'TASKS',any).

sliding_time_window_from_start_r(WINDOW_SIZE,LIMIT,TASKS,START) :-
  integer(WINDOW_SIZE),
  WINDOW_SIZE>0,
  integer(LIMIT),
  LIMIT>=0,
  collection(TASKS,[dvar,dvar_gteq(0)]),
  check_type(dvar,START),
  get_attr1(TASKS,ORIGINS),
  get_attr2(TASKS,DURATIONS),
  sliding_time_window1(
    [START],
    [WINDOW_SIZE],
    0,
    ORIGINS,
    DURATIONS,
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

WINDOW_SIZE,
LIMIT).
B.318  sliding_time_window_sum

◊ **META-DATA:**

```prolog
ctr_date(
    sliding_time_window_sum,
    ['20030820','20060815','20090530']).
```

```prolog
ctr_origin(
    sliding_time_window_sum,
    Derived from %c.,
    [sliding_time_window]).
```

```prolog
ctr_arguments(
    sliding_time_window_sum,
    ['WINDOW_SIZE'-int,
     'LIMIT'-int,
     'TASKS'—collection(origin-dvar,end-dvar,npoint-dvar)]).
```

```prolog
ctr_restrictions(
    sliding_time_window_sum,
    ['WINDOW_SIZE'>0,
     'LIMIT'>=0,
     required('TASKS',[origin,end,npoint]),
     'TASKS'\^origin='TASKS'\^end,
     'TASKS'\^npoint>=0]).
```

```prolog
ctr_example(
    sliding_time_window_sum,
    sliding_time_window_sum(
        9,
        16,
        [[origin-10,end-13,npoint-2],
         [origin-5,end-6,npoint-3],
         [origin-6,end-8,npoint-4],
         [origin-14,end-16,npoint-5],
         [origin-2,end-4,npoint-6]]).
```

```prolog
ctr_typical(
    sliding_time_window_sum,
    ['WINDOW_SIZE'>1,
     'LIMIT'>0,
     'LIMIT'<sum('TASKS'\^npoint),
     size('TASKS')>1,
     'TASKS'\^origin='TASKS'\^end,
     'TASKS'\^npoint>0]).
```
ctr_exchangeable(
    sliding_time_window_sum,
    [vals(["WINDOW_SIZE"],int,>,dontcare,dontcare),
     vals(["LIMIT"],int,<,dontcare,dontcare),
     items("TASKS",all),
     vals(["TASKS"^npoint],int(>=(0)),>,dontcare,dontcare),
     translate(["TASKS"^origin,"TASKS"^end])]).

ctr_graph(
    sliding_time_window_sum,
    ['TASKS'],
    1,
    ['SELF'>>collection(tasks)],
    [tasks^origin<tasks^end],
    ['NARC'=size('TASKS')],
    []).

ctr_graph(
    sliding_time_window_sum,
    ['TASKS'],
    2,
    ['CLIQUE'>>collection(tasks1,tasks2)],
    [tasks1^end<tasks2^end,
     tasks2^origin-tasks1^end<"WINDOW_SIZE"-1],
    [],
    [],
    [SUCC>>
     [source,
      variables-
      col('VARIABLES' collection(var-dvar),
          [item(var-'TASKS' npoint)]),
     [sum_ctr(variables,=<,'LIMIT')]].

ctr_eval(
    sliding_time_window_sum,
    [reformulation(sliding_time_window_sum_r)]).

c_tr_contractible(sliding_time_window_sum,[],'TASKS',any).

sliding_time_window_sum_r(WINDOW_SIZE,LIMIT,TASKS) :-
    integer(WINDOW_SIZE),
    WINDOW_SIZE>0,
    integer(LIMIT),
    LIMIT>=0,
    collection(TASKS,[dvar,dvar,dvar_gteq(0)]),
get_attr1(TASKS, ORIGINS),
get_attr2(TASKS, ENDS),
get_attr3(TASKS, NPOINTS),
sliding_time_window_sum1(
  ORIGINS, ENDS, NPOINTS, 1, ORIGINS, ENDS, NPOINTS, WINDOW_SIZE, LIMIT).

sliding_time_window_sum1([], [], [], [], _38356, _38402, _38448, _38494, _38540, _38586).

sliding_time_window_sum1([Oi|RO], [Ei|RE], [Pi|RP], I, ORIGINS, ENDS, NPOINTS, WINDOW_SIZE, LIMIT) :-
  Oi#=<Ei,
  sliding_time_window_sum2( ORIGINS, ENDS, NPOINTS, 1, Oi, Ei, Pi, I, WINDOW_SIZE,
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

LIMIT, SUM_NPOINTS),
call(SUM_NPOINTS#=<LIMIT),
I1 is I+1,
sliding_time_window_sum1(
  RO,
  RE,
  RP,
  I1,
  ORIGINS,
  ENDS,
  NPOINTS,
  WINDOW_SIZE,
  LIMIT).

sliding_time_window_sum2(
  [],
  [],
  [],
  _38365,
  _38411,
  _38457,
  _38503,
  _38549,
  _38595,
  _38641,
  0) :-
  !.

sliding_time_window_sum2(
  _37994|RO],
  _37998|RE],
  _38002|RP],
  J,
  O1,
  E1,
  Pi,
  I,
  WINDOW_SIZE,
  LIMIT,
  Pi+SUM) :-
  I=J,
  !,
  J1 is J+1,
  sliding_time_window_sum2( RO,
sliding_time_window_sum2(RO, RE, RP, J1, Oi, Ei, Pi, I, WINDOW_SIZE, LIMIT, SUM) :-
   I#=\=J,
   fd_max(Ej,MaxEj),
   fd_min(Oi,MinOi),
   MaxEj<MinOi,
   !, J1 is J+1,
   sliding_time_window_sum2(RO, RE, RP, J1, Oi, Ei, Pi, I, WINDOW_SIZE, LIMIT, SUM).

sliding_time_window_sum2(RO, RE, RP, J1, Oi, Ei, Pi, I, WINDOW_SIZE, LIMIT, SUM) :-
   I#=\=J,
   fd_max(Ej,MaxEj),
   fd_min(Oi,MinOi),
   MaxEj<MinOi,
\[ \begin{align*} &\text{J,} \\
&\text{Oi,} \\
&\text{Ei,} \\
&\text{Pi,} \\
&\text{I,} \\
&\text{WINDOW\_SIZE,} \\
&\text{LIMIT,} \\
&\text{SUM) :-} \\
&\text{I=\neq J,} \\
&\text{fd\_min(Oj,MinOj),} \\
&\text{fd\_max(Oi,MaxOi),} \\
&\text{E is MaxOi+WINDOW\_SIZE-1,} \\
&\text{MinOj>E,} \\
&\text{!,} \\
&\text{J1 is J+1,} \\
&\text{sliding\_time\_window\_sum2(} \\
&\text{RO,} \\
&\text{RE,} \\
&\text{RP,} \\
&\text{J1,} \\
&\text{Oi,} \\
&\text{Ei,} \\
&\text{Pi,} \\
&\text{I,} \\
&\text{WINDOW\_SIZE,} \\
&\text{LIMIT,} \\
&\text{SUM).} \\
\end{align*} \]

\[ \text{sliding\_time\_window\_sum2(} \\
\text{[Oj|RO],} \\
\text{[Ej|RE],} \\
\text{[Pj|RP]}, \\
\text{J,} \\
\text{Oi,} \\
\text{Ei,} \\
\text{Pi,} \\
\text{I,} \\
\text{WINDOW\_SIZE,} \\
\text{LIMIT,} \\
\text{min(1,max(0,min(Oi+WINDOW\_SIZE,Ej)-max(Oi,Oj)))*Pj+SUM) :-} \\
\text{J1 is J+1,} \\
\text{sliding\_time\_window\_sum2(} \\
\text{RO,} \\
\text{RE,} \\
\text{RP,} \\
\text{J1,} \]
Oi,
Ei,
P_i,
I,
WINDOW_SIZE,
LIMIT,
SUM).
B.319 smooth

◊ Meta-Data:

ctr_date(smooth, ['20000128', '20030820', '20040530', '20060815']).

ctr_origin(smooth, 'Derived from %c.', [change]).

ctr_arguments(
    smooth,
    ['NCHANGE'-dvar,
     'TOLERANCE'-int,
     'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    smooth,
    ['NCHANGE' >= 0,
     'NCHANGE' < size('VARIABLES'),
     'TOLERANCE' >= 0,
     required('VARIABLES', var)]).

ctr_example(
    smooth,
    smooth(1, 2, [[var-1], [var-3], [var-4], [var-5], [var-2]])).

ctr_typical(
    smooth,
    ['NCHANGE' > 0,
     'TOLERANCE' > 0,
     size('VARIABLES') > 2,
     range('VARIABLES' `var) > 1]).

ctr_exchangeable(
    smooth,
    [items('VARIABLES', reverse), translate(['VARIABLES' `var])])).

ctr_graph(
    smooth,
    ['VARIABLES'],
    2,
    ['PATH' => collection(variables1, variables2)],
    [abs(variables1 `var - variables2 `var) > 'TOLERANCE'],
    ['NARC' => 'NCHANGE'],
    []).

ctr_eval(smooth, [automaton(smooth_a)]).
ctr_pure_functional_dependency(smooth,[]).

ctr_functional_dependency(smooth,1,[2,3]).

ctr_contractible(smooth,[‘NCHANGE’=0],’VARIABLES’,prefix).

ctr_contractible(smooth,[‘NCHANGE’=0],’VARIABLES’,suffix).

ctr_contractible(
  smooth,
  [‘NCHANGE’=size(‘VARIABLES’)-1],
  VARIABLES,
  prefix).

ctr_contractible(
  smooth,
  [‘NCHANGE’=size(‘VARIABLES’)-1],
  VARIABLES,
  suffix).

smooth_a(FLAG,NCHANGE,TOLERANCE,VARIABLES) :-
  collection(VARIABLES,[dvar]),
  length(VARIABLES,N),
  N_1 is N-1,
  check_type(dvar(0,N_1),NCHANGE),
  integer(TOLERANCE),
  TOLERANCE>=0,
  smooth_signature(VARIABLES,SIGNATURE,TOLERANCE),
  automaton(
    SIGNATURE,
    _34799,
    SIGNATURE,
    [source(s),sink(s)],
    [arc(s,1,s,[C+1]),arc(s,0,s)],
    [C],
    [0],
    [COUNT]),
  COUNT#=NCHANGE#<=>FLAG.

smooth_signature([],[],_32967).

smooth_signature([_32971],[],_32970) :-
  !.

smooth_signature([[var-VAR1],[var-VAR2]|VARs],_32970,_,TOLERANCE) :-
abs(VAR1-VAR2) #> TOLERANCE #<=> S #= 1,
smooth_signature([[var-VAR2]|VARs], Ss, TOLERANCE).
2779

B.320  \textbf{soft\_all\_equal\_max\_var}

\textbf{META-DATA:}

\begin{verbatim}
ctr_date(soft_all_equal_max_var,'20090926').
ctr_origin(soft_all_equal_max_var,
cite{HebrardMarxSullivanRazgon09}).
ctr_arguments(soft_all_equal_max_var,
['N'-dvar,'VARIABLES'-collection(var-dvar)]).
ctr_restrictions(soft_all_equal_max_var,
['N'>=0,'N'=<size('VARIABLES'),required('VARIABLES',var)]).
ctr_example(soft_all_equal_max_var,
soft_all_equal_max_var(1,
[[var-5],[var-1],[var-5],[var-5]])).
ctr_typical(soft_all_equal_max_var,
['N'>0,'N'=<size('VARIABLES'),size('VARIABLES')>1]).
ctr_exchangeable(soft_all_equal_max_var,
[vals(['N'],int(>=(0)),>,dontcare,dontcare),
items('VARIABLES',all),
vals(['VARIABLES'ˆvar],int,=\=,all,dontcare)]).
ctr_graph(soft_all_equal_max_var,
['VARIABLES'],
2,
[CLIQUE']>>collection(variables1,variables2)],
[variables1^var=variables2^var],
[MAX_NSCC'=<size('VARIABLES')-'N'],
[]).
ctr_eval(soft_all_equal_max_var,

\end{verbatim}
soft_all_equal_max_var_r(N,VARIABLES) :-
    check_type(dvar_gteq(0),N),
    collection(VARIABLES,[dvar]),
    length(VARIABLES,L),
    get_attr1(VARIABLES,VARS),
    get_minimum(VARS,MINVARS),
    get_maximum(VARS,MAXVARS),
    complete_card(MINVARS,MAXVARS,L,OCC,VAL_OCC),
    global_cardinality(VARS,VAL_OCC),
    MAX_OCC in 0..L,
    eval(maximum(MAX_OCC,OCC)),
    call(N#=<L-MAX_OCC).
B.321  \textbf{soft}\textunderscore all\textunderscore equal\textunderscore min\textunderscore ctr}  

\textbf{META-DATA:} 

\texttt{ctr}\textunderscore date(soft\textunderscore all\textunderscore equal\textunderscore min\textunderscore ctr, ['20081004']).

\texttt{ctr}\textunderscore origin(

soft\textunderscore all\textunderscore equal\textunderscore min\textunderscore ctr,
\cite{HebrardSullivanRazgon08}, []).

\texttt{ctr}\textunderscore synonyms(

soft\textunderscore all\textunderscore equal\textunderscore min\textunderscore ctr,
[soft\textunderscore alldiff\textunderscore max\textunderscore ctr,
soft\textunderscore alldifferent\textunderscore max\textunderscore ctr,
soft\textunderscore alldistinct\textunderscore max\textunderscore ctr]).

\texttt{ctr}\textunderscore arguments(

soft\textunderscore all\textunderscore equal\textunderscore min\textunderscore ctr,
['N'-int,'VARIABLES'-collection(var-dvar)]).

\texttt{ctr}\textunderscore restrictions(

soft\textunderscore all\textunderscore equal\textunderscore min\textunderscore ctr,
['N'>=0,
  N<
  size('VARIABLES')*size('VARIABLES')-size('VARIABLES'),
  required('VARIABLES',var)]).

\texttt{ctr}\textunderscore example(

soft\textunderscore all\textunderscore equal\textunderscore min\textunderscore ctr,

soft\textunderscore all\textunderscore equal\textunderscore min\textunderscore ctr(  
6,
  [[var-5],[var-1],[var-5],[var-5]])).

\texttt{ctr}\textunderscore typical(

soft\textunderscore all\textunderscore equal\textunderscore min\textunderscore ctr,
['N'>0,
  N<
  size('VARIABLES')*size('VARIABLES')-size('VARIABLES'),
  size('VARIABLES')>1]).

\texttt{ctr}\textunderscore exchangeable(

soft\textunderscore all\textunderscore equal\textunderscore min\textunderscore ctr,
[vals(['N'],int(>=(0)),>,dontcare,dontcare),
  items('VARIABLES',all),
  vals(['VARIABLES''var'],int,=\_,all,dontcare)].
ctr_graph(
    soft_all_equal_min_ctr,
    ['VARIABLES'],
    2,
    ['CLIQUE'(\=\>)collection(variables1,variables2)},
    [variables1^var=variables2^var],
    ['NARC'='N'],
    []).

ctr_eval(
    soft_all_equal_min_ctr,
    [reformulation(soft_all_equal_min_ctr_r)])).

soft_all_equal_min_ctr_r(N,VARIABLES) :-
    collection(VARIABLES,[dvar]),
    length(VARIABLES,L),
    L2 is L*L-L,
    check_type(dvar(0,L2),N),
    get_attr1(VARIABLES,VARS),
    soft_all_equal_min_ctr1(VARS,TERM),
    call(N#=<TERM).

soft_all_equal_min_ctr1([],0).

soft_all_equal_min_ctr1([V|R],S+T) :-
    soft_all_equal_min_ctr2(R,V,S),
    soft_all_equal_min_ctr1(R,T).

soft_all_equal_min_ctr2([],_29900,0).

soft_all_equal_min_ctr2([U|R],V,2*B+T) :-
    B#=<\>U#=V,
    soft_all_equal_min_ctr2(R,V,T).
B.322 soft_all_equal_min_var

◊ Meta-Data:

ctr_date(soft_all_equal_min_var,[‘20090926’]).

ctr_origin(
    soft_all_equal_min_var,
    \cite{HebrardMarxSullivanRazgon09}, []).

ctr_arguments(
    soft_all_equal_min_var,
    [‘N’-dvar,’VARIABLES’-collection(var-dvar)]).

ctr_restrictions(
    soft_all_equal_min_var,
    [‘N’>=0,required(‘VARIABLES’,var)]).

ctr_example(
    soft_all_equal_min_var,
    soft_all_equal_min_var(1,
        [[var-5],[var-1],[var-5],[var-5]])).

ctr_typical(soft_all_equal_min_var,[‘N’>0,size(‘VARIABLES’)>1]).

ctr_exchangeable(
    soft_all_equal_min_var,
    [vals([‘N’],int,<,dontcare,dontcare),
        items(‘VARIABLES’,all),
        vals([‘VARIABLES’^var],int,=\=,all,dontcare)]).

ctr_graph(
    soft_all_equal_min_var,
    [‘VARIABLES’],
    2,
    [‘CLIQUE’>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    [‘MAX_NSCC’=size(‘VARIABLES’)-’N’],
    []).

ctr_eval(
    soft_all_equal_min_var,
    [reformulation(soft_all_equal_min_var_r)]).
soft_all_equal_min_var_r(N,VARIABLES) :-
    check_type(dvar_gteq(0),N),
collection(VARIABLES,[dvar]),
length(VARIABLES,L),
get_attr1(VARIABLES,VARS),
get_minimum(VARS,MINVARS),
get_maximum(VARS,MAXVARS),
complete_card(MINVARS,MAXVARS,L,OCC,VAL_OCC),
global_cardinality(VARS,VAL_OCC),
MAX_OCC in 0..L,
eval(maximum(MAX_OCC,OCC)),
call(N#>=L-MAX_OCC).
B.323  soft_alldifferent_ctr

◊ **META-DATA:**

ctr_date(
    soft_alldifferent_ctr,
    ['20030820', '20060815', '20090926']).

ctr_origin(
    soft_alldifferent_ctr,
    \cite{PetitReginBessiere01}, []).

ctr_synonyms(
    soft_alldifferent_ctr,
    [soft_alldiff_ctr,
     soft_alldistinct_ctr,
     soft_alldiff_min_ctr,
     soft_alldifferent_min_ctr,
     soft_alldistinct_min_ctr,
     soft_all_equal_max_ctr]).

ctr_arguments(
    soft_alldifferent_ctr,
    ['C'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    soft_alldifferent_ctr,
    ['C'\geq0, required('VARIABLES', var)]).

ctr_example(
    soft_alldifferent_ctr,
    soft_alldifferent_ctr(4,
        [[var-5],[var-1],[var-9],[var-1],[var-5],[var-5]])).

ctr_typical(soft_alldifferent_ctr, ['C'\geq0, size('VARIABLES')\geq1]).

ctr_exchangeable(
    soft_alldifferent_ctr,
    [vals(['C'],int,<,dontcare,dontcare),
     items('VARIABLES',all),
     vals(['VARIABLES'\^\textsf{\textasciitilde}var],int,\texttt{\textasciitilde}=,all,dontcare)]).

ctr_graph(
    soft_alldifferent_ctr,
['VARIABLES'],
2,
['CLIQUE'(<)\>collection(variables1,variables2)],
[variables1\>var=variables2\>var],
['NARC'=<'C'],
[]).

ctr_eval(
    soft_alldifferent_ctr,
    [reformulation(soft_alldifferent_ctr_r)]).

soft_alldifferent_ctr_r(C,VARIABLES) :-
    length(VARIABLES,N),
    N2 is (N+N-N)//2,
    check_type(dvar(0,N2),C),
    collection(VARIABLES,[dvar]),
    get_attr1(VARIABLES,VARS),
    soft_alldifferent_ctr1(VARS,TERM),
    call(C#>=TERM).

soft_alldifferent_ctr1([],0).

soft_alldifferent_ctr1([V|R],S+T) :-
    soft_alldifferent_ctr2(R,V,S),
    soft_alldifferent_ctr1(R,T).

soft_alldifferent_ctr2([],_35279,0).

soft_alldifferent_ctr2([U|R],V,B+T) :-
    B#<=U#=V,
    soft_alldifferent_ctr2(R,V,T).
B.324  soft_alldifferent_var

◊ **META-DATA:**

ctr_date(soft_alldifferent_var, ['20030820', '20060815', '20090926']).

ctr_origin(soft_alldifferent_var, \cite{PetitReginBessiere01}, []).

ctr_synonyms(soft_alldifferent_var, [soft_alldiff_var, soft_alldistinct_var, soft_alldiff_min_var, soft_alldifferent_min_var, soft_alldistinct_min_var]).

ctr_arguments(soft_alldifferent_var, ['C'\-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(soft_alldifferent_var, ['C'>=0, required('VARIABLES', var)]).

ctr_example(soft_alldifferent_var, soft_alldifferent_var(3, [[var-5],[var-1],[var-9],[var-1],[var-5],[var-5]])).

ctr_typical(soft_alldifferent_var, ['C'>0,2*'C'=<size('VARIABLES'), size('VARIABLES')>1]).

ctr_exchangeable(soft_alldifferent_var, [vals(['C'],int,<,dontcare,dontcare), items('VARIABLES', all), vals(['VARIABLES'\-var],int,\=,all,dontcare)]).

ctr_graph(

"""

""

"""
soft_alldifferent_var,
['VARIABLES'],
2,
['CLIQUE'>>collection(variables1,variables2)],
[variables1^'var=variables2^'var],
['NSCC'>=size('VARIABLES')-'C'],
[]).

ctr_eval(
  soft_alldifferent_var,
  [reformulation(soft_alldifferent_var_r)]).

soft_alldifferent_var_r(C,VARIABLES) :-
  length(VARIABLES,N),
  check_type(dvar(0,N),C),
  collection(VARIABLES,[dvar]),
  eval(in_interval(M,1,N)),
  eval(nvalue(M,VARIABLES)),
  C#>=N-M.
B.325  soft_cumulative

◊ **META-DATA:**

ctr_predefined(soft_cumulative).

ctr_date(soft_cumulative,[’20091121’]).

ctr_origin(soft_cumulative,’Derived from %c’,[cumulative]).

ctr_arguments(
    soft_cumulative,
    [TASKS-
        collection(
            origin-dvar,
            duration-dvar,
            end-dvar,
            height-dvar),
            ’LIMIT’-int,
            ’INTERMEDIATE_LEVEL’-int,
            ’SURFACE_ON_TOP’-dvar]).

ctr_restrictions(
    soft_cumulative,
    [require_at_least(2,’TASKS’,[origin,duration,end]),
     required(’TASKS’,height),
     ’TASKS’^duration>=0,
     ’TASKS’^origin=<’TASKS’^end,
     ’TASKS’^height>=0,
     ’LIMIT’>=0,
     ’INTERMEDIATE_LEVEL’>=0,
     ’INTERMEDIATE_LEVEL’=<’LIMIT’,
     ’SURFACE_ON_TOP’>=0]).

ctr_example(
    soft_cumulative,
    soft_cumulative(
        [[[origin-1,duration-4,end-5,height-1],
          [origin-1,duration-1,end-3,height-2],
          [origin-3,duration-3,end-6,height-2]],
        3,
        2,
        3)).

ctr_typical(
    soft_cumulative,
[size('TASKS') > 1, 
  range('TASKS'ˆorigin) > 1, 
  range('TASKS'ˆduration) > 1, 
  range('TASKS'ˆend) > 1, 
  range('TASKS'ˆheight) > 1, 
  'TASKS'ˆduration > 0, 
  'TASKS'ˆheight > 0, 
  'LIMIT' < sum('TASKS'ˆheight), 
  'INTERMEDIATE_LEVEL' > 0, 
  'INTERMEDIATE_LEVEL' < 'LIMIT', 
  'SURFACE_ON_TOP' > 0]).

ctr_exchangeable(
  soft_cumulative,
  [items('TASKS', all), 
   translate(['TASKS'ˆorigin, 'TASKS'ˆend]), 
   vals(['LIMIT'], int, <, dontcare, dontcare))).
B.326  soft_same_interval_var

◊ **META-DATA:**

```prolog
ctr_date(soft_same_interval_var,['20050507','20060815']).
```

```prolog
ctr_origin(soft_same_interval_var, Derived from %c, [same_interval]).
```

```prolog
ctr_synonyms(soft_same_interval_var,[soft_same_interval]).
```

```prolog
ctr_arguments(soft_same_interval_var, ['C'-dvar, 'VARIABLES1'-collection(var-dvar), 'VARIABLES2'-collection(var-dvar), 'SIZE_INTERVAL'-int]).
```

```prolog
ctr_restrictions(soft_same_interval_var, ['C'>=0, 'C'=<size('VARIABLES1'), size('VARIABLES1')=size('VARIABLES2'), required('VARIABLES1',var), required('VARIABLES2',var), 'SIZE_INTERVAL'>0]).
```

```prolog
ctr_example(soft_same_interval_var, soft_same_interval_var(4, [[var-9],[var-9],[var-9],[var-9],[var-9],[var-9]], [[var-9],[var-1],[var-1],[var-1],[var-1],[var-8]], 3)).
```

```prolog
ctr_typical(soft_same_interval_var, ['C'>0, size('VARIABLES1')>1, range('VARIABLES1'`var)>1, range('VARIABLES2'`var)>1, 'SIZE_INTERVAL'>1, 'SIZE_INTERVAL'<range('VARIABLES1'`var), 'SIZE_INTERVAL'<range('VARIABLES2'`var))].
```
ctr_exchangeable(Key,Soft_Same_Interval_Var, [Args([C], [VARIABLES1, VARIABLES2], [SIZE_INTERVAL]), Items('VARIABLES1', all), Items('VARIABLES2', all), Vals([VARIABLES1 `var], intervals('SIZE_INTERVAL'), =, dontcare, dontcare), Vals([VARIABLES2 `var], intervals('SIZE_INTERVAL'), =, dontcare, dontcare))].

ctr_graph(Key,Soft_Same_Interval_Var, ['VARIABLES1', 'VARIABLES2'], 2, ['PRODUCT'>>collection(variables1,variables2)], [variables1 `var`/SIZE_INTERVAL=variables2 `var`/SIZE_INTERVAL], ['NSINK_NSOURCE'=size('VARIABLES1')-'C'], []).

ctr_eval(Key,Soft_Same_Interval_Var, [reformulation(soft_same_interval_var_r)]).

soft_same_interval_var_r(Key, VARIABLES1, VARIABLES2, SIZE_INTERVAL) :-
    length(VARIABLES1, L1),
    length(VARIABLES2, L2),
    L1=L2,
    check_type(dvar(0,L1),C),
    collection(VARIABLES1, [dvar]),
    collection(VARIABLES2, [dvar]),
    integer(SIZE_INTERVAL),
    SIZE_INTERVAL>0,
    get_attr1(VARIABLES1, VARS1),
    ...
get_attr1(VARIABLES2, VARS2),
gen_quotient(VARS1, SIZE_INTERVAL, QUOTVARS1),
gen_quotient(VARS2, SIZE_INTERVAL, QUOTVARS2),
gen_collection(QUOTVARS1, var, CVARS1),
gen_collection(QUOTVARS2, var, CVARS2),
eval(soft_same_var(C, CVARS1, CVARS2)).
B.327  soft_same_modulo_var

◊ Meta-Data:

ctr_date(soft_same_modulo_var,['20050507','20060815']).

ctr_origin(
    soft_same_modulo_var,
    Derived from %c,
    [samemodulo]).

ctr_synonyms(soft_same_modulo_var,[soft_same_modulo]).

ctr_arguments(
    soft_same_modulo_var,
    ['C'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'M'-int]).

ctr_restrictions(
    soft_same_modulo_var,
    ['C'>=0,
     'C'=<size('VARIABLES1'),
     size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     'M'>0]).

ctr_example(
    soft_same_modulo_var,
    soft_same_modulo_var(
        4,
        [[var-9],[var-9],[var-9],[var-9],[var-9],[var-9],[var-9],[var-9],[var-1]],
        [[var-9],[var-1],[var-1],[var-1],[var-1],[var-1],[var-1],[var-1],[var-8]],
        3)).

ctr_typical(
    soft_same_modulo_var,
    ['C'>0,
     size('VARIABLES1')>1,
     range('VARIABLES1'\var)>1,
     range('VARIABLES2'\var)>1,
     'M'>1,
     'M'<maxval('VARIABLES1'\var),
     'M'<maxval('VARIABLES2'\var))].
ctr_exchangeable(
    soft_same_modulo_var,
    [args([[C'],[VARIABLES1','VARIABLES2'],[M']])],
    items('VARIABLES1',all),
    items('VARIABLES2',all),
    vals([VARIABLES1'\text{\textasciitilde}var],mod('M'),=,dontcare,dontcare),
    vals([VARIABLES2'\text{\textasciitilde}var],mod('M'),=,dontcare,dontcare)).

ctr_graph(
    soft_same_modulo_var,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT']>>collection(variables1,variables2)],
    [variables1'\text{\textasciitilde}var mod 'M'=variables2'\text{\textasciitilde}var mod 'M'],
    ['NSINK_NSOURCE'=size('VARIABLES1')-'C'],
    []).

ctr_eval(
    soft_same_modulo_var,
    [reformulation(soft_same_modulo_var_r)]).

soft_same_modulo_var_r(C,VARIABLES1,VARIABLES2,M) :-
    length(VARIABLES1,L1),
    length(VARIABLES2,L2),
    L1=L2,
    check_type(dvar(0,L1),C),
    collection(VARIABLES1,[dvar]),
    collection(VARIABLES2,[dvar]),
    integer(M),
    M>0,
    get_attr1(VARIABLES1,VARS1),
    get_attr1(VARIABLES2,VARS2),
    gen_remainder(VARS1,M,REMVARS1),
    gen_remainder(VARS2,M,REMVARS2),
    gen_collection(REMVARS1,var,CVARS1),
    gen_collection(REMVARS2,var,CVARS2),
    eval(soft_same_var(C,CVARS1,CVARS2)).
B.328  soft_same_partition_var

♦ Meta-Data:

ctr_date(soft_same_partition_var, ['20050507', '20060816']).

ctr_origin(
    soft_same_partition_var,
    Derived from %c,
    [same_partition]).

ctr_synonyms(soft_same_partition_var, [soft_same_partition]).

ctr_types(
    soft_same_partition_var,
    ['VALUES'-collection(val-int)]).

ctr_arguments(
    soft_same_partition_var,
    ['C'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    soft_same_partition_var,
    ['C'>=0,
     'C'<size('VARIABLES1'),
     size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var),
     required('PARTITIONS', p),
     size('PARTITIONS')>=2,
     size('VALUES')>=1,
     required('VALUES', val),
     distinct('VALUES', val))].

ctr_example(
    soft_same_partition_var,
    soft_same_partition_var(4,
       [[var-9], [var-9], [var-9], [var-9], [var-9], [var-1]],
       [[var-9], [var-1], [var-1], [var-1], [var-1], [var-8]],
       [[p-[[val-1], [val-2]]],
        [p-[[val-9]]],
        [p-[[val-7], [val-8]]])).
ctr_typical(
    soft_same_partition_var,
    ['C'] > 0,
    size('VARIABLES1') > 1,
    range('VARIABLES1' \ var) > 1,
    range('VARIABLES2' \ var) > 1,
    size('VARIABLES1') > size('PARTITIONS'),
    size('VARIABLES2') > size('PARTITIONS')).

ctr_exchangeable(
    soft_same_partition_var,
    [args([['C'], ['VARIABLES1', 'VARIABLES2'], ['PARTITIONS']]),
     items('VARIABLES1', all),
     items('VARIABLES2', all),
     items('PARTITIONS', all),
     items('PARTITIONS' \ p, all),
     vals(
         ['VARIABLES1' \ var],
         part('PARTITIONS'),
         =, dontcare, dontcare),
     vals(
         ['VARIABLES2' \ var],
         part('PARTITIONS'),
         =, dontcare, dontcare)).

ctr_graph(
    soft_same_partition_var,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT' >> collection(variables1, variables2)],
    [in_same_partition(
        variables1 \ var,
        variables2 \ var,
        PARTITIONS)],
    ['NSINK_NSOURCE' = size('VARIABLES1') - 'C'],
    []).

ctr_eval(
    soft_same_partition_var,
    [reformulation(soft_same_partition_var_r)]).
soft_same_partition_var_r(C,VARIABLES1,VARIABLES2,PARTITIONS) :-
    length(VARIABLES1,L1),
    length(VARIABLES2,L2),
    L1=L2,
    check_type(dvar(0,L1),C),
    collection(VARIABLES1,[dvar]),
    collection(VARIABLES2,[dvar]),
    collection(PARTITIONS,[col_len_gteq(1,[int])]),
    get_attr1(VARIABLES1,VARS1),
    get_attr1(VARIABLES2,VARS2),
    get_col_attr1(PARTITIONS,1,PVALS),
    flattern(PVALS,VALS),
    all_different(VALS),
    length(PARTITIONS,M),
    M>1,
    length(PVALS,LPVALS),
    get_partition_var(VARS1,PVALS,PVARS1,LPVALS,0),
    get_partition_var(VARS2,PVALS,PVARS2,LPVALS,0),
    gen_collection(PVARS1,var,CVARS1),
    gen_collection(PVARS2,var,CVARS2),
    eval(soft_same_var(C,CVARS1,CVARS2)).
B.329  soft_same_var

◊ Meta-Data:

```prolog
ctr_date(soft_same_var,['20050507','20060816','20090522']).
ctr_origin(soft_same_var,'\cite{vanHoeve05}',[]).
ctr_synonyms(soft_same_var,[soft_same]).
ctr_arguments(
  soft_same_var,
  ['C'-dvar,
   'VARIABLES1'-collection(var-dvar),
   'VARIABLES2'-collection(var-dvar)]).
ctr_restrictions(
  soft_same_var,
  ['C'>=0,
   'C'=<size('VARIABLES1'),
   size('VARIABLES1')=size('VARIABLES2'),
   required('VARIABLES1',var),
   required('VARIABLES2',var)]).
ctr_example(
  soft_same_var,
  soft_same_var(4,
    [[var-9],[var-9],[var-9],[var-9],[var-9],[var-1]],
    [[var-9],[var-1],[var-1],[var-1],[var-1],[var-1],[var-8]])).
ctr_typical(
  soft_same_var,
  ['C'>0,
   size('VARIABLES1')>1,
   range('VARIABLES1'\^var)>1,
   range('VARIABLES2'\^var)>1]).
ctr_exchangeable(
  soft_same_var,
  [args([['C'],['VARIABLES1','VARIABLES2']]),
   items('VARIABLES1',all),
   items('VARIABLES2',all),
   vals(
     ['VARIABLES1'\^var,'VARIABLES2'\^var],
     int,
```
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

\texttt{ctr\_graph(}
\texttt{soft\_same\_var,}
\texttt{[\textquotearm{VARIABLES1'},\textquotearm{VARIABLES2'}],}
\texttt{2,}
\texttt{[\textquotequotearm{PRODUCT'}\texttt{>>}\texttt{collection(variables1,variables2)}],}
\texttt{[variables1\texttt{\^}var=variables2\texttt{\^}var],}
\texttt{[\textquotequotearm{NSINK\_NSOURCE'}=\texttt{size('VARIABLES1')-\textquotearm{C'}},}
\texttt{[\textquotequotequotearm{]}].}
\texttt{ctr\_eval(soft\_same\_var,\texttt{[reformulation(soft\_same\_var\_r)]}).}

\texttt{soft\_same\_var\_r(C,VARIABLES1,VARIABLES2) :-}
\texttt{length(VARIABLES1,L1),}
\texttt{length(VARIABLES2,L2),}
\texttt{L1=L2,}
\texttt{check\_type(dvar(0,L1),C),}
\texttt{collection(VARIABLES1,[dvar]),}
\texttt{collection(VARIABLES2,[dvar]),}
\texttt{eval(soft\_used\_by\_var(C,VARIABLES1,VARIABLES2)).}
B.330  soft_used_by_interval_var

◊ **META-DATA:**

ctr_date(soft_used_by_interval_var, ['20050507', '20060816']).

ctr_origin(soft_used_by_interval_var, Derived from %c., [used_by_interval]).

ctr_synonyms(soft_used_by_interval_var, [soft_used_by_interval]).

ctr_arguments(soft_used_by_interval_var, ['C'-dvar, 'VARIABLES1'-collection(var-dvar), 'VARIABLES2'-collection(var-dvar), 'SIZE_INTERVAL'-int]).

ctr_restrictions(soft_used_by_interval_var, ['C'>=0, 'C'=<size('VARIABLES2'), size('VARIABLES1')>=size('VARIABLES2'), required('VARIABLES1', var), required('VARIABLES2', var), 'SIZE_INTERVAL'>0]).

ctr_example(soft_used_by_interval_var, soft_used_by_interval_var(2, [[var-9], [var-1], [var-1], [var-8], [var-8]], [[var-9], [var-9], [var-9], [var-1]], 3)).

ctr_typical(soft_used_by_interval_var, ['C']>0, size('VARIABLES1')>1, size('VARIABLES2')>1, range('VARIABLES1'\`var)>1, range('VARIABLES2'\`var)>1, 'SIZE_INTERVAL'>1, 'SIZE_INTERVAL'<range('VARIABLES1'\`var),
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

'SIZE_INTERVAL' <range('VARIABLES2'~var)).

ctr_exchangeable(
    soft_used_by_interval_var,
    [items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(
         ['VARIABLES1'~var],
         intervals('SIZE_INTERVAL'),
         =,
         dontcare,
         dontcare),
     vals(
         ['VARIABLES2'~var],
         intervals('SIZE_INTERVAL'),
         =,
         dontcare,
         dontcare))).

ctr_graph(
    soft_used_by_interval_var,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1~var/'SIZE_INTERVAL' =
     variables2~var/'SIZE_INTERVAL'],
    ['NSINK_NSOURCE'=size('VARIABLES2')-'C'],
    []).

ctr_eval(
    soft_used_by_interval_var,
    [reformulation(soft_used_by_interval_var_r)]).

soft_used_by_interval_var_r(
    C,
    VARIABLES1,
    VARIABLES2,
    SIZE_INTERVAL) :-
    length(VARIABLES1,L1),
    length(VARIABLES2,L2),
    L1>=L2,
    check_type(dvar(0,L2),C),
    collection(VARIABLES1,[dvar]),
    collection(VARIABLES2,[dvar]),
    integer(SIZE_INTERVAL),
    SIZE_INTERVAL>0,
get_attr1(VARIABLES1,VARS1),
get_attr1(VARIABLES2,VARS2),
gen_quotient(VARS1,SIZE_INTERVAL,QUOTVARS1),
gen_quotient(VARS2,SIZE_INTERVAL,QUOTVARS2),
gen_collection(QUOTVARS1,var,CVARS1),
gen_collection(QUOTVARS2,var,CVARS2),
eval(soft_used_by_var(C,CVARS1,CVARS2)).
B.331  soft_used_by_modulo_var

◊ **Meta-Data:**

```prolog
ctr_date(soft_used_by_modulo_var, ['20050507','20060816']).
```

```prolog
ctr_origin(
    soft_used_by_modulo_var,
    Derived from %c,
    [used_by_modulo]).
```

```prolog
ctr_synonyms(soft_used_by_modulo_var, [soft_used_by_modulo]).
```

```prolog
ctr_arguments(
    soft_used_by_modulo_var,
    ['C'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'M'-int]).
```

```prolog
ctr_restrictions(
    soft_used_by_modulo_var,
    ['C'>=0,
     'C'=<size('VARIABLES2'),
     size('VARIABLES1')>=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     'M'>0]).
```

```prolog
ctr_example(
    soft_used_by_modulo_var,
    soft_used_by_modulo_var(2,
      [[var-9],[var-1],[var-1],[var-8],[var-8]],
      [[var-9],[var-9],[var-9],[var-8],[var-8]],
      3)).
```

```prolog
ctr_typical(
    soft_used_by_modulo_var,
    ['C'>0,
     size('VARIABLES1')>1,
     size('VARIABLES2')>1,
     range('VARIABLES1'\^var)>1,
     range('VARIABLES2'\^var)>1,
     'M'>1,
     'M' < maxval('VARIABLES1'\^var),
```
'M' < maxval(‘VARIABLES2’^var)).

ctr_exchangeable(
    soft_used_by_modulo_var,
    [items(‘VARIABLES1’, all),
     items(‘VARIABLES2’, all),
     vals([‘VARIABLES1’^var], mod(‘M’), =, dontcare, dontcare),
     vals([‘VARIABLES2’^var], mod(‘M’), =, dontcare, dontcare)]).

ctr_graph(
    soft_used_by_modulo_var,
    ['VARIABLES1', 'VARIABLES2'],
    2,
    ['PRODUCT']>> collection([variables1, variables2]),
    [variables1^var mod 'M' = variables2^var mod 'M'],
    ['NSINK_NSOURCE' = size('VARIABLES2') - 'C'],
    []).

ctr_eval(
    soft_used_by_modulo_var,
    [reformulation(soft_used_by_modulo_var_r)]).

soft_used_by_modulo_var_r(C, VARIABLES1, VARIABLES2, M) :-
    length(VARIABLES1, L1),
    length(VARIABLES2, L2),
    L1 >= L2,
    check_type(dvar(0, L2), C),
    collection(VARIABLES1, [dvar]),
    collection(VARIABLES2, [dvar]),
    integer(M),
    M > 0,
    get_attr1(VARIABLES1, VAR1),
    get_attr1(VARIABLES2, VAR2),
    gen_remainder(VAR1, M, REMVAR1),
    gen_remainder(VAR2, M, REMVAR2),
    gen_collection(REMVAR1, var, CVAR1),
    gen_collection(REMVAR2, var, CVAR2),
    eval(soft_used_by_var(C, CVAR1, CVAR2)).
B.332 soft_used_by_partition_var

◊ META-DATA:

ctr_date(soft_used_by_partition_var, [‘20050507’, ‘20060816’]).

ctr_origin(
  soft_used_by_partition_var,
  Derived from %c.,
  [used_by_partition]).

ctr_synonyms(
  soft_used_by_partition_var,
  [soft_used_by_partition]).

ctr_types(
  soft_used_by_partition_var,
  [‘VALUES’-collection(val-int)]).

ctr_arguments(
  soft_used_by_partition_var,
  [‘C’-dvar,
   ‘VARIABLES1’-collection(var-dvar),
   ‘VARIABLES2’-collection(var-dvar),
   ‘PARTITIONS’-collection(p-‘VALUES’)]).

ctr_restrictions(
  soft_used_by_partition_var,
  [‘C’>=0,
   ‘C’=<size(‘VARIABLES2’),
   size(‘VARIABLES1’)>=size(‘VARIABLES2’),
   required(‘VARIABLES1’, var),
   required(‘VARIABLES2’, var),
   required(‘PARTITIONS’, p),
   size(‘PARTITIONS’)>=2,
   size(‘VALUES’)>=1,
   required(‘VALUES’, val),
   distinct(‘VALUES’, val)]).

ctr_example(
  soft_used_by_partition_var,
  soft_used_by_partition_var(2,
    [[var-9], [var-1], [var-1], [var-8], [var-8]],
    [[var-9], [var-9], [var-9], [var-1]],
    [[p-[[val-1], [val-2]]],
    ...])
ctr_typical(
  soft_used_by_partition_var,
  ['C'>0,
   size('VARIABLES1')>1,
   size('VARIABLES2')>1,
   range('VARIABLES1'\^var)>1,
   range('VARIABLES2'\^var)>1,
   size('VARIABLES1')>size('PARTITIONS'),
   size('VARIABLES2')>size('PARTITIONS'))).

ctr_exchangeable(
  soft_used_by_partition_var,
  [items('VARIABLES1',all),
   items('VARIABLES2',all),
   items('PARTITIONS',all),
   items('PARTITIONS'\^p,all),
   vals(
     ['VARIABLES1'\^var],
     part('PARTITIONS'),
     =, dontcare, dontcare),
   vals(
     ['VARIABLES2'\^var],
     part('PARTITIONS'),
     =, dontcare, dontcare))).

ctr_graph(
  soft_used_by_partition_var,
  ['VARIABLES1','VARIABLES2'],
  2,
  ['PRODUCT'>>collection(variables1,variables2)],
  [in_same_partition(
    variables1\^var,
    variables2\^var,
    PARTITIONS)],
  ['NSINK_NSOURCE'=size('VARIABLES2')-'C'],
  []).
soft_used_by_partition_var_r(C, VARIABLES1, VARIABLES2, PARTITIONS) :-
  length(VARIABLES1, L1),
  length(VARIABLES2, L2),
  L1 >= L2,
  check_type(dvar(0, L2), C),
  collection(VARIABLES1, [dvar]),
  collection(VARIABLES2, [dvar]),
  collection(PARTITIONS, [col_len_gteq(1, [int])]),
  get_attr1(VARIABLES1, VARS1),
  get_attr1(VARIABLES2, VARS2),
  get_col_attr1(PARTITIONS, 1, PVALS),
  flatern(PVALS, VALS),
  all_different(VALS),
  length(PARTITIONS, M),
  M > 1,
  length(PVALS, LPVALS),
  get_partition_var(VARS1, PVALS, PVARS1, LPVALS, 0),
  get_partition_var(VARS2, PVALS, PVARS2, LPVALS, 0),
  gen_collection(PVARS1, var, CVARS1),
  gen_collection(PVARS2, var, CVARS2),
  eval(soft_used_by_var(C, CVARS1, CVARS2)).
B.333 soft_used_by_var

◊ **META-DATA:**

```
ctr_date(soft_used_by_var,[‘20050507’,’20060816’]).
ctr_origin(soft_used_by_var,‘Derived from %c’,[used_by]).
ctr_synonyms(soft_used_by_var,[soft_used_by]).
ctr_arguments(soft_used_by_var,[soft_used_by],
    [‘C’-dvar,
     ‘VARIABLES1’-collection(var-dvar),
     ‘VARIABLES2’-collection(var-dvar)]).
ctr_restrictions(soft_used_by_var,
    [‘C’>=0,
     ‘C’=<size(‘VARIABLES2’),
     size(‘VARIABLES1’)=size(‘VARIABLES2’),
     required(‘VARIABLES1’,var),
     required(‘VARIABLES2’,var)]).
ctr_example(soft_used_by_var,
    soft_used_by_var(2,
     [[var-9],[var-1],[var-1],[var-8],[var-8]],
     [[var-9],[var-9],[var-9],[var-1]])).
ctr_typical(soft_used_by_var,
    [‘C’>0,
     size(‘VARIABLES1’) >1,
     size(‘VARIABLES2’) >1,
     range(‘VARIABLES1’^var)>1,
     range(‘VARIABLES2’^var)>1]).
ctr_exchangeable(soft_used_by_var,
    [items(‘VARIABLES1’,all),
     items(‘VARIABLES2’,all),
     vals(
      [‘VARIABLES1’^var,’VARIABLES2’^var],
      int],
```
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

```prolog
=\=, all, dontcare)).

ctr_graph(
    soft_used_by_var, ['VARIABLES1','VARIABLES2'], 2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['NSINK_NSOURCE'=size('VARIABLES2')-'C'], []).

ctr_eval(soft_used_by_var,[reformulation(soft_used_by_var_r)]).

soft_used_by_var_r(C,VARIABLES1,VARIABLES2) :-
    length(VARIABLES1,L1),
    length(VARIABLES2,L2),
    L1>=L2,
    check_type(dvar(0,L2),C),
    collection(VARIABLES1,[dvar]),
    collection(VARIABLES2,[dvar]),
    get_attr1(VARIABLES2,VARS2),
    get_minimum(VARS2,MINVARS2),
    get_maximum(VARS2,MAXVARS2),
    soft_used_by_var1(MINVARS2, MAXVARS2, L1, OCCS1, OCCS2, TERM),
    eval(global_cardinality(VARIABLES1,OCCS1)),
    eval(global_cardinality(VARIABLES2,OCCS2)),
    call(C#=TERM).

soft_used_by_var1(I,S,_32157,[],[],0) :-
    I>S,
    !.

soft_used_by_var1( I, S, MAX, [[val-I,nocurrence-01]|R1],
    [[val-I,nocurrence-02]|R2],
```
max(O2-O1,0)+R) :-
   I=<S,
   O1 in 0..MAX,
   O2 in 0..MAX,
   I1 is I+1,
   soft_used_by_var1(I1,S,MAX,R1,R2,R).
B.334 some_equal

◊ **Meta-Data:**

```prolog
ctr_date(some_equal,['20110604']).
ctr_origin(some_equal,'Derived from %c',[alldifferent]).

ctr_synonyms(
    some_equal,
    [some_eq,
     not_alldifferent,
     not_alldiff,
     not_alldistinct,
     not_distinct]).

ctr_arguments(some_equal,['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    some_equal,
    [required('VARIABLES',var),size('VARIABLES')>1]).

ctr_example(
    some_equal,
    some_equal([[var-1],[var-4],[var-1],[var-6]])).

ctr_typical(some_equal,[size('VARIABLES')>2]).

ctr_exchangeable(
    some_equal,
    [items('VARIABLES',all),
     vals(['VARIABLES'~var],int,=\=,all,dontcare)]).

ctr_graph(
    some_equal,
    ['VARIABLES'],
    2,
    ['CLIQUE'(<)>>collection(variables1,variables2)],
    [variables1~var=variables2~var],
    ['NARC'>0],
    []).

ctr_eval(
    some_equal,
    [checker(some_equal_c),reformulation(some_equal_r)]).
```
ctr_extensible(some_equal,[],’VARIABLES’,any).

some_equal_c(VARIABLES) :-
    collection(VARIABLES,[int]),
    get_attr1(VARIABLES,VARS),
    sort(VARS,S),
    length(VARS,M),
    length(S,N),
    N<M.

some_equal_r(VARIABLES) :-
    collection(VARIABLES,[dvar]),
    get_attr1(VARIABLES,VARS),
    length(VARS,M),
    M>1,
    M1 is M-1,
    N in 1..M1,
    nvalue(N,VARS).
B.335  sort

◊ Meta-Data:

ctr_date(sort,['20030820','20060816']).

ctr_origin(sort,'\cite{OlderSwinkelsEmden95}',[]).

ctr_synonyms(sort,[sortedness,sorted,sorting]).

ctr_arguments(
sort,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
sort,
    [size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var)]).

ctr_example(
sort,
    sort(
        [[var-1],[var-9],[var-1],[var-5],[var-2],[var-1]],
        [[var-1],[var-1],[var-1],[var-2],[var-5],[var-9]]).

ctr_typical(
sort,
    [size('VARIABLES1')>1,range('VARIABLES1`var)>1]).

ctr_exchangeable(
sort,
    [items('VARIABLES1',all),
     translate(['VARIABLES1`var','VARIABLES2`var])].

ctr_graph(
sort,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT']>>collection(variables1,variables2),
    [variables1`var=variables2`var],
    [for_all('CC','NSOURCE'='NSINK'),
     'NSOURCE'=size('VARIABLES1'),
     'NSINK'=size('VARIABLES2')],
    []).
ctr_graph(
    sort,
    ['VARIABLES2'],
    2,
    ['PATH'>>collection(variables1,variables2)],
    [variables1\^var=<variables2\^var],
    ['NARC'=size('VARIABLES2')-1],
    []).

ctr_eval(sort,[reformulation(sort_r)]).

ctr_pure_functional_dependency(sort,[]).

ctr_functional_dependency(sort,2,[1]).

sort_r(VARIABLES1,VARIABLES2) :-
    collection(VARIABLES1,[dvar]),
    collection(VARIABLES2,[dvar]),
    length(VARIABLES1,L1),
    length(VARIABLES2,L2),
    L1=L2,
    get_attr1(VARIABLES1,VARS1),
    get_attr1(VARIABLES2,VARS2),
    length(P,L1),
    domain(P,1,L1),
    sorting(VARS1,P,VARS2).
B.336 sort_permutation

◊ Meta-Data:

ctr_date(sort_permutation,['20030820','20060816','20111025']).

ctr_origin(sort_permutation,'\cite{Zhou97}',[]).

ctr_usual_name(sort_permutation,sort).

ctr_synonyms(sort_permutation,sort_permutation,[extended_sortedness,sortedness,sorted,sorting]).

ctr_arguments(sort_permutation,'FROM'-collection(var-dvar),
 'PERMUTATION'-collection(var-dvar),
 'TO'-collection(var-dvar)).

ctr_restrictions(sort_permutation,
[size('PERMUTATION')=size('FROM'),
 size('PERMUTATION')=size('TO'),
 'PERMUTATION' ^var>=1,
 'PERMUTATION' ^var=<size('PERMUTATION'),
 alldifferent('PERMUTATION'),
 required('FROM',var),
 required('PERMUTATION',var),
 required('TO',var)].

ctr_example(sort_permutation,
 sort_permutation([ [var-1],[var-9],[var-1],[var-5],[var-2],[var-1]],
 [ [var-1],[var-6],[var-3],[var-5],[var-4],[var-2]],
 [ [var-1],[var-1],[var-1],[var-2],[var-5],[var-9]])].

ctr_typical(sort_permutation,
 [size('FROM')>1,
 range('FROM' ^var)>1,
 lex_different('FROM','TO')]).

ctr_exchangeable(sort_permutation,
[translate([FROM\textasciicircum{}var,TO\textasciicircum{}var])].

ctr\_derived\_collections(
  sort\_permutation,
  [col('FROM\_PERMUTATION'-collection(var-dvar,ind-dvar),
    [item(var='FROM\textasciicircum{}var,ind='PERMUTATION\textasciicircum{}var)])]).

ctr\_graph(
  sort\_permutation,
  ['FROM\_PERMUTATION','TO'],
  2,
  ['PRODUCT'\textasciicircum{}collection(from\_permutation,to)],
  [from\_permutation\textasciicircum{}var=to\textasciicircum{}var,from\_permutation\textasciicircum{}ind=to\textasciicircum{}key],
  ['NARC'=size('PERMUTATION')],
  []).

ctr\_graph(
  sort\_permutation,
  ['TO'],
  2,
  ['PATH'\textasciicircum{}collection(to1,to2)],
  [to1\textasciicircum{}var=<to2\textasciicircum{}var],
  ['NARC'=size('TO')-1],
  []).

ctr\_eval(sort\_permutation,[builtin(sort\_permutation\_b)]).

ctr\_functional\_dependency(sort\_permutation,3,[1]).

ctr\_functional\_dependency(sort\_permutation,2,[1,3]).

sort\_permutation\_b(FROM,PERMUTATION,TO) :-
  length(FROM,F),
  length(PERMUTATION,P),
  length(TO,T),
  F=P,
  P=T,
  collection(FROM,[dvar]),
  collection(PERMUTATION,[dvar(1,P)]),
  collection(TO,[dvar]),
  get\_attrl(FROM,FVARS),
  get\_attrl(PERMUTATION,PVARS),
  get\_attrl(TO,TVARS),
  sorting(FVARS,PVARS,TVARS).
B.337  stable compatibility

◊ Meta-Data:

\texttt{ctr\_date(stable\_compatibility,\[\textquote{20070601}\]).}

\texttt{ctr\_origin(}
\begin{itemize}
  \item stable\_compatibility, P.\textquote{^\textregistered}Flener, \cite{BeldiceanuFlenerLorca06}, []).
\end{itemize}

\texttt{ctr\_arguments(}
\begin{itemize}
  \item stable\_compatibility, \[\text{NODES-}
    \text{-collection(index-int,father-dvar,prec-sint,inc-sint)\].}
\end{itemize}

\texttt{ctr\_restrictions(}
\begin{itemize}
  \item stable\_compatibility, required\(\text{\textquote{\textquotesingle NODES\textquotesingle},\[\text{index,father,prec,inc}\]}, \text{\textquotesingle NODES\textquotesingle}^{\text{\textquote{\textquotesingle index\textquotesingle}}}\geq 1, \text{\textquotesingle NODES\textquotesingle}^{\text{\textquote{\textquotesingle index\textquotesingle}}}\leq \text{\textquotesingle size\textquotesingle (\text{\textquotesingle NODES\textquotesingle)}}, \text{\textquotesingle NODES\textquotesingle}^{\text{\textquote{\textquotesingle father\textquotesingle}}}\geq 1, \text{\textquotesingle NODES\textquotesingle}^{\text{\textquote{\textquotesingle father\textquotesingle}}}\leq \text{\textquotesingle size\textquotesingle (\text{\textquotesingle NODES\textquotesingle)}}, \text{\textquotesingle NODES\textquotesingle}^{\text{\textquote{\textquotesingle prec\textquotesingle}}}\geq 1, \text{\textquotesingle NODES\textquotesingle}^{\text{\textquote{\textquotesingle prec\textquotesingle}}}\leq \text{\textquotesingle size\textquotesingle (\text{\textquotesingle NODES\textquotesingle)}}, \text{\textquotesingle NODES\textquotesingle}^{\text{\textquote{\textquotesingle inc\textquotesingle}}}\geq 1, \text{\textquotesingle NODES\textquotesingle}^{\text{\textquote{\textquotesingle inc\textquotesingle}}}\leq \text{\textquotesingle size\textquotesingle (\text{\textquotesingle NODES\textquotesingle)}}, \text{\textquotesingle NODES\textquotesingle}^{\text{\textquote{\textquotesingle inc\textquotesingle}}}\textgreater \text{\textquotesingle NODES\textquotesingle}^{\text{\textquote{\textquotesingle index\textquotesingle}}}).
\end{itemize}

\texttt{ctr\_example(}
\begin{itemize}
  \item stable\_compatibility, stable\_compatibility(}
  \begin{itemize}
  \item [[\text{\textquote{\textquotesingle index\textquotesingle}=1, \text{\textquote{\textquotesingle father\textquotesingle}=4, \text{\textquotesingle prec\textquotesingle}={11,12}, \text{\textquote{\textquotesingle inc\textquotesingle}={}}]], \text{\textquote{\textquotesingle index\textquotesingle}=2, \text{\textquote{\textquotesingle father\textquotesingle}=3, \text{\textquotesingle prec\textquotesingle}={8,9}, \text{\textquote{\textquotesingle inc\textquotesingle}={}}]], \text{\textquote{\textquotesingle index\textquotesingle}=3, \text{\textquote{\textquotesingle father\textquotesingle}=4, \text{\textquotesingle prec\textquotesingle}={2,10}, \text{\textquote{\textquotesingle inc\textquotesingle}={}}]], \text{\textquote{\textquotesingle index\textquotesingle}=4, \text{\textquote{\textquotesingle father\textquotesingle}=5, \text{\textquotesingle prec\textquotesingle}={1,3}, \text{\textquote{\textquotesingle inc\textquotesingle}={}}]], \text{\textquote{\textquotesingle index\textquotesingle}=5, \text{\textquote{\textquotesingle father\textquotesingle}=7, \text{\textquotesingle prec\textquotesingle}={4,13}, \text{\textquote{\textquotesingle inc\textquotesingle}={}}]], \text{\textquote{\textquotesingle index\textquotesingle}=6, \text{\textquote{\textquotesingle father\textquotesingle}=2, \text{\textquotesingle prec\textquotesingle}={8,14}, \text{\textquote{\textquotesingle inc\textquotesingle}={}}]], \text{\textquote{\textquotesingle index\textquotesingle}=7, \text{\textquote{\textquotesingle father\textquotesingle}=7, \text{\textquotesingle prec\textquotesingle}={6,13}, \text{\textquote{\textquotesingle inc\textquotesingle}={}}]], \text{\textquote{\textquotesingle index\textquotesingle}=8, \text{\textquote{\textquotesingle father\textquotesingle}=6, \text{\textquotesingle prec\textquotesingle}={}, \text{\textquote{\textquotesingle inc\textquotesingle}={9,10,11,12,13,14}}]], \text{\textquote{\textquotesingle index\textquotesingle}=9, \text{\textquote{\textquotesingle father\textquotesingle}=2, \text{\textquote{\textquotesingle prec\textquotesingle}={}}, \text{\textquote{\textquotesingle inc\textquotesingle}={10,11,12,13}}]], \text{\textquote{\textquotesingle index\textquotesingle}=10, \text{\textquote{\textquotesingle father\textquotesingle}=3, \text{\textquote{\textquotesingle prec\textquotesingle}={11,12,13}}]], \text{\textquote{\textquotesingle index\textquotesingle}=11, \text{\textquote{\textquotesingle father\textquotesingle}=1, \text{\textquote{\textquotesingle prec\textquotesingle}={}}, \text{\textquote{\textquotesingle inc\textquotesingle}={12,13}}]], \text{\textquote{\textquotesingle index\textquotesingle}=12, \text{\textquote{\textquotesingle father\textquotesingle}=1, \text{\textquote{\textquotesingle prec\textquotesingle}={}}, \text{\textquote{\textquotesingle inc\textquotesingle}={13}}]], \text{\textquote{\textquotesingle index\textquotesingle}=13, \text{\textquote{\textquotesingle father\textquotesingle}=5, \text{\textquote{\textquotesingle prec\textquotesingle}={}}, \text{\textquote{\textquotesingle inc\textquotesingle}={14}}]],}
\end{itemize}
\end{itemize}
\end{itemize}
[index-14,father-6,prec-{},inc-{}])

ctr_typical(
    stable_compatibility,
    [size('NODES')>2,range('NODES'\father)>1]).

ctr_exchangeable(stable_compatibility,[items('NODES',all)]).

ctr_graph(
    stable_compatibility,
    ['NODES'],
    2,
    ['CLIQUE'\collection(nodes1,nodes2)],
    [nodes1\father=nodes2\index],
    ['MAX_NSNC'=<1,
     'NCC'=1,
     'MAX_ID'=<2,
     'PATH_FROM_TO'(index,index,prec)=1,
     'PATH_FROM_TO'(index,index,inc)=0,
     'PATH_FROM_TO'(index,inc,index)=0],
    []).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.338 stage_element

Meta-Data:

ctr_date(stage_element, [‘20040828’, ‘20060816’]).

ctr_origin(
    stage_element,
    \index{Choco|indexuse}Choco, derived from %c.,
    [element]).

ctr_usual_name(stage_element, stage_elt).

ctr_synonyms(stage_element, [stage_elem]).

ctr_arguments(
    stage_element,
    [‘ITEM’-collection(index-dvar, value-dvar),
     ‘TABLE’-collection(low-int, up-int, value-int)]).

ctr_restrictions(
    stage_element,
    [required(‘ITEM’, [index, value]),
     size(‘ITEM’) = 1,
     size(‘TABLE’) > 0,
     required(‘TABLE’, [low, up, value]),
     ‘TABLE’^low =< ‘TABLE’^up,
     increasing_seq(‘TABLE’, [low])]).

ctr_example(
    stage_element,
    stage_element(
        [[index-5, value-6]],
        [[low-3, up-7, value-6],
         [low-8, up-8, value-9],
         [low-9, up-14, value-2],
         [low-15, up-19, value-9]])).

ctr_typical(
    stage_element,
    [size(‘TABLE’) > 1,
     range(‘TABLE’^value) > 1,
     ‘TABLE’^low =< ‘TABLE’^up]).

ctr_exchangeable(
    stage_element,
vals([ITEM\^value, TABLE\^value], int, =\=, all, dontcare)).

ctr_graph(
    stage_element,
    ['TABLE'],
    2,
    ['PATH'] >> collection(table1, table2),
    [table1\^low =< table1\^up,
     table1\^up + 1 = table2\^low,
     table2\^low =< table2\^up],
    ['NARC' = size('TABLE') - 1],
    []).

ctr_graph(
    stage_element,
    ['ITEM', 'TABLE'],
    2,
    ['PRODUCT'] >> collection(item, table),
    [item\^index = = table\^low,
     item\^index =< table\^up,
     item\^value = table\^value],
    ['NARC' = 1],
    []).

ctr_eval(stage_element, [automaton(stage_element_a)]).

ctr_pure_functional_dependency(stage_element, []).

ctr_functional_dependency(stage_element, 1-2, [1-1, 2]).

ctr_extensible(stage_element, [], 'TABLE', suffix).

stage_element_a(FLAG, ITEM, TABLE) :-
    collection(ITEM, [dvar, dvar]),
    collection(TABLE, [int, int, int]),
    length(TABLE, N),
    N>0,
    get_attr1(TABLE, LOWS),
    get_attr2(TABLE, UPS),
    check_lesseq(LOWS, UPS),
    collection_increasing_seq(TABLE, [1]),
    ITEM=[[index-ITEM_INDEX, value-ITEM_VALUE]],
    stage_element_signature(
        TABLE,
        SIGNATURE,
        ITEM_INDEX,
ITEM_VALUE),
AUTOMATON=
automaton(
  SIGNATURE,
_38249,
SIGNATURE,
[source(s),sink(t)],
[arc(s,0,s),arc(s,1,t),arc(t,0,t),arc(t,1,t)],
[],
[],
[]),
automaton_bool(FLAG,[0,1],AUTOMATON).

stage_element_signature([],[],_35286,_35287).

stage_element_signature(
  [[low-TABLE_LOW,up-TABLE_UP,value-TABLE_VALUE]|TABLEs],
  [S|Ss],
  ITEM_INDEX,
  ITEM_VALUE) :-
  TABLE_LOW#=ITEM_INDEX#/\ITEM_INDEX#=<TABLE_UP#/\ITEM_VALUE#=TABLE_VALUE#><=>S,
  stage_element_signature( TABLEs, Ss, ITEM_INDEX, ITEM_VALUE).
B.339  stretch_circuit

◊ **META-DATA:**

ctr_date(stretch_circuit,[’20030820’,’20060817’,’20090716’]).

ctr_origin(stretch_circuit,’\cite{Pesant01},[ ]).

ctr_usual_name(stretch_circuit,stretch).

ctr_arguments(stretch_circuit,
   [’VARIABLES’-collection(var-dvar),
    ’VALUES’-collection(val-int,lmin-int,lmax-int)]).

ctr_restrictions(stretch_circuit,
   [size(’VARIABLES’) > 0,
    required(’VARIABLES’,var),
    size(’VALUES’) > 0,
    required(’VALUES’,[val,lmin,lmax]),
    distinct(’VALUES’,val),
    ’VALUES’ˆlmin=<’VALUES’ˆlmax,
    ’VALUES’ˆlmin=<size(’VARIABLES’),
    sum(’VALUES’ˆlmin)=<size(’VARIABLES’))].

ctr_example(stretch_circuit,
   stretch_circuit(
      [[var-6],
       [var-6],
       [var-3],
       [var-1],
       [var-1],
       [var-1],
       [var-6],
       [var-6]],
      [[val-1,lmin-2,lmax-4],
       [val-2,lmin-2,lmax-3],
       [val-3,lmin-1,lmax-6],
       [val-6,lmin-2,lmax-4]]).

ctr_typical(stretch_circuit,
   [size(’VARIABLES’) > 1,
    range(’VARIABLES’ˆvar)>1,
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

\[
\begin{align*}
&\text{size('VARIABLES') > size('VALUES'),} \\
&\text{size('VALUES') > 1,} \\
&\text{'VALUES' \^ lmax =< size('VARIABLES')).}
\end{align*}
\]

\[
\text{ctr\textsubscript{exchangeable}(} \\
\text{stretch\_circuit,} \\
[\text{items('VARIABLES',shift),} \\
\text{items('VALUES',all),} \\
\text{vals(} \\
[\text{'VARIABLES' \^ var,'VALUES' \^ val}, \\
\text{int,} \\
\text{=}\neq,} \\
\text{all,} \\
\text{dontcare)])].}
\]

\[
\text{ctr\textsubscript{graph}(} \\
\text{stretch\_circuit,} \\
[\text{'VARIABLES'],} \\
2,} \\
\text{foreach(} \\
\text{VALUES,} \\
[\text{'CIRCUIT'\^ collection(variables1,variables2),} \\
\text{'LOOP'\^ collection(variables1,variables2))},} \\
[\text{variables1\^ var='VALUES'\^ val,variables2\^ var='VALUES'\^ val},} \\
[\text{not\_in('MIN\_NCC',1,'VALUES'\^ lmin-1),} \\
\text{'MAX\_NCC' =< 'VALUES'\^ lmax}],} \\
[]].
\]

\[
\text{ctr\textsubscript{eval}(stretch\_circuit,[reformulation(stretch\_circuit\_r)])}.}
\]

\[
\text{stretch\_circuit\_r(VARIABLES,VALUES) :-} \\
\text{collection(VARIABLES,[dvar]),} \\
\text{collection(VALUES,[int,int,int]),} \\
\text{length(VARIABLES,N),} \\
\text{stretch\_circuit1(VALUES,0,N,DELTA),} \\
\text{prefix\_length(VARIABLES,VARS\_DELTA,DELTA),} \\
\text{append(VARIABLES,VARS\_DELTA,VARS),} \\
\text{ND is N+DELTA,} \\
\text{stretch\_circuit2(VALUES,N,ND,VALS),} \\
\text{eval(stretch\_path(VARS,VALS)).}
\]

\[
\text{stretch\_circuit1([],C,N,DELTA) :-} \\
\text{DELTA is min(C,N).}
\]

\[
\text{stretch\_circuit1([[_42300,_42302,_42307-1]|R],C,N,DELTA) :-} \\
\text{M is max(L,C),}
\]
stretch_circuit1(R,M,N,DELTA).

stretch_circuit2([],_42292,_42293,[]).

\[
\text{stretch\_circuit2}([\,[A,B,1\_max-L]\mid R],N,ND,[[A,B,1\_max-LL]\mid S]) \leftarrow \\
( \begin{array}{l}
L\geq N \rightarrow \\
\quad LL=ND \\
L < N \\
\quad LL=L
\end{array} ).
\]

\]

stretch_circuit2(R,N,ND,S).
**B.340  stretch_path**

◊ **Meta-Data:**

```prolog
ctr_date(stretch_path, ['20030820', '20060817', '20090712']).

ctr_origin(stretch_path, '\cite{Pesant01}', []).

ctr_usual_name(stretch_path, stretch).

ctr_arguments(
    stretch_path, 
    ['VARIABLES' - collection(var-dvar), 
     'VALUES' - collection(val-int, lmin-int, lmax-int)]).

ctr_restrictions(
    stretch_path, 
    [size('VARIABLES') > 0, 
     required('VARIABLES', var), 
     size('VALUES') > 0, 
     required('VALUES', [val, lmin, lmax]), 
     distinct('VALUES', val), 
     'VALUES' `lmin >= 0, 
     'VALUES' `lmin =<' VALUES' `lmax, 
     'VALUES' `lmin =< size('VARIABLES')]).

ctr_example(
    stretch_path, 
    stretch_path(
        [[var-6], 
         [var-6], 
         [var-3], 
         [var-1], 
         [var-1], 
         [var-1], 
         [var-6], 
         [var-6]], 
        [[val-1, lmin-2, lmax-4], 
         [val-2, lmin-2, lmax-3], 
         [val-3, lmin-1, lmax-6], 
         [val-6, lmin-2, lmax-2]])).

ctr_typical(
    stretch_path, 
    [size('VARIABLES') > 1, 
     range('VARIABLES' `var) > 1, 
     ...])
```
size('VARIABLES') > size('VALUES'),
size('VALUES') > 1,
sum('VALUES' \textasciicircum lmin) \leq size('VARIABLES'),
'VALUES' \textasciicircum lmax \leq size('VARIABLES')).

\texttt{ctr\_exchangeable(}
\texttt{stretch\_path,}
\texttt{[items('VARIABLES',reverse),}
\texttt{items('VALUES',all),}
\texttt{vals(
  ['VARIABLES' \textasciicircum var,'VALUES' \textasciicircum val],
  int,
  =\textasciimacron,
  all,
  dontcare))].

\texttt{ctr\_graph(}
\texttt{stretch\_path,}
\texttt{['VARIABLES'],}
\texttt{2,}
\texttt{foreach(}
\texttt{VALUES,}
\texttt{['PATH' \textasciicircum collection(variables1,variables2),}
\texttt{'LOOP' \textasciicircum collection(variables1,variables2)]},
\texttt{[variables1 \textasciicircum var='VALUES' \textasciicircum val,variables2 \textasciicircum var='VALUES' \textasciicircum val],
[not\_in('MIN\_NCC',i,'VALUES' \textasciicircum lmin-1),
'MAX\_NCC' \textasciicircum =<'VALUES' \textasciicircum lmax],
[]).}

\texttt{ctr\_eval(stretch\_path, [automaton(stretch\_path\_a)])}.

\texttt{stretch\_path\_a(FLAG,VARIABLES,VALUES) :-}
\texttt{stretch\_path\_get\_a(VARIABLES,VALUES,AUTOMATON,ALPHABET),}
\texttt{automaton\_bool(FLAG,ALPHABET,AUTOMATON).}

\texttt{stretch\_path\_get\_a(VARIABLES,VALUES,AUTOMATON,ALPHABET) :-}
\texttt{length(VARIABLES,N),}
\texttt{N>0,}
\texttt{collection(VARIABLES,[dvar]),}
\texttt{collection(VALUES,[int,int(0,N),int]),}
\texttt{get\_attr1(VARIABLES,VARS),}
\texttt{get\_attr1(VALUES,VALS),}
\texttt{get\_attr2(VALUES,LMINS),}
\texttt{get\_attr3(VALUES,LMAXS),}
\texttt{length(VALS,M),}
\texttt{M>0,}
all_different(VALS),
check_leqseq(LMINS,LMAXS),
stretch_lmin(LMINS,LMINS1),
stretch_reduce_lmax(LMAXS,N,LMAXSR),
stretch_gen_states(LMINS1,LMAXSR,N,1,STATES),
stretch_gen_transitions(
  1,
  M,
  LMINS1,
  LMAXSR,
  LMINS1,
  LMAXSR,
  N,
  TRANSITIONS),
get_minimum(VARS,MINVARS),
get_maximum(VARS,MAXVARS),
sort(VALS,SVALS),
SVALS=[MINVARS|46577],
last(SVALS,MAXVARS),
VALS_RANGE is MAXVARS-MINVARS+1,
( VALS_RANGE=M,
  MINVARS=<MINVARS,
  MAXVARS=<MAXVARS ->
stretch_path_simplify_transitions(
  TRANSITIONS,
  MINVALS,
  SIMPLIFIED_TRANSITIONS),
AUTOMATON=
automaton(
  VARS,
  _52619,
  VARS,
  STATES,
  SIMPLIFIED_TRANSITIONS,
  [],
  [],
  []),
SIG in MINVARS..MAXVARS
); stretch_path_signature(VARS,VALS,M,SIGNALATURE),
AUTOMATON=
automaton(
  SIGNALATURE,
  _54066,
  SIGNALATURE,
  STATES,
  TRANSITIONS,
SIG in 0..M
union_dom_list_int([SIG], ALPHABET).

stretch_path_simplify_transitions([], 46399, []).

stretch_path_simplify_transitions( [arc(_46404, 0, _46406) | R], MINVALS, S) :-
  !,
stretch_path_simplify_transitions(R, MINVALS, S).

stretch_path_simplify_transitions( [arc(Si, E, Sj) | R], MINVALS, [arc(Si, NE, Sj) | S]) :-
  NE is MINVALS + E - 1,
stretch_path_simplify_transitions(R, MINVALS, S).

stretch_path_signature([], 46396, 46397, []).

stretch_path_signature([VAR | VARs], VALS, M, [S | Ss]) :-
  S in 0..M,
stretch_path_signature1(VALS, VALS, VAR, 1, S),
stretch_path_signature(VARs, VALS, M, Ss).

stretch_path_signature1([], 46396, 46397, []).

stretch_path_signature1([VAL | VALs], VAR, VAR# = VAL# / R) :-
  VAR# = VAL# / R => S = I,
  I1 is I + 1,
stretch_path_signature1(VALs, VALS, VAR, I1, S).

stretch_path_signature2([], 46396, 1).

stretch_path_signature2([VAL | VALs], VAR, VAR# = VAL#/ R) :-
  stretch_path_signature2(VALs, VAR, R).
B.341 stretch_path_partition

◊ **META-DATA:**

```
ctr_date(stretch_path_partition, ['20091106']).

ctr_origin(
    stretch_path_partition,
    Derived from %c.,
    [stretch_path]).

ctr_synonyms(stretch_path_partition, [stretch]).

ctr_types(
    stretch_path_partition,
    ['VALUES'-collection(val-int)]).

ctr_arguments(
    stretch_path_partition,
    ['VARIABLES'-collection(var-dvar),
     'PARTLIMITS'-collection(p-VALUES, lmin-int, lmax-int)]).

ctr_restrictions(
    stretch_path_partition,
    [size('VALUES')>=1,
     required('VALUES',val),
     distinct('VALUES',val),
     size('VARIABLES')>0,
     required('VARIABLES', var),
     size('PARTLIMITS')>0,
     required('PARTLIMITS', [p, lmin, lmax]),
     'PARTLIMITS'\lmin>=0,
     'PARTLIMITS'\lmin=<'PARTLIMITS'\lmax,
     'PARTLIMITS'\lmin=<size('VARIABLES'))].

ctr_example(
    stretch_path_partition,
    stretch_path_partition(  
        [[var-1],
         [var-2],
         [var-0],
         [var-0],
         [var-2],
         [var-2],
         [var-2],
         [var-0]],
    
```
[[p-[[val-1],[val-2]],lmin-2,lmax-4],
[p-[[val-3]],lmin-0,lmax-2]]).

ctr_typical(
    stretch_path_partition,
    [size('VARIABLES')>1,
     range('VARIABLES'\var)>1,
     size('VARIABLES')>size('PARTLIMITS'),
     size('PARTLIMITS')>1,
     sum('PARTLIMITS'\lmin)<size('VARIABLES'),
     'PARTLIMITS'\lmax<=size('VARIABLES')]).

ctr_exchangeable(
    stretch_path_partition,
    [items('VARIABLES',reverse),
     items('PARTLIMITS',all),
     items('PARTLIMITS'\p,all),
     vals(
         ['VARIABLES'\var,'PARTLIMITS'\p\val],
         int,
         =\=,
         all,
         dontcare))).

ctr_eval(
    stretch_path_partition,
    [reformulation(stretch_path_partition_r),
     automaton(stretch_path_partition_a)]).

stretch_path_partition_r(VARIABLES,PARTLIMITS) :-
    length(VARIABLES,N),
    N>0,
    collection(VARIABLES,[dvar]),
    collection(PARTLIMITS,
        [col_len_gteq(1,[int]),int(0,N),int]),
    get_attr1(VARIABLES,VARS),
    get_col_attr1(PARTLIMITS,1,PVALS),
    get_attr2(PARTLIMITS,LMINS),
    get_attr3(PARTLIMITS,LMAXS),
    length(PVALS,M),
    M>0,
    check_lesseq(LMINS,LMAXS),
    flattern(PVALS,VALS),
    all_different(VALS),
    get_partition_var(VARS,PVALS,PVARS),
    ...
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

```prolog
gen_collection(PVARS, var, PVARIABLES),
stretch_path_partition_values(PARTLIMITS, 1, VALUES),
eval(stretch_path(PVARIABLES, VALUES)).
```

```prolog
stretch_path_partition_values([], _, _20734, []) :- !.
```

```prolog
stretch_path_partition_values([|_20738, lmin-LMIN, lmax-LMAX| R],
V,
[|val-V, lmin-LMIN, lmax-LMAX| S]) :-
V1 is V+1,
stretch_path_partition_values(R, V1, S).
```

```prolog
stretch_path_partition_a(FLAG, VARIABLES, PARTLIMITS) :-
stretch_path_partition_get_a(VARIABLES,
PARTLIMITS,
AUTOMATON,
ALPHABET),
automaton_bool(FLAG, ALPHABET, AUTOMATON).
```

```prolog
stretch_path_partition_get_a(VARIABLES, PARTLIMITS, AUTOMATON, ALPHABET) :-
length(VARIABLES, N),
N>0,
collection(VARIABLES, [dvar]),
collection(PARTLIMITS, [col_len_gteq(1, [int]), int(0, N), int]),
get_attr1(VARIABLES, VARS),
get_col_attr1(PARTLIMITS, 1, PVALS),
get_attr2(PARTLIMITS, LMINS),
get_attr3(PARTLIMITS, LMAXS),
length(PVALS, M),
M>0,
check_lesseq(LMINS, LMAXS),
flattern(PVALS, VALS),
all_different(VALS),
stretch_lmin(LMINS, LMINS1),
stretch_reduce_lmax(LMAXS, N, LMAXSR),
stretch_gen_states(LMINS1, LMAXSR, N, 1, STATES),
stretch_gen_transitions(1)
```

```prolog
```
```
1, M, LMIN1, LMAXSR, LMIN1, LMAXSR, N, TRANSITIONS), get_minimum(VARS, MINVARS), get_maximum(VARS, MAXVARS), sort(VALS, SVALS), SVALS=[MINVALS|_20926], last(SVALS, MAXVALS), VALS_RANGE is MAXVALS-MINVALS+1, ( VALS_RANGE=M, MINVALS=<MINVARS, MAXVARS=<MAXVALS -> COMP_VALS=[]; stretch_path_partition_complement( MINVARS, MAXVARS, VALS, COMP_VALS) ), stretch_path_partition_expand_transitions( TRANSITIONS, COMP_VALS, PVALS, EXPANDED_TRANSITIONS), AUTOMATON=automaton( VARS, _28426, VARS, STATES, EXPANDED_TRANSITIONS, [], [], []), append(VARS, SVALS, ALL_VALS), union_dom_list_int(ALL_VALS, ALPHABET).

stretch_path_partition_complement(MIN, MAX, _20735,[]) :- MIN>MAX, !.
stretch_path_partition_complement(MIN, MAX, VALS, C) :-
    member(MIN, VALS),
    !,
    MIN1 is MIN+1,
    stretch_path_partition_complement(MIN1, MAX, VALS, C).

stretch_path_partition_complement(MIN, MAX, VALS, [MIN|C]) :-
    MIN1 is MIN+1,
    stretch_path_partition_complement(MIN1, MAX, VALS, C).

stretch_path_partition_expand_transitions([], _, _, []) :- !.

stretch_path_partition_expand_transitions(
    [arc(_, 0, _)|R], [], PVALS, S) :-
    !,
    stretch_path_partition_expand_transitions(R, [], PVALS, S).

stretch_path_partition_expand_transitions(
    [arc(Si, 0, Sj)|R], [CV|CR], PVALS, TS) :-
    !,
    stretch_path_partition_tr([CV|CR], arc(Si, 0, Sj), T),
    stretch_path_partition_expand_transitions(R, [CV|CR], PVALS, S),
    append(T, S, TS).

stretch_path_partition_expand_transitions(
    [arc(Si, E, Sj)|R], CL, PVALS, TS) :-
    nth1(E, PVALS, VALS),
    stretch_path_partition_tr(VALS, arc(Si, E, Sj), T),
    stretch_path_partition_expand_transitions(R, CL, PVALS, S),
    append(T, S, TS).

stretch_path_partition_tr([], _).
stretch_path_partition_tr(
    [VAL|R],
    arc(Si,E,Sj),
    [arc(Si,VAL,Sj)|S]) :-
    stretch_path_partition_tr(R,arc(Si,E,Sj),S).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.342  \texttt{strict\_lex2}

\textbf{Meta-Data:}

\begin{verbatim}
ctr_predefined(strict\_lex2).
ctr_date(strict\_lex2,[‘20031016’,‘20060817’]).
ctr_origin(
  strict\_lex2,
  \cite{FlenerFrischHnichKiziltanMiguelPearsonWalsh02}, []).
ctr_types(strict\_lex2,[‘VECTOR’-collection(var-dvar)]).
ctr_arguments(strict\_lex2,[‘MATRIX’-collection(vec-‘VECTOR’)]).
ctr_restrictions(
  strict\_lex2,
  \[size(‘VECTOR’)\geq1,
  \text{required(‘VECTOR’,var)},
  \text{required(‘MATRIX’,vec)},
  \text{same\_size(‘MATRIX’,vec)}\]).
ctr_example(
  strict\_lex2,
  strict\_lex2(
    \[[vec-[[var-2],[var-2],[var-3]]],
    [vec-[[var-2],[var-3],[var-1]]]]).
ctr_typical(strict\_lex2,[size(‘VECTOR’)>1,size(‘MATRIX’)>1]).
ctr_exchangeable(strict\_lex2,[translate([‘MATRIX’\^vec\^var])]).
ctr_eval(strict\_lex2,[reformulation(strict\_lex2\_r)]).
\end{verbatim}

\begin{verbatim}
strict\_lex2\_r(MATRIX) :-
  \text{collection}(MATRIX,[col([dvar])] ),
  \text{same\_size}(MATRIX),
  \text{get\_attr11}(MATRIX,MAT),
  \text{lex\_chain}(MAT,[op(#<)]),
  \text{transpose}(MAT,TMAT),
  \text{lex\_chain}(TMAT,[op(#<)]).
\end{verbatim}
B.343  strictly_decreasing

◊ META-DATA:

ctr_date(strictly_decreasing,[’20040814’,’20060817’]).

ctr_origin(
    strictly_decreasing,
    Derived from %c.,
    [strictly_increasing]).

ctr_arguments(
    strictly_decreasing,
    [’VARIABLES’-collection(var-dvar)]).

ctr_restrictions(
    strictly_decreasing,
    [required(’VARIABLES’,var)]).

ctr_example(
    strictly_decreasing,
    strictly_decreasing([[var-8],[var-4],[var-3],[var-1]])).

ctr_typical(strictly_decreasing,[size(’VARIABLES’) > 2]).

ctr_exchangeable(
    strictly_decreasing,
    [translate([’VARIABLES’^var])]).

ctr_graph(
    strictly_decreasing,
    [’VARIABLES’],
    2,
    [’PATH’>>collection(variables1,variables2)],
    [variables1^var>variables2^var],
    [’NARC’=size(’VARIABLES’)-1],
    []).

ctr_eval(
    strictly_decreasing,
    [checker(strictly_decreasing_c),
    automaton(strictly_decreasing_a)]).

ctr_contractible(strictly_decreasing,[],’VARIABLES’,any).

strictly_decreasing_c([]) :-
strictly_decreasing_c(VARIABLES) :-
collection(VARIABLES,[int]),
get_attr1(VARIABLES,VARS),
strictly_decreasing_c1(VARS).

strictly_decreasing_c1([]) :- !.
strictly_decreasing_c1([_27417]) :- !.
strictly_decreasing_c1([X,Y|R]) :-
  X>Y,
  strictly_decreasing_c1([Y|R]).

strictly_decreasing_a(1,[]) :- !.
strictly_decreasing_a(0,[]) :- !, fail.

strictly_decreasing_a(FLAG,VARIABLES) :-
collection(VARIABLES,[dvar]),
strictly_decreasing_signature(VARIABLES,SIGNATURE),
AUTOMATON= automaton(
  SIGNATURE,
  _28548,
  SIGNATURE,
  [source(s),sink(s)],
  [arc(s,0,s)],
  [],
  [],
  []),
automaton_bool(FLAG,[0,1],AUTOMATON).

strictly_decreasing_signature([_27418],[]) :- !.

strictly_decreasing_signature(
  [[var-VAR1],[var-VAR2]|VARS],
  [S|Ss]) :- S in 0..1,
VAR1#=<VAR2#<<=>S,
strictly_decreasing_signature([VAR-VAR2]|VARs],Ss).
B.344  strictly increasing

◊ **Meta-Data:**

ctr_date(strictly_increasing,[‘20040814’,’20060817’]).

ctr_origin(strictly_increasing,’KOALOG’,[]).

ctr_arguments(
  strictly_increasing,
  [‘VARIABLES’-collection(var-dvar)]).

ctr_restrictions(
  strictly_increasing,
  [required(‘VARIABLES’),var])).

ctr_example(
  strictly_increasing,
  strictly_increasing([[var-1],[var-3],[var-6],[var-8]]))

ctr_typical(strictly_increasing,[size(‘VARIABLES’) >2])

ctr_exchangeable(
  strictly_increasing,
  [translate([‘VARIABLES’¨var])])

ctr_graph(
  strictly_increasing,
  ['VARIABLES'],
  2,
  ['PATH’>>collection(variables1,variables2)],
  [variables1¨var<variables2¨var],
  ['NARC'=size(‘VARIABLES’)–1],
  []).

ctr_eval(
  strictly_increasing,
  [checker(strictly_increasing_c),
   automaton(strictly_increasing_a)])

ctr_contractible(strictly_increasing,[],‘VARIABLES’,any).

strictly_increasing_c([]) :- !.

strictly_increasing_c(VARIABLES) :- !.
collection(VARIABLES,[int]),
get_attr1(VARIABLES,VARS),
strictly_increasing_c1(VARS).

strictly_increasing_c1([]) :- !.
strictly_increasing_c1([_28356]) :- !.
strictly_increasing_c1([X,Y|R]) :- X<Y,
strictly_increasing_c1([Y|R]).

strictly_increasing_a(1,[]) :- !.
strictly_increasing_a(0,[]) :- !,
fail.

strictly_increasing_a(FLAG,VARIABLES) :-
collection(VARIABLES,[dvar]),
strictly_increasing_signature(VARIABLES,SIGNATURE),
AUTOMATON=automaton(
    SIGNATURE,
    _29487,
    SIGNATURE,
    [source(s),sink(s)],
    [arc(s,0,s)],
    [],
    [],
    []),
automaton_bool(FLAG,[0,1],AUTOMATON).

strictly_increasing_signature([_28357],[]) :- !.

strictly_increasing_signature(
    [[var-VAR1],[var-VAR2]|VARS],
    [S|Ss]) :-
    S in 0..1,
    VAR1#>=VAR2#<=S,
    strictly_increasing_signature([[var-VAR2]|VARS],Ss).
B.345 strongly_connected

◊ **Meta-Data:**

```prolog
ctr_date(strongly_connected,[\'20030820\',\'20040726\',\'20060817\']).

ctr_origin(
  strongly_connected,
  \cite{AlthausBockmayrElfKasperJungerMehlhorn02}, []).

ctr_arguments(
  strongly_connected,
  \['NODES\'-collection(index-int,succ-svar)]).

ctr_restrictions(
  strongly_connected,
  [required('NODES',[index,succ]),
   'NODES'\^index>=1,
   'NODES'\^index=<size('NODES'),
   distinct('NODES',index)]).

ctr_example(
  strongly_connected,
  strongly_connected(
    [[index-1,succ-{2}]],
    [index-2,succ-{3}],
    [index-3,succ-{2,5}],
    [index-4,succ-{1}],
    [index-5,succ-{4}]]).

ctr_typical(strongly_connected,[size('NODES')>2]).

ctr_exchangeable(strongly_connected,[items('NODES',all)]).

ctr_graph(
  strongly_connected,
  ['NODES'],
  2,
  ['\text{CIQUE}'>>collection(nodes1,nodes2)],
  [nodes2\^index in_set nodes1\^succ],
  ['\text{MIN_NSCC}'=size('NODES')],
  []).
```
B.346 subgraph_isomorphism

◊ Meta-Data:

ctr_predefined(subgraph_isomorphism).

ctr_date(subgraph_isomorphism, ['20090821']).

ctr_origin(subgraph_isomorphism, '\cite{Gregor79}', []).

ctr_arguments(subgraph_isomorphism, [\[NODES_PATTERN\]-collection(index-int,succ-sint)],\[NODES_TARGET\]-collection(index-int,succ-svar),\[FUNCTION\]-collection(image-dvar])].

ctr_restrictions(subgraph_isomorphism, [required('NODES_PATTERN', [index, succ]), 'NODES_PATTERN' index>=1, 'NODES_PATTERN' index=<size('NODES_PATTERN'), distinct('NODES_PATTERN', index), 'NODES_PATTERN' succ>=1, 'NODES_PATTERN' succ=<size('NODES_PATTERN'), required('NODES_TARGET', [index, succ]), 'NODES_TARGET' index>=1, 'NODES_TARGET' index=<size('NODES_TARGET'), distinct('NODES_TARGET', index), 'NODES_TARGET' succ>=1, 'NODES_TARGET' succ=<size('NODES_TARGET'), required('FUNCTION', [image]), 'FUNCTION' image>=1, 'FUNCTION' image=<size('NODES_TARGET'), distinct('FUNCTION', image), size('FUNCTION') = size('NODES_PATTERN')].

ctr_example(subgraph_isomorphism, subgraph_isomorphism([[index-1, succ-{2,4}]], [index-2, succ-{1,3,4}]), [index-3, succ-{}], [index-4, succ-{}], [[index-1, succ-{}], [index-2, succ-{3,4,5}]], [index-3, succ-{}]).
[index-4, succ-{2,5}],
[index-5, succ-{}],
[[image-4], [image-2], [image-3], [image-5]]).

ctr_typical{
    subgraph_isomorphism,
    [size('NODES_PATTERN') > 1, size('NODES_TARGET') > 1]).

ctr_exchangeable{
    subgraph_isomorphism,
    [items('NODES_PATTERN', all), items('NODES_TARGET', all)])
}
B.347 sum

◊ **META-DATA:**

ctr_date(sum,[’20030820’,’20040726’,’20060817’]).

ctr_origin(sum,’\cite{Yunes02}.’,[]).

ctr_synonyms(sum,[sum_pred]).

ctr_arguments(
    sum,
    [’INDEX’-dvar,
     ’SETS’-collection(ind-int,set-sint),
     ’CONSTANTS’-collection(cst-int),
     ’S’-dvar]).

ctr_restrictions(
    sum,
    [size(’SETS’)>=1,
     required(’SETS’,[ind,set]),
     distinct(’SETS’,ind),
     size(’CONSTANTS’)>=1,
     required(’CONSTANTS’,cst)]).

ctr_example(
    sum,
    sum(8,
        sum([ind-8,set-{2,3}],
            [ind-1,set-{3}],
            [ind-3,set-{1,4,5}],
            [ind-6,set-{2,4}],
            [[cst-4],[cst-9],[cst-1],[cst-3],[cst-1]],
            10))).

ctr_typical(
    sum,
    [size(’SETS’)>1,
     size(’CONSTANTS’) size(’SETS’),
     range(’CONSTANTS’‘cst’>1)].

ctr_exchangeable(sum,[items(’SETS’,all)]).

ctr_graph(
    sum,
    [’SETS’,’CONSTANTS’],
    ...
2,
[‘PRODUCT’>>collection(sets,constants)],
[‘INDEX’=sets.ind,constants.key in_set sets.set],
[‘SUM’(‘CONSTANTS’,cst)=’S’],
[]).

ctr_functional_dependency(sum,4,[1,2,3]).
B.348 sum_ctr

◇ Meta-Data:

ctr_date(sum_ctr,[’20030820’,’20040807’,’20060817’]).

ctr_origin(sum_ctr,’Arithmetic constraint.’,[]).

ctr_synonyms(sum_ctr,[constant_sum,sum,linear,scalar_product]).

ctr_arguments(
  sum_ctr,
  [’VARIABLES’-collection(var-dvar),’CTR’-atom,’VAR’-dvar]).

ctr_restrictions(
  sum_ctr,
  [required(’VARIABLES’,var),
   in_list(’CTR’,[=,\=,<,\>=,\>=,\=\<])]).

ctr_example(sum_ctr,sum_ctr([[var-1],[var-1],[var-4]],=,6)).

ctr_typical(
  sum_ctr,
  [size(’VARIABLES’)\>1,
   range(’VARIABLES’\^var)\>1,
   in_list(’CTR’,[=,\<,\>,\>=,\=\<])]).

ctr_exchangeable(sum_ctr,[items(’VARIABLES’,all)]).

ctr_graph(
  sum_ctr,
  [’VARIABLES’],
  1,
  [’SELF’\>>collection(variables)],
  [’TRUE’],
  [’CTR’(’SUM’(’VARIABLES’,var),’VAR’)],
  []).

ctr_eval(sum_ctr,[reformulation(sum_ctr_r)]).

ctr_pure_functional_dependency(sum_ctr,[in_list(’CTR’,[=])]).

ctr_contractible(
  sum_ctr,
  [in_list(’CTR’,[\<,\=\<]),minval(’VARIABLES’\^var)\>=0],
  VARIABLES,
any).

```prolog
ctr_contractible(  
    sum_ctr,  
    [in_list('CTR', [>=, >]), maxval('VARIABLES' ^ var) =< 0],  
    VARIABLES,  
    any).

ctr_extensible(  
    sum_ctr,  
    [in_list('CTR', [>=, >]), minval('VARIABLES' ^ var) >= 0],  
    VARIABLES,  
    any).

ctr_extensible(  
    sum_ctr,  
    [in_list('CTR', [<, =<]), maxval('VARIABLES' ^ var) =< 0],  
    VARIABLES,  
    any).
  
ctr_aggregate(sum_ctr, [], [union, id, +]).
```

```prolog
sum_ctr_r(VARIABLES, CTR, VAR) :-  
    collection(VARIABLES, [dvar]),  
    memberchk(CTR, [=, =\, <, =\, >, =\, =<]),  
    check_type(dvar, VAR),  
    get_attr1(VARIABLES, VARS),  
    build_sum_var(VARS, SUM),  
    call_term_relop_value(SUM, CTR, VAR).
```
B.349  sum_cubes_ctr

◊ **META-DATA:**

\(\texttt{ctr\_predefined(sum\_cubes\_ctr)}\).

\(\texttt{ctr\_date(sum\_cubes\_ctr, ['20111111'])}\).

\(\texttt{ctr\_origin(sum\_cubes\_ctr, 'Arithmetic constraint.', [])}\).

\(\texttt{ctr\_synonyms(}
\quad \texttt{sum\_cubes\_ctr},
\quad \texttt{[sum\_cubes, sum\_of\_cubes, sum\_of\_cubes\_ctr]}\).
\)

\(\texttt{ctr\_arguments(}
\quad \texttt{sum\_cubes\_ctr},
\quad \texttt{['VARIABLES'\-collection(var\-dvar), 'CTR'\-atom, 'VAR'\-dvar]}\).
\)

\(\texttt{ctr\_restrictions(}
\quad \texttt{sum\_cubes\_ctr},
\quad \texttt{[required('VARIABLES', var),}
\quad \quad \texttt{in\_list('CTR', [\=',\=,\<,\>,\>=,\<=])]}\).
\)

\(\texttt{ctr\_example(}
\quad \texttt{sum\_cubes\_ctr},
\quad \texttt{sum\_cubes\_ctr([[\text{var-1}],[\text{var-2}],[\text{var-2}]], =, 17)}\).
\)

\(\texttt{ctr\_typical(}
\quad \texttt{sum\_cubes\_ctr},
\quad \texttt{[size('VARIABLES')\>1,}
\quad \quad \texttt{range('VARIABLES'\^\text{var})\>1,}
\quad \quad \texttt{in\_list('CTR', [\!,\<,\>,\>=,\<=])]}\).
\)

\(\texttt{ctr\_exchangeable(sum\_cubes\_ctr, [items('VARIABLES', all)])}\).

\(\texttt{ctr\_eval(sum\_cubes\_ctr, [reformulation(sum\_cubes\_ctr\_r)])}\).

\(\texttt{ctr\_pure\_functional\_dependency(}
\quad \texttt{sum\_cubes\_ctr},
\quad \texttt{[in\_list('CTR', [\=])]})\).

\(\texttt{ctr\_contractible(}
\quad \texttt{sum\_cubes\_ctr},
\quad \texttt{[in\_list('CTR', [\!,\<,\<=])], minval('VARIABLES'\^\text{var})\>=0],}
\quad \texttt{VARIABLES, any})\).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

\begin{verbatim}
ctr_contractible(
    sum_cubes_ctr,
    [in_list('CTR',[>=,>]),maxval('VARIABLES'\^var)=<0],
    VARIABLES,
    any).

ctr_extensible(
    sum_cubes_ctr,
    [in_list('CTR',[>=,>]),minval('VARIABLES'\^var)>=0],
    VARIABLES,
    any).

ctr_extensible(
    sum_cubes_ctr,
    [in_list('CTR',[<,=<]),maxval('VARIABLES'\^var)=<0],
    VARIABLES,
    any).

ctr_aggregate(sum_cubes_ctr,[],[union,id,+]).

sum_cubes_ctr_r(VARIABLES,CTR,VAR) :-
    collection(VARIABLES,[dvar]),
    memberchk(CTR,[=,\=,<,>,>=,<=]),
    check_type(dvar,VAR),
    get_attr1(VARIABLES,VARS),
    build_sum_cubes_var(VARS,SUM_CUBES),
    call_term_relop_value(SUM_CUBES,CTR,VAR).
\end{verbatim}
B.350 sum_free

◊ **Meta-Data:**

ctr_predefined(sum_free).

ctr_date(sum_free,['20061003']).

ctr_origin(sum_free,'\\cite{HoeveSabharwal07}',[]).

ctr_arguments(sum_free,['S'-svar]).

ctr_example(sum_free,sum_free({1,3,5,9})).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.351 sum_of_increments

♦ META-DATA:

ctr_predefined(sum_of_increments).

ctr_date(sum_of_increments,['20111105']).

ctr_origin(sum_of_increments,'\cite{Brand09}',[]).

ctr_synonyms(
    sum_of_increments,
    [increments_sum, incr_sum, sum_incr, sum_increments]).

ctr_arguments(
    sum_of_increments,
    ['VARIABLES'-collection(var-dvar), 'LIMIT'-dvar]).

ctr_restrictions(
    sum_of_increments,
    [required('VARIABLES', var), 'VARIABLES'~var>=0, 'LIMIT'>=0]).

ctr_example(
    sum_of_increments,
    [sum_of_increments(
        [[var-4],[var-4],[var-3],[var-4],[var-6]],
        7)]).

ctr_typical(
    sum_of_increments,
    [size('VARIABLES')>2,
      range('VARIABLES'~var)>1,
      maxval('VARIABLES'~var)>0,
      'LIMIT'>0]).

ctr_exchangeable(
    sum_of_increments,
    [translate(['VARIABLES'~var,'LIMIT']),
      items('VARIABLES',reverse),
      vals(['LIMIT'],int,<,dontcare,dontcare)]).

ctr_eval(
    sum_of_increments,
    [reformulation(sum_of_increments_r)]).

ctr_contractible(sum_of_increments,[],'VARIABLES',prefix).
ctr_contractible(sum_of_increments, [], 'VARIABLES', suffix).

sum_of_increments_r([], _17148) :- !.

sum_of_increments_r(VARIABLES, LIMIT) :-
  collection(VARIABLES, [dvar_gteq(0)]),
  check_type(dvar_gteq(0), LIMIT),
  get_attr1(VARIABLES, VARS),
  fd_max(LIMIT, MaxL),
  sum_of_increments_r1([0|VARS], MaxL, SUM),
  call(SUM#=<LIMIT).

sum_of_increments_r1([_17150], _17148, 0) :- !.

sum_of_increments_r1([V1, V2|R], MaxL, S2+S) :-
  S2 in 0..MaxL,
  V2-V1#=<S2,
  sum_of_increments_r1([V2|R], MaxL, S).
B.352 sum_of_weights_of_distinct_values

◊ Meta-Data:

ctr_date(
    sum_of_weights_of_distinct_values,
    ['20030820','20040726','20060817']).

ctr_origin(
    sum_of_weights_of_distinct_values,
    \cite{BeldiceanuCarlssonThiel02},
    []).

ctr_synonyms(sum_of_weights_of_distinct_values,[swd]).

ctr_arguments(
    sum_of_weights_of_distinct_values,
    ['VARIABLES'-collection(var-dvar),
    'VALUES'-collection(val-int,weight-int),
    'COST'-dvar]).

ctr_restrictions(
    sum_of_weights_of_distinct_values,
    [required('VARIABLES',var),
    size('VALUES')>0,
    required('VALUES',[val,weight]),
    'VALUES'\^weight>=0,
    distinct('VALUES',val),
    in_attr('VARIABLES',var,'VALUES',val),
    'COST'>=0]).

ctr_example(
    sum_of_weights_of_distinct_values,
    sum_of_weights_of_distinct_values( 
      [[var-1],[var-6],[var-1]],
      [[val-1,weight-5],[val-2,weight-3],[val-6,weight-7]],
      12)).

ctr_typical(
    sum_of_weights_of_distinct_values,
    [size('VARIABLES')>1,
    range('VARIABLES'\^var)>1,
    size('VALUES')>1,
    range('VALUES'\^weight)>1,
    'VALUES'\^weight>0]).
ctr_exchangeable(
    sum_of_weights_of_distinct_values,
    [items('VARIABLES', all),
     vals(['VARIABLES'°var], int, =\=, all, in),
     items('VALUES', all),
     vals(
         ['VARIABLES'°var,'VALUES'°val],
         int,
         =\=,
         all,
         dontcare))).

ctr_graph(
    sum_of_weights_of_distinct_values,
    ['VARIABLES','VALUES'],
    2,
    ['PRODUCT'>>collection(variables,values)],
    [variables°var=values°val],
    ['NSOURCE'=size('VARIABLES'),
     'SUM'('VALUES',weight)='COST'],
    []).

ctr_eval(
    sum_of_weights_of_distinct_values,
    [reformulation(sum_of_weights_of_distinct_values_r)]).

ctr_pure_functional_dependency(
    sum_of_weights_of_distinct_values,
    []).

ctr_functional_dependency(
    sum_of_weights_of_distinct_values,
    3,
    [1,2]).

sum_of_weights_of_distinct_values_r(VARIABLES,VALUES,COST) :-
    collection(VARIABLES,[dvar]),
    collection(VALUES,[int,int_gteq(0)]),
    check_type(dvar_gteq(0),COST),
    get_attr1(VARIABLES,VARS),
    get_attr1(VALUES,VALS),
    get_attr2(VALUES,WEIGHTS),
    all_different(VALS),
    ( VALUES=[] ->
        COST\#=0
    ;   sum_of_weights_of_distinct_values1(VARS,VALS),
        ... )

sum_of_weights_of_distinct_values1(VARS,VALS),
    ...
sum_of_weights_of_distinct_values3(
  VALS,
  WEIGHTS,
  VARS,
  TERM),
  call(COST#=TERM)
).

sum_of_weights_of_distinct_values1([],34082).

sum_of_weights_of_distinct_values1([VAR|RVAR],VALS) :-
  sum_of_weights_of_distinct_values2(VALS,VAR,OR_TERM),
  call(OR_TERM),
  sum_of_weights_of_distinct_values1(RVAR,VALS).

sum_of_weights_of_distinct_values2([],34082,0).

sum_of_weights_of_distinct_values2([VAL|RVAL],
  VAR,
  VAR#=VAL#
  /TERM) :-
  sum_of_weights_of_distinct_values2(RVAL,VAR,TERM).

sum_of_weights_of_distinct_values3([],[],34083,0).

sum_of_weights_of_distinct_values3([VAL|RVAL],
  [WEIGHT|RWEIGHT],
  VARS,
  WEIGHT* B+TERM) :-
  sum_of_weights_of_distinct_values4(VARS,VAL,OR_TERM),
  call(B#<=>OR_TERM),
  sum_of_weights_of_distinct_values3(
    RVAL,
    RWEIGHT,
    VARS,
    TERM).

sum_of_weights_of_distinct_values4([],34082,0).

sum_of_weights_of_distinct_values4([VAR|RVAR],
  VAL,
  VAL#=VAR#
  /TERM) :-
B.353 sum_set

◊ META-DATA:

ctr_date(sum_set,[‘20031001’,‘20060818’]).

ctr_origin(sum_set,’H.˜Cambazard’,[]).

ctr_arguments(
    sum_set,
    [‘SV’-svar,
      ‘VALUES’-collection(val-int,coef-int),
      ‘CTR’-atom,
      ‘VAR’-dvar]).

ctr_restrictions(
    sum_set,
    [required(‘VALUES’,[val,coef]),
     distinct(‘VALUES’,val),
     ‘VALUES’^coef>=0,
     in_list(‘CTR’,[=,\=,<,>=,>,=<])).

ctr_example(
    sum_set,
    sum_set(
      [2,3,6],
      [val-2,coef-7],
      [val-9,coef-1],
      [val-5,coef-7],
      [val-6,coef-2],
      =,
      9)).

ctr_typical(
    sum_set,
    [size(‘VALUES’)>1,
     ‘VALUES’^coef>0,
     in_list(‘CTR’,[=,\=,<,>=,>,=<]))).

ctr_exchangeable(sum_set,[items(‘VALUES’,all)]).

ctr_graph(
    sum_set,
    [‘VALUES’],
    1,
    [‘SELF’>>collection(values)],
[values\text{'}val \text{ in_set } \text{'}SV\text{'}],
[\text{'}CTR\text{'}(\text{'}SUM\text{'}(\text{'}VALUES\text{', coef}, \text{'}VAR\text{'}))],
[\text{'}]].
**B.354 sum_squares_ctr**

◊ **META-DATA:**

```prolog
ctr_predefined(sum_squares_ctr).

ctr_date(sum_squares_ctr, [’20110612’]).

ctr_origin(sum_squares_ctr, ’Arithmetic constraint.’, []).

ctr_synonyms(
    sum_squares_ctr,
    [sum_squares, sum_of_squares, sum_of_squares_ctr]).

ctr_arguments(
    sum_squares_ctr,
    [’VARIABLES’-collection(var-dvar), ’CTR’-atom, ’VAR’-dvar]).

ctr_restrictions(
    sum_squares_ctr,
    [required(’VARIABLES’, var),
        in_list(’CTR’, [=, ==, <, >, >=, =<])]).

ctr_example(
    sum_squares_ctr,
    sum_squares_ctr([[var-1], [var-1], [var-4]], =, 18)).

ctr_typical(
    sum_squares_ctr,
    [size(’VARIABLES’) > 1,
        range(’VARIABLES’^var) > 1,
        in_list(’CTR’, [=, <, >, >=, =<])]).

ctr_exchangeable(sum_squares_ctr, [items(’VARIABLES’, all)]).

ctr_eval(sum_squares_ctr, [reformulation(sum_squares_ctr_r)]).

ctr_pure_functional_dependency(
    sum_squares_ctr,
    [in_list(’CTR’, [=])]).

ctr_contractible(
    sum_squares_ctr,
    [in_list(’CTR’, [<, =<]), VARIABLES, any]).
```
ctr_extensible(
    sum_squares_ctr,
    [in_list('CTR',[>=,>])],
    VARIABLES,
    any).

ctr_aggregate(sum_squares_ctr,[],[union,id,+]).

sum_squares_ctr_r(VARIABLES,CTR,VAR) :-
    collection(VARIABLES,[dvar]),
    memberchk(CTR,[=,\=,<,\>=,>,\=,<=,\=,>]),
    check_type(dvar,VAR),
    get_attr1(VARIABLES,VARS),
    build_sum_squares_var(VARS,SUM_SQUARES),
    call_term_relop_value(SUM_SQUARES,CTR,VAR).
B.355 symmetric

◇ Meta-Data:

ctr_date(symmetric, ['20060930']).

ctr_origin(symmetric, '\cite{Dooms06}', []).

ctr_arguments(
  symmetric,
  ['NODES' - collection(index-int, succ-svar)]).

ctr_restrictions(
  symmetric,
  [required('NODES', [index, succ]),
   'NODES'^index>=1,
   'NODES'^index=<size('NODES'),
   distinct('NODES', index)]).

ctr_example(
  symmetric,
  symmetric(
    [[index-1, succ-{1,2,3}],
     [index-2, succ-{1,3}],
     [index-3, succ-{1,2}],
     [index-4, succ-{5,6}],
     [index-5, succ-{4}],
     [index-6, succ-{4}]]).

ctr_typical(symmetric, [size('NODES')>2]).

ctr_exchangeable(symmetric, [items('NODES', all)]).

ctr_graph(
  symmetric,
  ['NODES'],
  2,
  ['CLIQUE'>>collection(nodes1, nodes2)],
  [nodes2^index in_set nodes1^succ],
  [],
  ['SYMMETRIC']).
B.356  symmetric_alldifferent

♢ Meta-Data:

ctr_date(
    symmetric_alldifferent,
    ['20000128','20030820','20060818']).

ctr_origin(symmetric_alldifferent,‘\cite{Regin99}’,[],[]).

ctr_synonyms(
    symmetric_alldifferent,
    [symmetric_alldiff,
     symmetric_alldistinct,
     symm_alldifferent,
     symm_alldiff,
     symm_alldistinct,
     one_factor,
     two_cycle]).

ctr_arguments(
    symmetric_alldifferent,
    ['NODES'—collection(index-int,succ-dvar)]).

ctr_restrictions(
    symmetric_alldifferent,
    [size('NODES')mod 2=0,
     required('NODES',[index,succ]),
     'NODES'~index>=1,
     'NODES'~index=<size('NODES'),
     distinct('NODES',index),
     'NODES'~succ>=1,
     'NODES'~succ=<size('NODES')]).

ctr_example(
    symmetric_alldifferent,
    symmetric_alldifferent(
        [[index-1,succ-3],
         [index-2,succ-4],
         [index-3,succ-1],
         [index-4,succ-2]]).

ctr_typical(symmetric_alldifferent,[size('NODES')>=4]).

ctr_exchangeable(symmetric_alldifferent,[items('NODES',all)]).
ctr_graph(
    symmetric_alldifferent,
    ['NODES'],
    2,
    ['CLIQUE'(\=\=>collection(nodes1,nodes2)],
    [nodes1\^succ=x=nodes2\^index,nodes2\^succ=x=nodes1\^index],
    ['NARC'=size('NODES')],
    []).

ctr_eval(
    symmetric_alldifferent,
    [reformulation(symmetric_alldifferent_r1),
     reformulation(symmetric_alldifferent_r2)]).

symmetric_alldifferent_r1(NODES) :-
    symmetric_alldifferent_r1a(NODES,INODES),
    eval(inverse(INODES)).

symmetric_alldifferent_r1a([],[]).

symmetric_alldifferent_r1a(  
    [[index-INDEX,succ-SUCC]|R],
    [[index-INDEX,succ-SUCC,pred-SUCC]|S] ) :-
    SUCC\#\=INDEX,
    symmetric_alldifferent_r1a(R,S).

symmetric_alldifferent_r2([],[]) :-
    !.

symmetric_alldifferent_r2(NODES) :-
    symmetric_alldifferent0(NODES,SNODES),
    length(SNODES,N),
    collection(SNODES,[int(1,N),dvar(1,N)]),
    get_attr1(SNODES,INDEXES),
    get_attr2(SNODES,SUCCS),
    all_different(INDEXES),
    derangement1(SUCCS,INDEXES),
    symmetric_alldifferent1(SUCCS,1,SUCCS).
B.357 symmetric_alldifferent_except_0

◊ **Meta-Data:**

```
ctr_predefined(symmetric_alldifferent_except_0).
ctr_date(symmetric_alldifferent_except_0, [’20120208’]).
ctr_origin(symmetric_alldifferent_except_0, Derived from %c, [symmetric_alldifferent]).
ctr_synonyms(symmetric_alldifferent_except_0, [symmetric_alldiff_except_0, symmetric_alldistinct_except_0, symm_alldifferent_except_0, symm_alldiff_except_0, symm_alldistinct_except_0]).
ctr_arguments(symmetric_alldifferent_except_0, [’NODES’-collection(index-int, succ-dvar)]).
ctr_restrictions(symmetric_alldifferent_except_0, [required(’NODES’, [index, succ]), ’NODES’^index>=1, ’NODES’^index=<size(’NODES’), distinct(’NODES’, index), ’NODES’^succ>=0, ’NODES’^succ=<size(’NODES’)]).
ctr_example(symmetric_alldifferent_except_0, symmetric_alldifferent_except_0( [[index-1, succ-3], [index-2, succ-0], [index-3, succ-1], [index-4, succ-0]])).
ctr_typical(symmetric_alldifferent_except_0, [size(’NODES’) >=4, minval(’NODES’^succ)=0]).
```
ctr_exchangeable(
    symmetric_alldifferent_except_0,
    [items('NODES',all)]).

ctr_eval(
    symmetric_alldifferent_except_0,
    [reformulation(symmetric_alldifferent_except_0_r)]).

symmetric_alldifferent_except_0_r([]) :- !.

symmetric_alldifferent_except_0_r(NODES) :-
    symmetric_alldifferent0(NODES,SNODES),
    length(SNODES,N),
    collection(SNODES,[int(1,N),dvar(0,N)]),
    get_attr1(SNODES,INDEXES),
    get_attr2(SNODES,SUCCS),
    all_different(INDEXES),
    derangement1(SUCCS,INDEXES),
    symmetric_alldifferent1(SUCCS,1,SUCCS).
B.358  symmetric_cardinality

◊ Meta-Data:

ctr_date(symmetric_cardinality,['20040530','20060818']).

ctr_origin(symmetric_cardinality,
Derived from %c by W. Kocjan. [global_cardinality]).

ctr_arguments(symmetric_cardinality,
['VARS'-collection(idvar-int,var-svar,l-int,u-int),
 'VALS'-collection(idval-int,val-svar,l-int,u-int)]).

ctr_restrictions(symmetric_cardinality,
[required('VARS',[idvar,var,l,u]),
 size('VARS')>=1,
 'VARS''idvar=1,
 'VARS''idvar=<size('VARS'),
 distinct('VARS',idvar),
 'VARS''1'>=0,
 'VARS''1=<'VARS''u,
 'VARS''u=<size('VARS'),
 required('VALS',[idval,val,l,u]),
 size('VALS')>=1,
 'VALS''idval=1,
 'VALS''idval=<size('VALS'),
 distinct('VALS',idval),
 'VALS''1'>=0,
 'VALS''1=<'VALS''u,
 'VALS''u=<size('VARS')]).

ctr_example(symmetric_cardinality,
[symmetrical_cardinality(
[[idvar-1,var-{3},l-0,u-1],
 [idvar-2,var-{1},l-1,u-2],
 [idvar-3,var-{1,2},l-1,u-2],
 [idvar-4,var-{1,3},l-2,u-3]],
 [[idval-1,val-{2,3,4},l-3,u-4],
 [idval-2,val-{3},l-1,u-1],
 [idval-3,val-{1,4},l-1,u-2],
 [idval-4,val-{},l-0,u-1}]).]}).
ctr_typical(
    symmetric_cardinality,
    [size('VARS')>1,size('VALS')>1]).

ctr_exchangeable(
    symmetric_cardinality,
    [items('VARS',all),items('VALS',all)]).
B.359 symmetric_gcc

◊ Meta-Data:

ctr_date(symmetric_gcc,[‘20030820’,’20040530’,’20060818’]).

ctr_origin(
    symmetric_gcc,
    Derived from %c by W. Kocjan.,
    [global_cardinality]).

ctr_synonyms(symmetric_gcc,[sgcc]).

ctr_arguments(
    symmetric_gcc,
    [‘VARS’-collection(idvar-int,var-svar,nocc-dvar),
    ‘VALS’-collection(idval-int,val-svar,nocc-dvar)]).

ctr_restrictions(
    symmetric_gcc,
    [required(‘VARS’,[idvar,var,nocc]),
    size(‘VARS’)>=1,
    ‘VARS’ˆidvar>=1,
    ‘VARS’ˆidvar=<size(‘VARS’),
    distinct(‘VARS’,idvar),
    ‘VARS’ˆnocc>=0,
    ‘VARS’ˆnocc=<size(‘VALS’),
    required(‘VALS’,[idval,val,nocc]),
    size(‘VALS’)>=1,
    ‘VALS’ˆidval>=1,
    ‘VALS’ˆidval=<size(‘VALS’),
    distinct(‘VALS’,idval),
    ‘VALS’ˆnocc>=0,
    ‘VALS’ˆnocc=<size(‘VARS’))].

ctr_example(
    symmetric_gcc,
    symmetric_gcc(
        [[idvar-1,var-{3},nocc-1],
        [idvar-2,var-{1},nocc-1],
        [idvar-3,var-{1,2},nocc-2],
        [idvar-4,var-{1,3},nocc-2]],
        [[idval-1,val-{2,3,4},nocc-3],
        [idval-2,val-{3},nocc-1],
        [idval-3,val-{1,4},nocc-2],
        [idval-4,val-{},nocc-0]]).
ctr_typical(symmetric_gcc,[size('VARS')>1,size('VALS')>1]).

ctr_exchangeable(
    symmetric_gcc,
    [items('VARS',all),items('VALS',all)]).

ctr_graph(
    symmetric_gcc,
    ['VARS','VALS'],
    2,
    ['PRODUCT'>>collection(vars,vals)],
    [vars`idvar in_set vals`val#<=>vals`idval in_set vars`var,
     vars`nocc=card_set(vars`var),
     vals`nocc=card_set(vals`val)],
    ['NARC'=size('VARS')*size('VALS')],
    []).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.360  temporal_path

◊ Meta-Data:

ctr_date(
  temporal_path,
  ['20000128','20030820','20060818','20090511']).

ctr_origin(temporal_path,'ILOG',[]).

ctr_arguments(
  temporal_path,
  ['NPATH'-dvar, NODES-
   collection(index-int,succ-dvar,start-dvar,end-dvar)]).

ctr_restrictions(
  temporal_path,
  ['NPATH']>=1, 'NPATH'=<size('NODES'),
  required('NODES',[index,succ,start,end]),
  size('NODES')>0, 'NODES'\^index>=1, 'NODES'\^index=<size('NODES'),
  distinct('NODES',index), 'NODES'\^succ=1, 'NODES'\^succ<size('NODES'),
  'NODES'\^start=<'NODES'\^end).

ctr_example(
  temporal_path,
  temporal_path(2,
  [[index-1,succ-2,start-0,end-1],
   [index-2,succ-6,start-3,end-5],
   [index-3,succ-4,start-0,end-3],
   [index-4,succ-5,start-4,end-6],
   [index-5,succ-7,start-7,end-8],
   [index-6,succ-6,start-7,end-9],
   [index-7,succ-7,start-9,end-10]])).

ctr_typical(
  temporal_path,
  ['NPATH'<size('NODES'),
   size('NODES')>1, 'NODES'\^start< 'NODES'\^end]).
ctr_exchangeable(  
temporal_path,  
[items('NODES',all),  
  translate([`NODES`^start,`NODES`^end])]).

ctr_graph(  
temporal_path,  
['NODES'],  
2,  
['CLIQUE'>>collection(nodes1,nodes2)],  
[nodes1`succ=nodes2`index,  
  nodes1`succ=nodes1`index#/nodes1`end=<nodes2`start,  
  nodes1`start=<nodes1`end,  
  nodes2`start=<nodes2`end],  
['MAX_ID'=<1,'NCC'='NPATH','NVERTEX'=size('NODES')],  
[]).

ctr_eval(temporal_path,[reformulation(temporal_path_r)]).

ctr_functional_dependency(temporal_path,1,[2]).

temporal_path_r(NPATH,NODES) :-  
temporal_path0(NODES,SNODES),  
  length(SNODES,N),  
  N>0,  
  check_type(dvar(1,N),NPATH),  
  collection(SNODES,[int(1,N),dvar(1,N),dvar,dvar]),  
  get_attr1(SNODES,INDEXES),  
  get_attr2(SNODES,SUCCS),  
  get_attr3(SNODES,STARTS),  
  get_attr4(SNODES,ENDS),  
  all_different(INDEXES),  
  ori_end(STARTS,ENDS),  
  temporal_path1(INDEXES,SUCCS,TNODES),  
  eval(path(NPATH,TNODES)),  
  temporal_path2(SUCCS,ENDS,[]),STARTS).

temporal_path0(NODES,SNODES) :-  
temporal_path0a(NODES,L),  
  sort(L,S),  
  temporal_path0a(SNODES,S),  
  !.

temporal_path0a([],[]).
temporal_path0a([\[index-INDEX,succ-SUCC,start-START,end-END\]|R], [INDEX-(SUCC,START,END)|T]) :-
    temporal_path0a(R,T).

temporal_path1([],[],[]).

temporal_path1([INDEX|RINDEX], [SUCC|RSUCC], [[index-INDEX,succ-SUCC]|RNODES]) :-
    temporal_path1(RINDEX,RSUCC,RNODES).

temporal_path2([],[],_43171,_43172).

temporal_path2([SUCCi|RSUCC], [ENDi|REND], PREV_STARTS, [_STARTi|RSTART]) :-
    append(PREV_STARTS,[ENDi],NEW_PREV_STARTS),
    append(NEW_PREV_STARTS,RSTART,TABLE),
    element(SUCCi,TABLE,START_SUCCI),
    ENDi#=<START_SUCCI,
    temporal_path2(RSUCC,REND,NEW_PREV_STARTS,RSTART).
B.361  tour

◊ **META-DATA:**

ctr_date(tour,['20030820','20060819']).

ctr_origin(tour,\cite{AlthausBockmayrElfKasperJungerMehlhorn02},[]).

ctr_synonyms(tour,[atour,cycle]).

ctr_arguments(tour,['NODES'-collection(index-int,succ-svar)]).

ctr_restrictions(tour,[size('NODES')>=3,
                    required('NODES',\{index,succ\}),
                    'NODES'\^index>=1,
                    'NODES'\^index=<size('NODES'),
                    distinct('NODES',\{index\})].

ctr_example(tour,
tour(\[\[\{index-1,succ-{2,4}\}],
            [\{index-2,succ-{1,3}\}],
            [\{index-3,succ-{2,4}\}],
            [\{index-4,succ-{1,3}\}]]).)

ctr_exchangeable(tour,[items('NODES',all)]).

ctr_graph(tour,['NODES'],2,
          ['CLIQUE'=('=')==>collection(nodes1,nodes2)],
          [nodes2\^index in_set nodes1\^succ#=>
           nodes1\^index in_set nodes2\^succ],
          ['NARC'=size('NODES')\*size('NODES')-size('NODES'),[]].

ctr_graph(tour,['NODES'],
          ['CLIQUE'=('=')==>collection(nodes1,nodes2)],
          [nodes2\^index in_set nodes1\^succ#=>
           nodes1\^index in_set nodes2\^succ],
          ['NARC'=size('NODES')\*size('NODES')-size('NODES'),[]].)
2,
[‘CLIQUE’(\=\=)>>collection(nodes1,nodes2)],
[nodes2\^index in_set nodes1\^succ],
[‘MIN_NSCC’=size(‘NODES’),
 ‘MIN_ID’=2,
 ‘MAX_ID’=2,
 ‘MIN_OD’=2,
 ‘MAX_OD’=2],
[]).
B.362 track

◊ META-DATA:

ctr_date(track,[’20030820’,’20060819’,’20090510’]).

ctr_origin(track,’\cite{Marte01}’,[]).

ctr_arguments(
    track,
    [’NTRAIL’-int,
     ’TASKS’-collection(trail-int,origin-dvar,end-dvar)]).

ctr_restrictions(
    track,
    [’NTRAIL’>0,
     ’NTRAIL’=<size(’TASKS’),
     required(’TASKS’,[trail,origin,end]),
     ’TASKS’^origin<’TASKS’^end]).

ctr_example(
    track,
    track(2,
        [[trail-1,origin-1,end-2],
         [trail-2,origin-1,end-2],
         [trail-1,origin-2,end-4],
         [trail-2,origin-2,end-3],
         [trail-2,origin-3,end-4]])).

ctr_typical(
    track,
    [’NTRAIL’<size(’TASKS’),
     size(’TASKS’)>1,
     range(’TASKS’^trail)>1,
     ’TASKS’^origin<’TASKS’^end]).

ctr_exchangeable(
    track,
    [items(’TASKS’,all),
     vals([’TASKS’^trail],int,\=,all,dontcare),
     translate([’TASKS’^origin,’TASKS’^end])].

ctr_derived_collections(
    track,
    [col(TIME_POINTS-}
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

collection(origin-dvar,end-dvar,point-dvar),
  [item(
    origin-'TASKS'ˆorigin,
    end-'TASKS'ˆend,
    point-'TASKS'ˆorigin),
    item(
    origin-'TASKS'ˆorigin,
    end-'TASKS'ˆend,
    point-'TASKS'ˆend-1)]).

ctr_graph(
  track, [\['TASKS'\],
  1, [\['SELF'\] >> collection(tasks)],
  [tasksˆorigin=<tasksˆend],
  [\['NARC'=size('TASKS')]],
  []).

ctr_graph(
  track, [\['TIME_POINTS'\], 'TASKS'],
  2, [\['PRODUCT'\] >> collection(time_points,tasks)],
  [time_pointsˆend>time_pointsˆorigin, 
    tasksˆorigin=<time_pointsˆpoint, 
    time_pointsˆpoint<tasksˆend],
  [], [], [], 
  [SUCC>>
    [source, 
      variables-
      col('VARIABLES' - collection(var-dvar),
      [item(var-'TASKS'ˆtrail)]),
      [nvalue('NTRAIL',variables)])].

ctr_eval(track, [reformulation(track_r)]).

track_r(NTRAIL,TASKS) :-
  length(TASKS,N),
  check_type(dvar(1,N),NTRAIL),
  collection(TASKS,[int(1,N),dvar,dvar]),
  get_attr1(TASKS,TRAILS),
  get_attr2(TASKS,ORIGINS),
  get_attr3(TASKS,ENDS),
  ori_end(ORIGINS,ENDS),
track1(ORIGINS, ENDS, TRAILS, 1, ORIGINS, ENDS, TRAILS, NTRAIL),

track3(ORIGINS, ENDS, TRAILS, 1, ORIGINS, ENDS, TRAILS, NTRAIL).

track1([], [], [], _49469, _49470, _49471, _49472, _49473).

track1([Oi|RO], [Ei|RE], [Ti|TC], I, ORIGINS, ENDS, TRAILS, NTRAIL) :-
  track2(ORIGINS, ENDS, TRAILS, 1, I, Oi, Ei, Ti, COLi),
  nvalue(NTRAIL, COLi),
  I1 is I+1,
  track1(RO, RE, TC, I1, ORIGINS, ENDS, TRAILS, NTRAIL).

track2([], [], [], _49469, _49470, _49471, _49472, _49473, []).

track2([_49478|RO], [_49482|RE], [_49486|RT], J, I, Oi, Ei, Ti, [Ti|R]) :-
  I=J,
  !,
  J1 is J+1,
  track2(RO, RE, RT, J1, I, Oi, Ei, Ti, R).

track2([Oj|RO], [Ej|RE], [Tj|RT], J, I, Oi, Ei, Ti, [Tij|R]) :-
  I!=J,
  K in 1..2,
  Min is min(Ti, Tj),
  Max is max(Ti, Tj),
  Tij in Min..Max,
  element(K, [Ti, Tj], Tij),
  Oj#=<Oi#/\Ej#<Oi#/\Tij#=Tj#/\Oj#<Oi#/\Ej#<Oi#/\Tij#=Ti,
  J1 is J+1,
  track2(RO, RE, RT, J1, I, Oi, Ei, Ti, R).
track3([],[],[],_49469,_49470,_49471,_49472,_49473).

track3([Oi|RO],[Ei|RE],[Ti|TC],I,ORIGINS,ENDS,TRAILS,NTRAIL) :-
    track4(ORIGINS,ENDS,TRAILS,I,Oi,Ei,Ti,COLi),
    I1 is I+1,
    track3(RO,RE,TC,I1,ORIGINS,ENDS,TRAILS,NTRAIL).

track4([],[],[],_49469,_49470,_49471,_49472,_49473,[]).

track4([_49478|RO],[_49482|RE],[_49486|RT],J,I,Oi,Ei,Ti,[Ti|R]) :-
    I=J,
    _,
    J1 is J+1,
    track4(RO,RE,RT,J1,I,Oi,Ei,Ti,R).

track4([Oj|RO],[Ej|RE],[Tj|RT],J,I,Oi,Ei,Ti,[Tij|R]) :-
    I\=\=J,
    K in 1..2,
    Min is min(Ti,Tj),
    Max is max(Ti,Tj),
    Tij in Min..Max,
    element(K,[Ti,Tj],Tij),
    Oj\=<Ei-1#/Ej#/Ei-1#/\Tij#=Tj#/\Oj#/Ei-1#/\Ej#=Ei-1#/\Tij#=Ti,
    J1 is J+1,
    track4(RO,RE,RT,J1,I,Oi,Ei,Ti,R).
B.363  tree

◊ META-DATA:

ctr_date(tree,[‘20000128’,‘20030820’,‘20060819’]).

ctr_origin(tree,’N. Beldiceanu’,[]).

ctr_arguments(
  tree,
  [‘NTREES’-dvar,’NODES’-collection(index-int,succ-dvar)]).

ctr_restrictions(
  tree,
  [‘NTREES’>=1,
   ‘NTREES’=<size(‘NODES’),
   required(‘NODES’,[index,succ]),
   ‘NODES’^index>=1,
   ‘NODES’^index=<size(‘NODES’),
   distinct(‘NODES’,index),
   ‘NODES’^succ>=1,
   ‘NODES’^succ=<size(‘NODES’)]).

ctr_example(
  tree,
  tree(2,
    [[index-1,succ-1],
     [index-2,succ-5],
     [index-3,succ-5],
     [index-4,succ-7],
     [index-5,succ-1],
     [index-6,succ-1],
     [index-7,succ-7],
     [index-8,succ-5]])).

ctr_typical(tree,[‘NTREES’<size(‘NODES’),size(‘NODES’)>2]).

ctr_exchangeable(tree,[items(‘NODES’,all)]).

ctr_graph(
  tree,
  [‘NODES’],
  2,
  [‘CLIQUE’>>collection(nodes1,nodes2)],
  [nodes1^succ=nodes2^index],
[‘MAX_NSCC’=<1,’NCC’=’NTREES’],
[],).

ctr_eval(tree,[reformulation(tree_r)]).

ctr_functional_dependency(tree,1,[2]).

ctr_sol(tree,_A000272,[1,3,16,125,1296,16807,262144]).

tree_r(NTREES,NODES) :-
   length(NODES,N),
   check_type(dvar(1,N),NTREES),
   collection(NODES,[int(1,N),dvar(1,N)]),
   get_attr1(NODES,INDEXES),
   get_attr2(NODES,SUCCS),
   all_different(INDEXES),
   length(RANKS,N),
   domain(RANKS,1,N),
   tree1(SUCCS,RANKS,INDEXES,RANKS,INDEXES,Term),
   call(NTREES#=Term).

tree1([],[],_48669,_48670,_48671,0).

tree1([S|U],[R|P],[I|K],RANKS,INDEXES,B+T) :-
   S#=I#<=>B,
   tree2(S,R,I,RANKS,INDEXES),
   tree1(U,P,K,RANKS,INDEXES,T).

tree2(_48670,_48671,_48672,_48673,[]) :-
   !.

tree2(S_I,R_I,I,[R_J|P],[J|K]) :-
   S_I#=J#/I#\=J#=>R_I#<R_J,
   tree2(S_I,R_I,I,P,K).
B.364  tree_range

◊ Meta-Data:

ctr_date(
  tree_range,
  [‘20030820’,‘20040727’,‘20060819’,‘20090923’]).

ctr_origin(tree_range,’Derived from %c.’,[tree]).

ctr_arguments(
  tree_range,
  [‘NTREES’-dvar,
   ‘R’-dvar,
   ‘NODES’-collection(index-int,succ-dvar)]).

ctr_restrictions(
  tree_range,
  [‘NTREES’>=0,
   ‘R’>=0,
   ‘R’<size(‘NODES’),
   size(‘NODES’)>0,
   required(‘NODES’,[index,succ]),
   ‘NODES’-index=1,
   ‘NODES’-index=<size(‘NODES’),
   distinct(‘NODES’,index),
   ‘NODES’-succ=1,
   ‘NODES’-succ=<size(‘NODES’)]).

ctr_example(
  tree_range,
  tree_range(2,1,[[index-1,succ-1],[index-2,succ-5],[index-3,succ-5],[index-4,succ-7],[index-5,succ-1],[index-6,succ-1],[index-7,succ-7],[index-8,succ-5]]).

ctr_typical(
  tree_range,
  [‘NTREES’<size(‘NODES’),size(‘NODES’)>2]).
ctr_exchangeable(tree_range,[items('NODES',all)]).

ctr_graph(
  tree_range,
  [‘NODES’],
  2,
  ['CLIQUE'>>collection(nodes1,nodes2)],
  [nodes1\succ=nodes2\^index],
  ['MAX_NSNC'=<1,'NCC'='NTREES','RANGE_DRG'='R'],
  [{}].

ctr_eval(tree_range,[reformulation(tree_range_r)]).

ctr_functional_dependency(tree_range,1,[3]).

ctr_functional_dependency(tree_range,2,[3]).

tree_range_r(NTREES,R,NODES) :-
  tree_range0(NODES,SNODES),
  length(SNODES,N),
  N>0,
  N1 is N-1,
  check_type(dvar(1,N),NTREES),
  check_type(dvar(0,N1),R),
  collection(SNODES,[int(1,N),dvar(1,N)]),
  get_attr1(SNODES,INDEXES),
  get_attr2(SNODES,SUCCS),
  all_different(INDEXES),
  eval(tree(NTREES,SNODES)),
  tree_range1( INDEXES, 
                SUCCS, 
                DISTS1, 
                DISTS2, 
                OCCS1, 
                OCCS2, 
                SUCCS1, 
                LS, 
                OLS),
  eval(domain(DISTS1,0,N)),
  tree_range2(INDEXES,SUCCS,N,[],DISTS2),
  eval(domain(OCCS1,0,N)),
  eval(global_cardinality(SUCCS1,OCCS2)),
  eval(domain(LS,0,1)),
  tree_range3(OLS),
\begin{verbatim}

eval(in_interval(MIN,0,N)),
eval(open_minimum(MIN,OLS)),
eval(in_interval(MAX,0,N)),
eval(maximum(MAX,DISTS1)),
eval(
    scalar_product(
        [[coeff-1,var-MAX],[coeff-1,var-MIN]],
        =,
        R)).

tree_range0(NODES,SNODES) :-
    tree_range0a(NODES,L),
    sort(L,S),
    tree_range0a(SNODES,S),
    !.

    tree_range0a([],[]).

    tree_range0a([[index-I,succ-S]|R],[I-S|T]) :-
    tree_range0a(R,T).

    tree_range1([],[],[],[],[],[],[],[],[]).

    tree_range1([I|RI], [S|RS], [[var-V]|RV1], [[value-V]|RV2], [[var-O]|RO], [[val-I,noccurrence-O]|RIO], [[var-S]|RSS], [[var-L]|RL], [[var-O,bool-L]|ROL]) :-
    tree_range1(RI,RS,RV1,RV2,RO,RIO,RSS,RL,ROL).

    tree_range2([],[],_,_42122,_42123,_42124).

    tree_range2([_IND|RIND], [SUCC|RSUCC], N, DISTS_BEFORE, DISTS_AFTER) :-
    append(DISTS_BEFORE,[[value-0]],TD),
    DISTS_AFTER=[[value-D]|RDISTS_AFTER],
    append(TD, RDISTS_AFTER, TABLE),
\end{verbatim}
eval(in_interval(DS, 0, N)),
eval(element(SUCC, TABLE, DS)),
eval(
    scalar_product(
        [[coeff-1, var-D], [coeff-1, var-DS]],
        =, 1)),
append(DISTS_BEFORE, [[value-D]], DISTS_BEFORE1),
tree_range2(RIND, RSUCC, N, DISTS_BEFORE1, RDISTS_AFTER).

tree_range3([]).

tree_range3([[var-O, bool-L] | ROL]) :-
    L#<=O#>0,
    tree_range3(ROL).
B.365  tree_resource

◇ META-DATA:

ctr_date(tree_resource,['20030820','20060819']).

ctr_origin(tree_resource,'Derived from %c.',[tree]).

ctr_arguments(
    tree_resource,
    ['RESOURCE'~collection(id-int,nb_task-dvar),
     'TASK'~collection(id-int,father-dvar,resource-dvar)]).

ctr_restrictions(
    tree_resource,
    [size('RESOURCE')>0,
     required('RESOURCE',[id,nb_task]),
     'RESOURCE'~id>=1,
     'RESOURCE'~id<=size('RESOURCE'),
     distinct('RESOURCE',id),
     'RESOURCE'~nb_task>=0,
     'RESOURCE'~nb_task<=size('TASK'),
     required('TASK',[id,father,resource]),
     'TASK'~id=size('RESOURCE'),
     'TASK'~id<=size('RESOURCE')+size('TASK'),
     distinct('TASK',id),
     'TASK'~father>=1,
     'TASK'~father=size('RESOURCE')+size('TASK'),
     'TASK'~resource>=1,
     'TASK'~resource<=size('RESOURCE')]).

ctr_example(
    tree_resource,
    tree_resource(
        [[id-1,nb_task-4],[id-2,nb_task-0],[id-3,nb_task-1]],
        [[id-4,father-8,resource-1],
         [id-5,father-3,resource-3],
         [id-6,father-8,resource-1],
         [id-7,father-1,resource-1],
         [id-8,father-1,resource-1]]).

ctr_typical(
    tree_resource,
    [size('RESOURCE')>0,size('TASK')>size('RESOURCE')]).

ctr_exchangeable(}
tree_resource,
[items('RESOURCE',all),items('TASK',all)]).

ctr_derived_collections(
  tree_resource,
  [col(RESOURCE_TASK-
    collection(index-int,succ-dvar,name-dvar),
    [item(
      index-'RESOURCE'ˆid,
      succ-'RESOURCE'ˆid,
      name-'RESOURCE'ˆid),
      item(
      index-'TASK'ˆid,
      succ-'TASK'ˆfather,
      name-'TASK'ˆresource)])]).

ctr_graph(
  tree_resource,
  ['RESOURCE_TASK'],
  2,
  ['CLIQUE'>>collection(resource_task1,resource_task2)],
  [resource_task1ˆsucc=resource_task2ˆindex,
   resource_task1ˆname=resource_task2ˆname],
  ['MAX_NSNC'=<1,
   'NCC'=size('RESOURCE'),
   'NVERTEX'=size('RESOURCE')+size('TASK')],
  []).

ctr_graph(
  tree_resource,
  ['RESOURCE_TASK'],
  2,
  foreach(RESOURCE,
    ['CLIQUE'>>collection(resource_task1,resource_task2)],
    [resource_task1ˆsucc=resource_task2ˆindex,
     resource_task1ˆname=resource_task2ˆname,
     resource_task1ˆname='RESOURCE'ˆid],
    ['NVERTEX'='RESOURCE'ˆnb_task+1],
    []).

ctr_eval(tree_resource,[reformulation(tree_resource_r)]).

tree_resource_r(RESOURCE,TASK) :-
  length(RESOURCE,R),
  length(TASK,T),
R>0,
collection(RESOURCE,[int(1,R),dvar(0,T)]),
get_attr1(RESOURCE,RIDS),
get_attr2(RESOURCE,RNBTASKS),
all_different(RIDS),
R1 is R+1,
RT is R+T,
collection(TASK,[int(R1,RT),dvar(1,RT),dvar(1,R)]),
get_attr1(TASK,TIDS),
get_attr2(TASK,TFATHERS),
get_attr3(TASK,TRESOURCES),
all_different(TIDS),
tree_resource1(RIDS,CNODES1),
tree_resource2(TIDS,TFATHERS,CNODES2),
append(CNODES1,CNODES2,NODES),
eval(tree(R,NODES)),
tree_resource3(TIDS,TRESOURCES,TIR),
sort(TIR,STIR),
tree_resource4(1,R,INC),
append(INC,STIR,TAB),
tree_resource5(TAB,TABR),
tree_resource6(TFATHERS,TRESOURCES,TABR),
tree_resource7(STIR,GCVARS),
tree_resource8(RIDS,RNBTASKS,GCVALS),
eval(global_cardinality(GCVARS,GCVALS)).

tree_resource1([],[]).

tree_resource1([I|R],[[index-I,succ-I]|S]) :-
  tree_resource1(R,S).

tree_resource2([],[],[]).

tree_resource2([I|R],[F|S],[[index-I,succ-F]|T]) :-
  tree_resource2(R,S,T).

tree_resource3([],[],[]).

tree_resource3([I|RI],[R|RR],[I-R|S]) :-
  tree_resource3(RI,RR,S).

tree_resource4(I,R,[]) :-
  I>R,
  !.

tree_resource4(I,R,[I-I|S]) :-
\[ I < R, \]
\[ I_1 \text{ is } I + 1, \]
\[ \text{tree_resource4}(I_1, R, S). \]

\[ \text{tree_resource5}([], []). \]

\[ \text{tree_resource5}([_51558-R|S], [[\text{value-R}]] | T)) :- \]
\[ \text{tree_resource5}(S, T). \]

\[ \text{tree_resource6}([], [], _51552). \]

\[ \text{tree_resource6}([F_i|R_F], [R_i|R_R], \text{TABR}) :- \]
\[ \text{eval(element}(F_i, \text{TABR}, R_i)), \]
\[ \text{tree_resource6}(R_F, R_R, \text{TABR}). \]

\[ \text{tree_resource7}([], []). \]

\[ \text{tree_resource7}([_51558-V|R], [[\text{var-V}]] | S)) :- \]
\[ \text{tree_resource7}(R, S). \]

\[ \text{tree_resource8}([], [], []). \]

\[ \text{tree_resource8}([V|R], [O|S], [[\text{val-V}, \text{noccurrence-O}]] | T)) :- \]
\[ \text{tree_resource8}(R, S, T). \]
B.366 twin

◊ Meta-Data:

ctr_predefined(twin).

ctr_date(twin, [‘20111129’]).

ctr_origin(
		twin,
		Pairs of variables related by hidden %c constraints sharing the same table.,
		[element]).

ctr_arguments(twin, [‘PAIRS’-collection(x-dvar,y-dvar)]).

ctr_restrictions(
		twin,
		[required(‘PAIRS’, x), required(‘PAIRS’, y), size(‘PAIRS’)>0]).

ctr_example(
		twin,
		twin(
			[x-1,y-8],
			[x-9,y-6],
			[x-1,y-8],
			[x-5,y-0],
			[x-6,y-7],
			[x-9,y-6])).

ctr_typical(
		twin,
		[size(‘PAIRS’)>1,
			size(‘PAIRS’)>nval(‘PAIRS’\^x),
			size(‘PAIRS’)>nval(‘PAIRS’\^y),
			nval(‘PAIRS’\^x)>1,
			nval(‘PAIRS’\^y)>1,
			nval(‘PAIRS’\^x)=nval(‘PAIRS’\^y),
			nval(‘PAIRS’\^x)<size(‘PAIRS’),
			nval(‘PAIRS’\^y)<size(‘PAIRS’)]).

ctr_eval(twin, [checker(twin_c)]).

ctr_contractible(twin, [], ‘PAIRS’, any).

twin_c(PAIRS) :-

collection(PAIRS, [int,int]),
length(PAIRS,N),
N>0,
get_attr12(PAIRS,P12),
sort(P12,S12),
twin1(S12),
get_attr21(PAIRS,P21),
sort(P21,S21),
twin1(S21).

twin1([]) :-
  !.

twin1([_13004]) :-
  !.

twin1([X1-_13008,X2-Y|R]) :-
  X1==X2,
  twin1([X2-Y|R]).
B.367  two_layer_edge_crossing

◊ **META-DATA:**

ctr_date(two_layer_edge_crossing, ['20030820', '20060819']).

ctr_origin(
    two_layer_edge_crossing,
    Inspired by \cite{HararySchwenk72}.
).

ctr_arguments(
    two_layer_edge_crossing,
    ['NCROSS'-dvar,
    'VERTICES_LAYER1'-collection(id-int,pos-dvar),
    'VERTICES_LAYER2'-collection(id-int,pos-dvar),
    'EDGES'-collection(id-int,vertex1-int,vertex2-int)]).

ctr_restrictions(
    two_layer_edge_crossing,
    ['NCROSS'>=0,
    required('VERTICES_LAYER1',[id,pos]),
    'VERTICES_LAYER1'ˆid>=1,
    'VERTICES_LAYER1'ˆid=<size('VERTICES_LAYER1'),
    distinct('VERTICES_LAYER1',id),
    distinct('VERTICES_LAYER1',pos),
    required('VERTICES_LAYER2',[id,pos]),
    'VERTICES_LAYER2'ˆid>=1,
    'VERTICES_LAYER2'ˆid=<size('VERTICES_LAYER2'),
    distinct('VERTICES_LAYER2',id),
    distinct('VERTICES_LAYER2',pos),
    required('EDGES',[id,vertex1,vertex2]),
    'EDGES'ˆid>=1,
    'EDGES'ˆid=<size('EDGES'),
    distinct('EDGES',id),
    'EDGES'ˆvertex1>=1,
    'EDGES'ˆvertex1=<size('VERTICES_LAYER1'),
    'EDGES'ˆvertex2>=1,
    'EDGES'ˆvertex2=<size('VERTICES_LAYER2')]).

ctr_example(
    two_layer_edge_crossing,
    two_layer_edge_crossing(
        2,
        [[id-1,pos-1],[id-2,pos-2]],
        [[id-1,pos-3],[id-2,pos-1],[id-3,pos-2]],
    )).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

[[id-1,vertex1-2,vertex2-2],
 [id-2,vertex1-2,vertex2-3],
 [id-3,vertex1-1,vertex2-1])].

ctr_typical(
   two_layer_edge_crossing,
   [size('VERTICES_LAYER1')>1,
    size('VERTICES_LAYER2')>1,
    size('EDGES')>=size('VERTICES_LAYER1'),
    size('EDGES')>=size('VERTICES_LAYER2')]).

ctr_exchangeable(
   two_layer_edge_crossing,
   [args(
       [['NCROSS'],
         ['VERTICES_LAYER1','VERTICES_LAYER2'],
         ['EDGES'])),
    items('VERTICES_LAYER1',all),
    items('VERTICES_LAYER2',all)]).

ctr_derived_collections(
   two_layer_edge_crossing,
   [col(EDGES_EXTREMITIES-
        collection(layer1-dvar,layer2-dvar),
        [item(
           layer1-
           'EDGES'\'vertex1('VERTICES_LAYER1',pos,id),
           layer2-
           'EDGES'\'vertex2('VERTICES_LAYER2',pos,id))])].

ctr_graph(
   two_layer_edge_crossing,
   ['EDGES_EXTREMITIES'],
   2,
   ['CLIQUE'(<)>>
    collection(edges_extremities1,edges_extremities2)],
   [edges_extremities1\layer1<
    edges_extremities2\layer1#/\n    edges_extremities1\layer2>edges_extremities2\layer2#/\n    edges_extremities1\layer1>edges_extremities2\layer1#/\n    edges_extremities1\layer2<edges_extremities2\layer2],
   ['NARC'='NCROSS'],
   [])).

ctr_pure_functional_dependency(two_layer_edge_crossing,[]).
ctr_functional_dependency(two_layer_edge_crossing,1,[2,3,4]).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.368 two_orth_are_in_contact

◊ Meta-Data:

ctr_date(  
two_orth_are_in_contact,  
['20030820','20040530','20060819']).

ctr_origin(  
two_orth_are_in_contact,  
\cite{Roach84}, used for defining %c.,  
[orths_are_connected]).

ctr_types(  
two_orth_are_in_contact,  
['ORTHOTOPE'-collection(ori-dvar,siz-dvar,end-dvar)]).

ctr_arguments(  
two_orth_are_in_contact,  
['ORTHOTOPE1'-'ORTHOTOPE2', 'ORTHOTOPE2'-'ORTHOTOPE1']).

ctr_restrictions(  
two_orth_are_in_contact,  
[size('ORTHOTOPE')>0,  
  require_at_least(2,'ORTHOTOPE',[ori,siz,end]),  
  'ORTHOTOPE'`siz>0,  
  'ORTHOTOPE'`ori='< 'ORTHOTOPE'`end,  
  size('ORTHOTOPE1')=size('ORTHOTOPE2'),  
  orth_link_ori_siz_end('ORTHOTOPE1'),  
  orth_link_ori_siz_end('ORTHOTOPE2'))).

ctr_example(  
two_orth_are_in_contact,  
two_orth_are_in_contact(  
  [[ori-1,siz-3,end-4],[ori-5,siz-2,end-7]],  
  [[ori-3,siz-2,end-5],[ori-2,siz-3,end-5]]).

ctr_typical(two_orth_are_in_contact,[size('ORTHOTOPE')>1]).

ctr_exchangeable(  
two_orth_are_in_contact,  
[\text{args}([[\text{'ORTHOTOPE1'},\text{'ORTHOTOPE2'}]]),  
  \text{items_sync}('ORTHOTOPE1','ORTHOTOPE2',\text{all})]).

ctr_graph(  
two_orth_are_in_contact,
['ORTHOTOPE1', 'ORTHOTOPE2'],
2,
['PRODUCT' (=)>>collection(orthotope1, orthotope2)],
[orthotope1'end>orthotope2'ori,
 orthotope2'end>orthotope1'ori],
['NARC'=size('ORTHOTOPE1')-1],
[]).

ctr_graph(
  two_orth_are_in_contact,
  ['ORTHOTOPE1', 'ORTHOTOPE2'],
  2,
  ['PRODUCT' (=)>>collection(orthotope1, orthotope2)],
  [max(0,
   max(orthotope1'ori, orthotope2'ori) -
   min(orthotope1'end, orthotope2'end)) =
  0],
  ['NARC'=size('ORTHOTOPE1')],
  []).

ctr_eval(
  two_orth_are_in_contact,
  [automaton(two_orth_are_in_contact_a)]).

two_orth_are_in_contact_a(FLAG, ORTHOTOPE1, ORTHOTOPE2) :-
  length(ORTHOTOPE1, D1),
  length(ORTHOTOPE2, D2),
  D1>0,
  D2>0,
  D1=D2,
  collection(ORTHOTOPE1, [dvar, dvar_gteq(1), dvar]),
  collection(ORTHOTOPE2, [dvar, dvar_gteq(1), dvar]),
  get_attr1(ORTHOTOPE1, ORIS1),
  get_attr3(ORTHOTOPE1, ENDS1),
  check_lesseq(ORIS1, ENDS1),
  get_attr1(ORTHOTOPE2, ORIS2),
  get_attr3(ORTHOTOPE2, ENDS2),
  check_lesseq(ORIS2, ENDS2),
  eval(orth_link_ori_siz_end(ORTHOTOPE1)),
  eval(orth_link_ori_siz_end(ORTHOTOPE2)),
  two_orth_are_in_contact_signature(
    ORTHOTOPE1,
    ORTHOTOPE2,
    SIGNATURE),
  AUTOMATON=
  automaton(
SIGNATURE, _38731, SIGNATURE,
[source(s), sink(t)],
[arc(s,0,s), arc(s,1,t), arc(t,0,t)],
[[]],
[[]],
automaton_bool(FLAG, [0,1,2], AUTOMATON).

two_orth_are_in_contact_signature([], [], []). 

two_orth_are_in_contact_signature(
  [[ori-ORI1, siz-SIZ1, end-END1]|Q1],
  [[ori-ORI2, siz-SIZ2, end-END2]|Q2],
  [S|Ss]) :-
   S in 0..2,
   SIZ1#>0#/SIZ2#>0#/END1#>ORI2#/END2#>ORI1#<=>S#=0,
   SIZ1#>0#/SIZ2#>0#/END1#=ORI2#/END2#=ORI1#<=>S#=1,
   two_orth_are_in_contact_signature(Q1, Q2, Ss).
B.369  two_orth_column

◊ Meta-Data:

ctr_date(two_orth_column,['20030820']).

ctr_origin(two_orth_column,
    Used for defining %c.,
    [diffn_column]).

ctr_types(two_orth_column,
    ['ORTHOTOPE'-collection(ori-dvar,siz-dvar,end-dvar)]).

ctr_arguments(two_orth_column,
    ['ORTHOTOPE1'-'ORTHOTOPE',
    'ORTHOTOPE2'-'ORTHOTOPE',
    'DIM'-int]).

ctr_restrictions(two_orth_column,
    [size('ORTHOTOPE')>0,
    require_at_least(2,'ORTHOTOPE',[ori,siz,end]),
    'ORTHOTOPE'`siz]=0,
    'ORTHOTOPE'`ori=<'ORTHOTOPE'`end,
    size('ORTHOTOPE1')=size('ORTHOTOPE2'),
    orth_link_ori_siz_end('ORTHOTOPE1'),
    orth_link_ori_siz_end('ORTHOTOPE2'),
    'DIM'>0,
    'DIM'=<size('ORTHOTOPE1'))].

ctr_example(two_orth_column,
    two_orth_column([ [ori-1,siz-3,end-4],[ori-1,siz-1,end-2] ],
    [ [ori-4,siz-2,end-6],[ori-1,siz-3,end-4],
    1]).

ctr_typical(two_orth_column,[size('ORTHOTOPE')>1]).

ctr_exchangeable(two_orth_column,
    [args([['ORTHOTOPE1','ORTHOTOPE2'],['DIM']])]).
ctr_graph(
    two_orth_column,
    ['ORTHOTOPE1','ORTHOTOPE2'],
    [PRODUCT(=)>>collection(orthotope1,orthotope2)],
    [orthotope1ˆkey='DIM'#/
      orthotope1ˆori<orthotope2ˆend#/
      orthotope2ˆori<orthotope1ˆend#/
      orthotope1ˆsiz>0#/
      orthotope2ˆsiz>0#=>
      min(orthotope1ˆend,orthotope2ˆend)-
      max(orthotope1ˆori,orthotope2ˆori)=
      orthotope1ˆsiz#/
      orthotope1ˆsiz=orthotope2ˆsiz],
    ['NARC'=1],
    []).

ctr_eval(two_orth_column,[reformulation(two_orth_column_r)]).

two_orth_column_r(ORTHOTOPE1,ORTHOTOPE2,DIM) :-
    collection(ORTHOTOPE1,[dvar,dvar_gteq(0),dvar]),
    collection(ORTHOTOPE2,[dvar,dvar_gteq(0),dvar]),
    length(ORTHOTOPE1,DIM1),
    length(ORTHOTOPE2,DIM2),
    DIM1=DIM2,
    check_type(int(1,DIM1),DIM),
    get_attr1(ORTHOTOPE1,ORIS1),
    nth1(DIM,ORIS1,O1),
    get_attr2(ORTHOTOPE1,SIZS1),
    nth1(DIM,SIZS1,S1),
    get_attr3(ORTHOTOPE1,ENDS1),
    nth1(DIM,ENDS1,E1),
    get_attr1(ORTHOTOPE2,ORIS2),
    nth1(DIM,ORIS2,O2),
    get_attr2(ORTHOTOPE2,SIZS2),
    nth1(DIM,SIZS2,S2),
    get_attr3(ORTHOTOPE2,ENDS2),
    nth1(DIM,ENDS2,E2),
    O1#<E2#/\O2#<E1#/\S1#>0#/\S2#>0#=>
    min(E1,E2)-max(O1,O2)#=S1#/\S1#=S2.
B.370 two_orth_do_not_overlap

◊ Meta-Data:

ctr_date:
  two_orth_do_not_overlap,
  ['20030820','20040530','20060819']).

ctr_origin:
  two_orth_do_not_overlap,
  Used for defining %c.,
  [diffn]).

ctr_types:
  two_orth_do_not_overlap,
  ['ORTHOTOPE'-collection(ori-dvar,siz-dvar,end-dvar)]).

ctr_arguments:
  two_orth_do_not_overlap,
  ['ORTHOTOPE1'-'ORTHOTOPE','ORTHOTOPE2'-'ORTHOTOPE']).

ctr_restrictions:
  two_orth_do_not_overlap,
  [size('ORTHOTOPE')>0,
   require_at_least(2,'ORTHOTOPE',[ori,siz,end]),
   'ORTHOTOPE'¨siz>=0,
   'ORTHOTOPE'¨ori=<'ORTHOTOPE'¨end,
   size('ORTHOTOPE1')=size('ORTHOTOPE2'),
   orth_link_ori_siz_end('ORTHOTOPE1'),
   orth_link_ori_siz_end('ORTHOTOPE2'))].

ctr_example:
  two_orth_do_not_overlap,
  two_orth_do_not_overlap([ori-2,siz-2,end-4],[ori-1,siz-3,end-4],
  [ori-4,siz-4,end-8],[ori-3,siz-3,end-6])).

ctr_typical(two_orth_do_not_overlap,[size('ORTHOTOPE')>1]).

ctr_exchangeable:
  two_orth_do_not_overlap,
  [args([[‘ORTHOTOPE1’,'ORTHOTOPE2'])),
   items_sync(‘ORTHOTOPE1’,'ORTHOTOPE2',all),
   vals(['ORTHOTOPE1'¨siz],int (>=0)),>,dontcare,dontcare),
   vals(['ORTHOTOPE2'¨siz],int (>=0)),>,dontcare,dontcare])].
ctr_graph(
    two_orth_do_not_overlap,
    ['ORTHOTOPE1','ORTHOTOPE2'],
    2,
    ['SYMmetric_product' (=) >>
    collection(orthotope1,orthotope2)],
    [orthotope1^end=<orthotope2^ori#/orthotope1^siz=0],
    ['Narc'=1],
    ['Bipartite','No_loop']).

ctr_eval(
    two_orth_do_not_overlap,
    [automaton(two_orth_do_not_overlap_a)]).

two_orth_do_not_overlap_a(FLAG,ORTHOTOPE1,ORTHOTOPE2) :-
    length(ORTHOTOPE1,D1),
    length(ORTHOTOPE2,D2),
    D1>0,
    D2>0,
    D1=D2,
    collection(ORTHOTOPE1,[dvar,dvar_gteq(0),dvar]),
    collection(ORTHOTOPE2,[dvar,dvar_gteq(0),dvar]),
    get_attr1(ORTHOTOPE1,ORIS1),
    get_attr3(ORTHOTOPE1,ENDS1),
    check_lesseq(ORIS1,ENDS1),
    get_attr1(ORTHOTOPE2,ORIS2),
    get_attr3(ORTHOTOPE2,ENDS2),
    check_lesseq(ORIS2,ENDS2),
    eval(orth_link_ori_siz_end(ORTHOTOPE1)),
    eval(orth_link_ori_siz_end(ORTHOTOPE2)),
    two_orth_do_not_overlap_signature(
        ORTHOTOPE1,
        ORTHOTOPE2,
        SIGNATURE),
    AUTOMATON=
    automaton(
        SIGNATURE, 
        35847, 
        SIGNATURE, 
        [source(s),sink(t)],
        [arc(s,1,s),arc(s,0,t),arc(t,0,t),arc(t,1,t)],
        [],
        [],
        []),
    automaton_bool(FLAG,[0,1],AUTOMATON).
two_orth_do_not_overlap_signature([],[],[]).

two_orth_do_not_overlap_signature(
    [[ori-ORI1,siz-SIZ1,end-END1]|Q1],
    [[ori-ORI2,siz-SIZ2,end-END2]|Q2],
    [S|Ss]) :-
    SIZ1#>0#/SIZ2#>0#/END1#>ORI2#/END2#>ORI1#<=>=S,
    two_orth_do_not_overlap_signature(Q1,Q2,Ss).
B.371 two_orth_include

Meta-Data:

\[
\text{ctr\_date(two\_orth\_include, ['20030820', '20090524'])}.
\]

\[
\text{ctr\_origin(two\_orth\_include, Used for defining %c., [diffn\_include])}.
\]

\[
\text{ctr\_types(two\_orth\_include, ['ORTHOTOPE'-collection(ori-dvar, siz-dvar, end-dvar)])}.
\]

\[
\text{ctr\_arguments(two\_orth\_include, ['ORTHOTOPE1'-'ORTHOTOPE', 'ORTHOTOPE2'-'ORTHOTOPE', 'DIM'-int])}.
\]

\[
\text{ctr\_restrictions(two\_orth\_include, [size('ORTHOTOPE')>0, require_at_least(2, 'ORTHOTOPE', [ori, siz, end]), 'ORTHOTOPE'\^siz>0, 'ORTHOTOPE'\^ori='ORTHOTOPE'\^end, size('ORTHOTOPE1')=size('ORTHOTOPE2'), orth\_link\_ori\_siz\_end('ORTHOTOPE1'), orth\_link\_ori\_siz\_end('ORTHOTOPE2'), 'DIM'>0, 'DIM'=<size('ORTHOTOPE1')])}.
\]

\[
\text{ctr\_example(two\_orth\_include, two\_orth\_include([[ori-1, siz-3, end-4], [ori-1, siz-1, end-2]], [[ori-1, siz-2, end-3], [ori-2, siz-3, end-5]], 1))}.
\]

\[
\text{ctr\_typical(two\_orth\_include, [size('ORTHOTOPE')>1])}.
\]

\[
\text{ctr\_exchangeable(two\_orth\_include, [args([[ORTHOPE1', 'ORTHOTOPE2', ['DIM']]])])}.
\]
ctr_graph(
    two_orth_include,
    ['ORTHOTOPE1','ORTHOTOPE2'],
    2,
    ['PRODUCT' (=) >> collection(orthotope1,orthotope2)],
    [orthotope1ˆkey='DIM'#/
     orthotope1ˆori<orthotope2ˆend#/
     orthotope2ˆori<orthotope1ˆend#/
     orthotope1ˆsiz>0#/
     orthotope2ˆsiz>0#=>
     min(orthotope1ˆend,orthotope2ˆend)-
     max(orthotope1ˆori,orthotope2ˆori)=
     min(orthotope1ˆsiz,orthotope2ˆsiz)],
    ['NARC'=1],
    []).

ctr_eval(two_orth_include,[reformulation(two_orth_include_r)]).

two_orth_include_r(ORTHOTOPE1,ORTHOTOPE2,DIM) :-
    collection(ORTHOTOPE1,[dvar,dvar_gteq(0),dvar]),
    collection(ORTHOTOPE2,[dvar,dvar_gteq(0),dvar]),
    length(ORTHOTOPE1,DIM1),
    length(ORTHOTOPE2,DIM2),
    DIM1=DIM2,
    check_type(int(1,DIM1),DIM),
    get_attr1(ORTHOTOPE1,ORIS1),
    nth1(DIM,ORIS1,O1),
    get_attr2(ORTHOTOPE1,SIZS1),
    nth1(DIM,SIZS1,S1),
    get_attr3(ORTHOTOPE1,ENDS1),
    nth1(DIM,ENDS1,E1),
    get_attr1(ORTHOTOPE2,ORIS2),
    nth1(DIM,ORIS2,O2),
    get_attr2(ORTHOTOPE2,SIZS2),
    nth1(DIM,SIZS2,S2),
    get_attr3(ORTHOTOPE2,ENDS2),
    nth1(DIM,ENDS2,E2),
    O1#<E2#/\O2#<E1#/\S1#>0#/\S2#>0#=>
    min(E1,E2)-max(O1,O2)#=min(S1,S2).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.372 used_by

◊ META-DATA:

ctr_date(used_by,['20000128','20030820','20040530','20060820']).

ctr_origin(used_by,'N.˘Beldiceanu',[]).

ctr_arguments(
    used_by,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    used_by,
    [size('VARIABLES1')>=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var)]).

ctr_example(
    used_by,
    used_by(
        [[var-1],[var-9],[var-1],[var-5],[var-2],[var-1]],
        [[var-1],[var-1],[var-2],[var-5]]).

ctr_typical(
    used_by,
    [size('VARIABLES1')>1,
     range('VARIABLES1'`var)>1,
     size('VARIABLES2')>1,
     range('VARIABLES2'`var)>1]).

ctr_exchangeable(
    used_by,
    [items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(
         ['VARIABLES1'`var,'VARIABLES2'`var],
         int,
         =\_,
         all,
         dontcare))).

ctr_graph(
    used_by,
    ['VARIABLES1','VARIABLES2'],
    ...
2, 
[PRODUCT']>>collection(variables1,variables2)],
[variables1^var=variables2^var],
[for_all('CC','NSOURCE'>='NSINK'),
'NSINK'=size('VARIABLES2')],
[]).

ctr_eval(used_by,[reformulation(used_by_r)]).

ctr_contractible(used_by,[],'VARIABLES2',any).

ctr_extensible(used_by,[],'VARIABLES1',any).

ctr_aggregate(used_by,[],[union,union]).

used_by_r(VARIABLES1,VARIABLES2) :-
collection(VARIABLES1,[dvar]),
collection(VARIABLES2,[dvar]),
length(VARIABLES1,N1),
length(VARIABLES2,N2),
N1>=N2,
get_attr1(VARIABLES1,VARS1),
get_attr1(VARIABLES2,VARS2),
used_by_reified(VARS2,VARS1,VARS2).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

B.373 used_by_interval

◊ **Meta-Data:**

```prolog
ctr_date(used_by_interval, ['20030820', '20060820']).
ctr_origin(used_by_interval, 'Derived from %c.', [used_by]).
ctr_arguments(
    used_by_interval,
    ['VARIABLES1' - collection(var-dvar),
     'VARIABLES2' - collection(var-dvar),
     'SIZE_INTERVAL' - int]).
ctr_restrictions(
    used_by_interval,
    [size('VARIABLES1') >= size('VARIABLES2'),
     required('VARIABLES1', var),
     required('VARIABLES2', var),
     'SIZE_INTERVAL' > 0]).
ctr_example(
    used_by_interval,
    used_by_interval(
        [[var-1], [var-9], [var-1], [var-8], [var-6], [var-2]],
        [[var-1], [var-0], [var-7], [var-7]],
        3)).
ctr_typical(
    used_by_interval,
    [size('VARIABLES1') > 1,
     range('VARIABLES1' ^ var) > 1,
     size('VARIABLES2') > 1,
     range('VARIABLES2' ^ var) > 1,
     'SIZE_INTERVAL' > 1,
     'SIZE_INTERVAL' < range('VARIABLES1' ^ var),
     'SIZE_INTERVAL' < range('VARIABLES2' ^ var)]).
ctr_exchangeable(
    used_by_interval,
    [items('VARIABLES1', all),
     items('VARIABLES2', all),
     vals(
         ['VARIABLES1' ^ var],
         intervals('SIZE_INTERVAL'),
         =,
         intervals('SIZE_INTERVAL'))].
```


dontcare, don'tcare),
vals(
dontcare, don'tcare),

ctr_graph(used_by_interval,
['VARIABLES1','VARIABLES2'],
2,
['PRODUCT']>>collection(vables1,variables2)],
[variables1\'var'/'SIZE_INTERVAL'=
variables2\'var'/'SIZE_INTERVAL'],
[for_all('CC','NSOURCE'='NSINK'),
 'NSINK'=size('VARIABLES2')],
[]).

ctr_eval(used_by_interval,[reformulation(used_by_interval_r)]).

ctr_contractible(used_by_interval,[],'VARIABLES2',any).
ctr_extensible(used_by_interval,[],'VARIABLES1',any).
ctr_aggregate(used_by_interval,[],[union,union,id]).

used_by_interval_r(VARIABLES1,VARIABLES2,SIZE_INTERVAL) :-
collection(VARIABLES1,[dvar]),
collection(VARIABLES2,[dvar]),
length(VARIABLES1,N1),
length(VARIABLES2,N2),
N1>=N2,
integer(SIZE_INTERVAL),
SIZE_INTERVAL>0,
get_attr1(VARIABLES1,VARS1),
get_attr1(VARIABLES2,VARS2),
gen_quotient(VARS1,SIZE_INTERVAL,QUOTVARS1),
gen_quotient(VARS2,SIZE_INTERVAL,QUOTVARS2),
used_by_reified(QUOTVARS2,QUOTVARS1,QUOTVARS2).
B.374 used_by_modulo

◊ **META-DATA:**

```prolog
ctr_date(used_by_modulo, ['20030820', '20060820']).
ctr_origin(used_by_modulo, 'Derived from %c.', [used_by]).
ctr_arguments(used_by_modulo, [used_by_modulo,
                               'VARIABLES1'-collection(var-dvar),
                               'VARIABLES2'-collection(var-dvar),
                               'M'-int]).
ctr_restrictions(used_by_modulo, [size('VARIABLES1')>=size('VARIABLES2'),
                                   required('VARIABLES1', var),
                                   required('VARIABLES2', var),
                                   'M'>0]).
ctr_example(used_by_modulo, used_by_modulo([var-1], [var-9], [var-4], [var-5], [var-2], [var-1],
                                          [var-7], [var-1], [var-2], [var-5], 3)).
ctr_typical(used_by_modulo, [size('VARIABLES1')>1,
                             range('VARIABLES1'\^var)>1,
                             size('VARIABLES2')>1,
                             range('VARIABLES2'\^var)>1,
                             'M'>1,
                             'M'<maxval('VARIABLES1'\^var),
                             'M'<maxval('VARIABLES2'\^var)]).
ctr_exchangeable(used_by_modulo, [items('VARIABLES1', all),
                                  items('VARIABLES2', all),
                                  vals(['VARIABLES1'\^var], mod('M'), =, dontcare, dontcare),
                                  vals(['VARIABLES2'\^var], mod('M'), =, dontcare, dontcare)]).
ctr_graph(
```
used_by_modulo,
['VARIABLES1','VARIABLES2'],
2,
['PRODUCT'>>collection(variables1,variables2)],
[variables1\var mod 'M'=variables2\var mod 'M'],
[for_all('CC','NSOURCE'='NSINK'),
 'NSINK'=size('VARIABLES2')],
[]).

ctr_eval(used_by_modulo,[reformulation(used_by_modulo_r)]).

ctr_contractible(used_by_modulo,[],'VARIABLES2',any).

ctr_extensible(used_by_modulo,[],'VARIABLES1',any).

ctr_aggregate(used_by_modulo,[],[union,union,id]).

used_by_modulo_r(VARIABLES1,VARIABLES2,M) :-
collection(VARIABLES1,[dvar]),
collection(VARIABLES2,[dvar]),
length(VARIABLES1,N1),
length(VARIABLES2,N2),
N1>=N2,
integer(M),
M>0,
get_attr1(VARIABLES1,VARS1),
get_attr1(VARIABLES2,VARS2),
gen_remainder(VARS1,M,REMVARS1),
gen_remainder(VARS2,M,REMVARS2),
used_by_reified(REMVARS2,REMVARS1,REMVARS2).
B.375 **used_by_partition**

◊ **Meta-Data:**

```prolog
ctr_date(used_by_partition,['20030820','20060820']).
ctr_origin(used_by_partition,'Derived from %c.',[used_by]).
ctr_types(used_by_partition,['VALUES'-collection(val-int)]).

ctr_arguments(used_by_partition,
               ['VARIABLES1'-collection(var-dvar),
               'VARIABLES2'-collection(var-dvar),
               'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(used_by_partition,
                 [size('VALUES')>=1,
                  required('VALUES',val),
                  distinct('VALUES',val),
                  size('VARIABLES1')>=size('VARIABLES2'),
                  required('VARIABLES1',var),
                  required('VARIABLES2',var),
                  required('PARTITIONS',p),
                  size('PARTITIONS')>=2]).

ctr_example(used_by_partition,
             used_by_partition(
               [[var-1],[var-9],[var-1],[var-6],[var-2],[var-3]],
               [[var-1],[var-3],[var-6],[var-6]],
               [[p-[[val-1],[val-3]]],
                [p-[[val-4]]],
                [p-[[val-2],[val-6]]])).

ctr_typical(used_by_partition,
            [size('VARIABLES1')>1,
             range('VARIABLES1'\^var)>1,
             size('VARIABLES2')>1,
             range('VARIABLES2'\^var)>1,
             size('VARIABLES1')>size('PARTITIONS'),
             size('VARIABLES2')>size('PARTITIONS')]).
```

ctr_exchangeable
used_by_partition,
[items('VARIABLES1',all),
items('VARIABLES2',all),
items('PARTITIONS',all),
items('PARTITIONS'p,all),
vals(
  ['VARIABLES1'\'var],
  part('PARTITIONS'),
  =,
  dontcare,
  dontcare),
vals(
  ['VARIABLES2'\'var],
  part('PARTITIONS'),
  =,
  dontcare,
  dontcare)).

ctr_graph(
  used_by_partition,
  ['VARIABLES1','VARIABLES2'],
  2,
  ['PRODUCT'>>collection(variables1,variables2)],
  [in_same_partition(
    variables1\'var,
    variables2\'var,
    PARTITIONS)],
  [for_all('CC','NSOURCE'='NSINK'),
   'NSINK'=size('VARIABLES2')],
  []).

ctr_eval(
  used_by_partition,
  [reformulation(used_by_partition_r)]).

ctr_aggregate(used_by_partition,[],[union,union,id]).

ctr_contractible(used_by_partition,[],'VARIABLES2',any).

ctr_extensible(used_by_partition,[],'VARIABLES1',any).

used_by_partition_r(VARIABLES1,VARIABLES2,PARTITIONS) :-
collection(VARIABLES1,[dvar]),
collection(VARIABLES2,[dvar]),
collection(PARTITIONS,[col_len_gteq(1,[int])]),
length(VARIABLES1,N1),
length(VARIABLES2,N2),
N1>=N2,
get_attr1(VARIABLES1,VARS1),
get_attr1(VARIABLES2,VARS2),
get_col_attr1(PARTITIONS,1,PVALS),
flatten(PVALS,VALS),
all_different(VALS),
length(PARTITIONS,P),
P>1,
length(PVALS,LPVALS),
LPVALS1 is LPVALS+1,
get_partition_var(VARS1,PVALS,PVARS1,LPVALS1,0),
get_partition_var(VARS2,PVALS,PVARS2,LPVALS1,0),
used_by_reified(PVARS2,PVARS1,PVARS2).
B.376 uses

◊ **META-DATA:**

```prolog
ctr_date(uses, ['20050917', '20060820']).

ctr_origin(uses, 
cite{BessiereHebrardHnichKiziltanWalsh05IJCAI}, []).

ctr_arguments(uses, 
['VARIABLES1'-collection(var-dvar), 'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(uses, 
[min(1,size('VARIABLES1'))>=min(1,size('VARIABLES2'))], 
required('VARIABLES1', var), 
required('VARIABLES2', var)).

ctr_example(uses, 
uses( 
[[var-3], [var-3], [var-4], [var-6]], 
[[var-3], [var-4], [var-4], [var-4], [var-4]]).

ctr_typical(uses, 
[size('VARIABLES1')>1, 
range('VARIABLES1'~var)>1, 
size('VARIABLES2')>1, 
range('VARIABLES2'~var)>1, 
size('VARIABLES1')=<size('VARIABLES2'))].

ctr_exchangeable(uses, 
[items('VARIABLES1', all), 
items('VARIABLES2', all), 
vals( 
['VARIABLES1'~var,'VARIABLES2'~var], 
int, 
=\=, 
all, 
dontcare)]).```
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

ctr_graph(
uses,
[‘VARIABLES1’,'VARIABLES2’],
2,
[‘PRODUCT’>>collection(variables1,variables2)],
[variables1^var=variables2^var],
[‘NSINK’=size(‘VARIABLES2’)],
[‘ACYCLIC’,‘BIPARTITE’,‘NO_LOOP’]).

ctr_eval(uses,[reformulation(uses_r)]).

ctr_contractible(uses,[],’VARIABLES2’,any).

ctr_extensible(uses,[],’VARIABLES1’,any).

ctr_aggregate(uses,[],[union,union]).

uses_r(VARIABLES1,VARIABLES2) :-
collection(VARIABLES1,[dvar]),
collection(VARIABLES2,[dvar]),
length(VARIABLES1,L1),
length(VARIABLES2,L2),
M1 is min(1,L1),
M2 is min(1,L2),
M1>=M2,
get_attr1(VARIABLES1,VARS1),
get_attr1(VARIABLES2,VARS2),
uses1(VARS2,VARS1).

uses1([],_35047).

uses1([VAR2|R],VARS1) :-
uses2(VARS1,VAR2,TERM),
call(TERM),
uses1(R,VARS1).

uses2([],_35047,0).

uses2([VAR1|R],VAR2,VAR2#=VAR1\$/S) :-
uses2(R,VAR2,S).
B.377 valley

◊ **META-DATA:**

```prolog
ctr_date(valley,['20040530']).

ctr_origin(valley,'Derived from %c.',[inflexion]).

ctr_arguments(
    valley,
    ['N'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    valley,
    ['N'>=0,
     2*N=<max(size('VARIABLES')-1,0),
     required('VARIABLES',var)]).

ctr_example(
    valley,
    valley(1,
    [[var-1],
     [var-1],
     [var-4],
     [var-8],
     [var-8],
     [var-2],
     [var-7],
     [var-1]])).

ctr_typical(
    valley,
    [size('VARIABLES')>2,range('VARIABLES'\^var)>1]).

ctr_exchangeable(
    valley,
    [items('VARIABLES',reverse),translate(['VARIABLES'\^var])]).

ctr_eval(valley,[automaton(valley_a)]).

ctr_contractible(valley,['N'=0],'VARIABLES',any).

valley_a(FLAG,N,VARIABLES) :-
    check_type(dvar_gteq(0),N),
    collection(VARIABLES,[dvar]),
```
length(VARIABLES,L),
MAX is max(L-1,0),
2*N#=<MAX,
valley_signature(VARIABLES,SIGNATURE),
automaton(
    SIGNATURE,
    _19215,
    SIGNATURE,
    [source(s),sink(u),sink(s)],
    [arc(s,0,s),
     arc(s,1,s),
     arc(s,2,u),
     arc(u,0,s,[C+1]),
     arc(u,1,u),
     arc(u,2,u)],
    [C],
    [0],
    [COUNT]),
    COUNT#=N#<=>FLAG.
valley_signature([],[]).
valley_signature([_17449],[]) :- !.
valley_signature([var-VAR1],Ss) :-
    S in 0..2,
    VAR1#<VAR2#<=>S#=0,
    VAR1#=VAR2#<=>S#=1,
    VAR1#>VAR2#<=>S#=2,
    valley_signature([var-VAR2],Ss).
B.378 vec_eq_tuple

◊ Meta-Data:

\[
\text{ctr\_date(vec_eq_tuple,} \{\text{’20030820’,’20060820’}\}).
\]

\[
\text{ctr\_origin(vec_eq_tuple,’Used for defining %c.’,in\_relation]).}
\]

\[
\text{ctr\_arguments(}
\]
\[
\text{vec_eq_tuple,}
\]
\[
\{\text{’VARIABLES’–collection(var-dvar),}
\]
\[
\text{’TUPLE’–collection(val-int)].}
\]

\[
\text{ctr\_restrictions(}
\]
\[
\text{vec_eq_tuple,}
\]
\[
\{\text{required(’VARIABLES’,var),}
\]
\[
\text{required(’TUPLE’,val),}
\]
\[
\text{size(’VARIABLES’)=size(’TUPLE’)].}
\]

\[
\text{ctr\_example(}
\]
\[
\text{vec_eq_tuple,}
\]
\[
\text{vec_eq_tuple(}
\]
\[
\{[[\text{var-5}],[\text{var-3}],[\text{var-3}]],
\]
\[
[[\text{val-5}],[\text{val-3}],[\text{val-3}]]\}].}
\]

\[
\text{ctr\_typical(}
\]
\[
\text{vec_eq_tuple,}
\]
\[
\{\text{size(’VARIABLES’)>1,}
\]
\[
\text{range(’VARIABLES’}^{\text{var}}\text{>1,}
\]
\[
\text{range(’TUPLE’}^{\text{val}}\text{>1]}].}
\]

\[
\text{ctr\_exchangeable(}
\]
\[
\text{vec_eq_tuple,}
\]
\[
\{\text{args(}[[\text{’VARIABLES’},’TUPLE’]],
\]
\[
\text{items\_sync(’VARIABLES’,’TUPLE’,all)]].}
\]

\[
\text{ctr\_graph(}
\]
\[
\text{vec_eq_tuple,}
\]
\[
\{\text{’VARIABLES’,’TUPLE’},}
\]
\[
2,
\]
\[
\{\text{’PRODUCT’}()\gg\text{collection(variables,tuple)},
\]
\[
\text{variables}^{\text{var}}=\text{tuple}^{\text{val}},
\]
\[
\text{’NARC’}=\text{size(’VARIABLES’)},
\]
\[
[]\}].}
\]

\[
\text{ctr\_eval(vec_eq_tuple,} \{\text{reformulation(vec_eq_tuple_r)]}.}
\]
ctr_contractible(vec_eq_tuple,[],['VARIABLES','TUPLE'],any).

vec_eq_tuple_r(VARIABLES,TUPLE) :-
  collection(VARIABLES,[dvar]),
  collection(TUPLE,[int]),
  length(VARIABLES,N),
  length(TUPLE,M),
  N=M,
  get_attr1(VARIABLES,VARS),
  get_attr1(TUPLE,VALS),
  vec_eq_tuple1(VARS,VALS).

vec_eq_tuple1([],[]).

vec_eq_tuple1([VAR|R],[VAL|S]) :-
  VAR#=VAL,
  vec_eq_tuple1(R,S).
B.379 visible

◊ **META-DATA:**

```prolog
ctr_predefined(visible).

ctr_date(visible, ['20071013']).

ctr_origin(
  visible,
  Extension of \textit{accessibility} parameter of \%c., [diffn]).

ctr_types(
  visible,
  ['VARIABLES'-collection(v-dvar),
   'INTEGERS'-collection(v-int),
   'POSITIVES'-collection(v-int),
   'DIMDIR'-collection(dim-int,dir-int)]).

ctr_arguments(
  visible,
  ['K'-int,
   'DIMS'-sint,
   'FROM'-'DIMDIR',
   OBJECTS-
     collection(
       oid-int,
       sid-dvar,
       x-'VARIABLES',
       start-dvar,
       duration-dvar,
       end-dvar),
   SBOXES-
     collection(
       sid-int,
       t-'INTEGERS',
       l-'POSITIVES',
       f-'DIMDIR'))).

ctr_restrictions(
  visible,
  [size('VARIABLES')\geq 1,
   size('INTEGERS')\geq 1,
   size('POSITIVES')\geq 1,
   required('VARIABLES',v),
   ...])
```
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

size('VARIABLES')='K',
required('INTEGERS',v),
size('INTEGERS')='K',
required('POSITIVES',v),
size('POSITIVES')='K',
'POSITIVES'ˆv>0,
required('DIMDIR',[dim,dir]),
size('DIMDIR')>0,
size('DIMDIR')='K'+'K',
distinct('DIMDIR',[]),
'DIMDIR'ˆdim>=0,
'DIMDIR'ˆdim<='K',
'DIMDIR'ˆdir>=0,
'DIMDIR'ˆdir<1,
'K'='0,
'DIMS'='0,
'DIMS'='K',
distinct('OBJECTS',oid),
required('OBJECTS',{oid,sid,x}),
require_at_least(2,'OBJECTS',[start,duration,end]),
'OBJECTS'ˆoid='1,
'OBJECTS'ˆoid='size('OBJECTS'),
'OBJECTS'ˆsid='1,
'OBJECTS'ˆsid='size('SBOXES'),
'OBJECTS'ˆduration='0,
size('SBOXES')='1,
required('SBOXES',{sid,t,l}),
'SBOXES'ˆsid='1,
'SBOXES'ˆsid='size('SBOXES'),
do_not_overlap('SBOXES')).

ctr_example(
  visible,
  [visible(
    2,
    {0,1},
    [[dim-0,dir-1]],
    [oid-1,
     sid-1,
     x-[[v-1],[v-2]],
     start-8,
     duration-8,
     end-16],
    [oid-2,
     sid-2,
     x-[[v-4],[v-2]],
     ...])
start-1,
duration-15,
end-16]],
[[sid-1,
t-[[v-0],[v-0]],
il-[[v-1],[v-2]],
fonf-[[dim-0,dir-1]],
[sid-2,
t-[[v-0],[v-0]],
il-[[v-2],[v-3]],
fonf-[[dim-0,dir-1]]],
visible(2,
{0,1},
[[dim-0,dir-1]],
[[oid-1,
sid-1,
x-[[v-1],[v-2]],
start-1,
duration-8,
end-9],
[oid-2,
sid-2,
x-[[v-4],[v-2]],
start-1,
duration-15,
end-16]],
[[sid-1,
t-[[v-0],[v-0]],
il-[[v-1],[v-2]],
fonf-[[dim-0,dir-1]],
[sid-2,
t-[[v-0],[v-0]],
il-[[v-2],[v-3]],
fonf-[[dim-0,dir-1]]],
visible(2,
{0,1},
[[dim-0,dir-1]],
[[oid-1,
sid-1,
x-[[v-1],[v-1]],
start-1,
duration-15,
end-16],
[oid-2,
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

sid-2,
  x-[[v-2],[v-2]],
  start-6,
  duration-6,
  end-12],
[[sid-1,
  t-[[v-0],[v-0]],
  l-[[v-1],[v-2]],
  f-[[dim-0,dir-1]]],
[sid-2,
  t-[[v-0],[v-0]],
  l-[[v-2],[v-3]],
  f-[[dim-0,dir-1]]]],
visible(
  2,
  {0,1},
  [[dim-0,dir-1]],
[[oid-1,
  sid-1,
  x-[[v-4],[v-1]],
  start-1,
  duration-8,
  end-9],
[oid-2,
  sid-2,
  x-[[v-1],[v-2]],
  start-1,
  duration-15,
  end-16]],
[[sid-1,
  t-[[v-0],[v-0]],
  l-[[v-1],[v-2]],
  f-[[dim-0,dir-1]]],
[sid-2,
  t-[[v-0],[v-0]],
  l-[[v-2],[v-3]],
  f-[[dim-0,dir-1]]]],
visible(
  2,
  {0},
  [[dim-0,dir-1]],
[[oid-1,
  sid-1,
  x-[[v-2],[v-1]],
  start-1,
  duration-8,
end-9],
oeid-2,
sid-2,
x-[[v-4],[v-3]],
start-1,
duration-15,
end-16]],
[[sid-1,
t-[[v-0],[v-0]],
l-[[v-1],[v-2]],
f-[[dim-0,dir-1]]],
[sid-2,
t-[[v-0],[v-0]],
l-[[v-2],[v-2]],
f-[[dim-0,dir-1]]]]).

ctr_typical(visible,[size('OBJECTS')>1]).

ctr_exchangeable(
    visible,
    [items('OBJECTS',all),items('SBOXES',all)]).
B.380  weighted_partial_alldiff

◊ Meta-Data:

ctr_date(
  weighted_partial_alldiff,
  ['20040814','20060820','20090503']).

ctr_origin(
  weighted_partial_alldiff,
  \cite[page 71]{Thiel04},
  []).

ctr_synonyms(
  weighted_partial_alldiff,
  [weighted_partial_alldifferent, 
   weighted_partial_alldistinct, 
   wpa]).

ctr_arguments(
  weighted_partial_alldiff,
  ['VARIABLES'-collection(var-dvar),
   'UNDEFINED'-int,
   'VALUES'-collection(val-int,weight-int),
   'COST'-dvar]).

ctr_restrictions(
  weighted_partial_alldiff,
  [required('VARIABLES',var),
   size('VALUES')>0,
   required('VALUES',[val,weight]),
   in_attr('VARIABLES',var,'VALUES',val),
   distinct('VALUES',val)]).

ctr_example(
  weighted_partial_alldiff,
  weighted_partial_alldiff(
    [[var-4],[var-0],[var-1],[var-2],[var-0],[var-0]],
    0,
    [[val-0,weight-0],
    [val-1,weight-2],
    [val-2,weight- -1],
    [val-4,weight-7],
    [val-5,weight- -8],
    [val-6,weight-2]],
    8)).
ctr_typical(
    weighted_partial_alldiff,
    [size('VARIABLES')>0,
     atleast(1,'VARIABLES','UNDEFINED'),
     size('VARIABLES')=<size('VALUES')+2]).

ctr_exchangeable(
    weighted_partial_alldiff,
    [items('VARIABLES',all),
     items('VALUES',all),
     vals(
      ['VARIABLES'\var,'VALUES'\val],
     int(=\\ &=('UNDEFINED'),
    ,
     all,
     dontcare)]).

ctr_graph(
    weighted_partial_alldiff,
    ['VARIABLES','VALUES'],
    2,
    ['PRODUCT'>>collection(variables,values)],
    [variables\var='UNDEFINED',variables\var=values\val],
    ['MAX_ID'<1,'SUM'('VALUES',weight)='COST'],
    []).

ctr_eval(
    weighted_partial_alldiff,
    [reformulation(weighted_partial_alldiff_r)]).

ctr_functional_dependency(weighted_partial_alldiff,4,[1,3]).

weighted_partial_alldiff_r(VARIABLES,UNDEFINED,VALUES,COST) :-
collection(VARIABLES,[dvar]),
integer(UNDEFINED),
collection(VALUES,[int,int]),
length(VALUES,N),
N>0,
check_type(dvar,COST),
get_attr1(VARIABLES,VARS),
get_attr1(VALUES,VALS),
get_attr2(VALUES,WEIGHTS),
all_different(VALS),
get_proj1(VALUES,CVALS),
weighted_partial_alldiff0(VALS,WEIGHTS,UNDEFINED),
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

weighted_partial_alldiff1(VARS,CVALS),
weighted_partial_alldiff2(VARS,UNDEFINED),
weighted_partial_alldiff4(VALS,WEIGHTS,VARS,TERM),
call(COST#=TERM).

weighted_partial_alldiff0([UNDEFINED|_44173],
[0|_44177],
UNDEFINED) :-
!.

weighted_partial_alldiff0([_V|R],[S|_44170],UNDEFINED) :-
weighted_partial_alldiff0(R,S,UNDEFINED).

weighted_partial_alldiff1([],_44167).

weighted_partial_alldiff1([VAR|R],VALUES) :-
eval(VAR in VALUES),
weighted_partial_alldiff1(R,VALUES).

weighted_partial_alldiff2([],_44167).

weighted_partial_alldiff2([_44171],_44167) :-
!

weighted_partial_alldiff2([VAR|R],UNDEFINED) :-
weighted_partial_alldiff3(R,VAR,UNDEFINED),
weighted_partial_alldiff2(R,UNDEFINED).

weighted_partial_alldiff3([],_44167,_44168).

weighted_partial_alldiff3([VAR|R],VAR,UNDEFINED) :-
VAR#\=VAR#/VAR#=UNDEFINED,
weighted_partial_alldiff3(R,VAR,UNDEFINED).

weighted_partial_alldiff4([],[],_44168,0).

weighted_partial_alldiff4([VAL|R],[WEIGHT|S],VARS,WEIGHT*B+T) :-
weighted_partial_alldiff5(VARS,VAL,WEIGHT,TERM),
call(B#<=>TERM),
weighted_partial_alldiff4(R,S,VARS,T).

weighted_partial_alldiff5([],_44167,_44168,0).

weighted_partial_alldiff5([VAR|R],VAL,WEIGHT,VAR#=VAL#/T) :-
weighted_partial_alldiff5(R,VAL,WEIGHT,T).
B.381 xor

◊ Meta-Data:

ctr_date(xor, ['20051226']).

ctr_origin(xor, 'Logic', []).

ctr_synonyms(xor, [rel]).

ctr_arguments(
    xor,
    ['VAR'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    xor,
    ['VAR'>=0,
     'VAR'=<1,
     size('VARIABLES')=2,
     required('VARIABLES', var),
     'VARIABLES'-'var'>=0,
     'VARIABLES'-'var'<1]).

ctr_example(
    xor,
    [xor(0, [[var-0], [var-0]]),
     xor(1, [[var-0], [var-1]]),
     xor(1, [[var-1], [var-0]]),
     xor(0, [[var-1], [var-1]])].

ctr_exchangeable(xor, [items('VARIABLES', all)]).

ctr_eval(xor, [automaton(xor_a)]).

ctr_pure_functional_dependency(xor, []).

ctr_functional_dependency(xor, 1, [2]).

xor_a(FLAG, VAR, VARIABLES) :-
    check_type(dvar(0,1), VAR),
    collection(VARIABLES, [dvar(0,1)]),
    length(VARIABLES, 2),
    get_attr1(VARIABLES, LIST),
    append([VAR], LIST, LIST VARIABLES),
    AUTOMATON= automaton(
LIST_VARIABLES,  
_19859,  
LIST_VARIABLES,  
[source(s), sink(t)],  
[arc(s,0,i),  
 arc(s,1,j),  
 arc(i,0,k),  
 arc(i,1,l),  
 arc(j,0,l),  
 arc(j,1,k),  
 arc(k,0,t),  
 arc(l,1,t)],  
[[]],  
[[]],  
[]),  
automaton_bool(FLAG, [0,1], AUTOMATON).
B.382 Utilities

:- use_module(library(lists)).
:- use_module(library(ordsets)).
:- use_module(library(clpfd)).
:- use_module(library(plunit)).
:- use_module(library(trees)).

% to use when everything is not necessarily ground
eval(Ctr) :-
Ctr =..[Name|Args],
ctr_eval(Name, Methods),
(member(   builtin(Pred), Methods) -> Goal =..[Pred|Args]
;member(  automaton(Pred), Methods) -> Goal =..[Pred,1|Args]
;member(   automata(Pred), Methods) -> Goal =..[Pred|Args] % defined by a
;member(reformulation(Pred), Methods) -> Goal =..[Pred|Args]
;member(   logic(Pred), Methods) -> Goal =..[Pred|Args]
), !,
call(Goal).

% to use when everything is ground: call first a checker if it exist (since
checker(Ctr) :-
Ctr =..[Name|Args],
ctr_eval(Name, Methods),
(member(   checker(Pred), Methods) -> Goal =..[Pred|Args]
;member(   builtin(Pred), Methods) -> Goal =..[Pred|Args]
;member(   automaton(Pred), Methods) -> Goal =..[Pred,1|Args]
;member(   automata(Pred), Methods) -> Goal =..[Pred|Args] % defined by a
;member(reformulation(Pred), Methods) -> Goal =..[Pred|Args]
;member(   logic(Pred), Methods) -> Goal =..[Pred|Args]
), !,
call(Goal).

% to use to evaluate the negation of a constraint, use:
% . reified automaton or
% . reified constraint for pure functional dependency or
% . existing constraint of the catalog with exactly same arguments
neg_eval(Ctr) :-
Ctr =..[Name|Args],
ctr_pure_functional_dependency(Name),
!,
NegCtr =..[Name,0|Args],
reified_ctr_pure_functional_dependency(NegCtr),
!.
neg_eval(Ctr) :-
Ctr =..[Name|Args],
ctr_eval(Name, Methods),
(member(automaton(Pred), Methods) -> Goal =..[Pred,0|Args]
),
!,
call(Goal),
!.

neg_eval(Ctr) :-
Ctr =..[Name|Args],
ctr_see_also(Name, Links),
member(link(negation, NegName, _, _), Links),
!,
NegCtr =..[NegName|Args],
eval(NegCtr),
!.

% reified version for constraints that can be described in term of pure functional dependencies.
reified_ctr_pure_functional_dependency(Ctr) :-
Ctr =..[Name,Bool|Args],
ctr_pure_functional_dependency(Name),
ctr_arguments(Name, ListArgsCtr),
findall(F, ctr_functional_dependency(Name,F,_,_), LF),
sort(LF, SLF),
length(Args, NArgs),
build_args_ctr(1, NArgs, Args, ListArgsCtr, SLF, NewArgs, AndExpr),
NewCtr =..[Name|NewArgs],
eval(NewCtr),
call(AndExpr #=> Bool).

build_args_ctr(I, N, [], [], [], [], 1) :-
I > N,
!.

build_args_ctr(I, N, [Arg|RArg], [ArgType|RArgType], [F|RF], [Var|R], Var#=Arg #/ S) :-
I =< N,
I = F,
ArgType = _-_dvar,
!,
Var in -1000000..1000000,
Il is I+1,
build_args_ctr(Il, N, RArg, RArgType, RF, R, S).

build_args_ctr(I, N, [Arg|RArg], [__|RArgType], LF, [Arg|R], S) :-
I =< N,
!,
Il is I+1,
build_args_ctr(Il, N, RArg, RArgType, LF, R, S).

% depending on the flag, call positive automaton, or computes negative automaton and
automaton_bool(1, _ALPHABET, POS_AUTOMATON) :- 
    !,
call(POS_AUTOMATON).
automaton_bool(0, ALPHABET, POS_AUTOMATON) :- 
    negaut(POS_AUTOMATON, ALPHABET, NEG_AUTOMATON, NEG_AUXILIARY),
call(NEG_AUTOMATON),
call(NEG_AUXILIARY).

% An utility for negating an automaton (WARNING: only valid if everything expressed
% negaut(+PosAutomaton, +Alphabet, -NegAutomaton, -AuxConstraint)
% PosAutomaton: automaton/8 constraint
% Alphabet: list of atom
% NegAutomaton: automaton/8 constraint
% AuxConstraint: constraint
% Semantics:
% - (NegAutomaton, AuxConstraint) expresses the negation of PosAutomaton.
% Synopsis:
% - If necessary, add a nonsink state 'fail', and:
%  * for every letter A of the alphabet: add an arc from 'fail' over A to 'fail';
%  * for every state S and letter A of the alphabet, if there is no outgoing arc from S over A, add an arc from S over A to 'fail'.
% - If the automaton is counter-free, compute NegAutomaton by swapping sinks and nonsinks. AuxConstraint is 'true'.
% Otherwise with counters [C1,...,Cn]:
%  * Suppose that the final counter values are [V1,...,Vn].
%  * Add a first counter C0 so that C0=0 iff the original automaton stops in a sink state.
%  * Convert arcs as follows:
%    arc(S1,A,S2) --> arc(S1,A,S2,[0,C1,...,Cn]) if S2 is sink
%    arc(S1,A,S2) --> arc(S1,A,S2,[1,C1,...,Cn]) if S2 is nonsink
%    arc(S1,A,S2,[Y1,...,Yn]) --> arc(S1,A,S2,[0,Y1,...,Yn]) if S2 is sink
%    arc(S1,A,S2,[Y1,...,Yn]) --> arc(S1,A,S2,[1,Y1,...,Yn]) if S2 is nonsink
%  * The counters for arcs with conditions are augmented similarly.
%  * For every arc with a condition:
%    arc(S1,A,_,(P1 -> Q1 ; ... ; Pm -> Qm))
%    such that (P1 \\ ... \\ Pm) could be false, add an arc:
%    arc(S1,A,fail,((#\P1 \\ ... \\ #\ Pm) -> [1,C1,...,Cn])
%  * Compute NegAutomaton by making all states sinks.
%  * Let the final counter values of NegAutomaton be [X0,X1,...,Xn].
negaut(PosAut, Alphabet1, NegAut, Aux) :-
    PosAut = automaton(Args, Arg, Signature,
                      PosSourcesSinks, PosArcs,
                      Counters, Initial, Final),
    NegAut = automaton(Args, Arg, Signature,
                      NegSourcesSinks, NegArCs,
                      NegCounters, NegInitial, NegFinal),
    ( foreach(SS1, PosSourcesSinks),
      fromto(Sources1, Sources1b, Sources1c, []),
      fromto(Sinks1, Sinks1b, Sinks1c, [])
    do ( SS1 = source(SS2) -> Sources1b = [SS2|Sources1c], Sinks 1b = Sinks1c
         ; SS1 = sink(SS2) -> Sinks1b = [SS2|Sinks1c], Sources1b = Sources1c
         )
    ),
    ( foreach(Arc, PosArCs),
      fromto(States1, [S1,S2|States1c], States1c, [])
    do ( Arc = arc(S1, _, S2) -> true
         ; Arc = arc(S1, _, S2, _)
         )
    ),
    sort(Alphabet1, Alphabet2),
    sort(Sources1, Sources2),
    sort(Sinks1, Sinks2),
    sort(States1, States2),
    ( foreach(P, Final),
      foreach(N, NegFinalT),
      foreach(N #= P, NeqsT)
    do true
    ),
    ( Counters==[] ->
      NegCounters = [],
      NegInitial = [],
      NegFinal = [],
      Aux = true,
      negaut_simple(PosArCs, NegSourcesSinks, NegArCs,
                    Sources2, Sinks2, States2, Alphabet2)
    ; NegCounters = [ | Counters],
      NegInitial = [0 | Initial],
      NegFinal = [FlagT | NegFinalT],
      Neqs = [FlagT #= 1 | NeqsT],
      orify(Neqs, Aux),
      negaut_counters(PosArCs, NegSourcesSinks, NegArCs,
                      Sources2, Sinks2, States2, Alphabet2, Counters)
    ).
negaut_simple(PosArcs, NegSourcesSinks, NegArcs, Sources1, Sinks1, States1, Alphabet) :-
  ord_subtract(States1, Sinks1, Sinks2),
  ( foreach(S1,Sources1),
    foreach(source(S1),Sources2) do true ),
  ( foreach(S2,Sinks2),
    foreach(sink(S2),Sinks3) do true ),
  ( foreach(arc(S3,K,_),PosArcs),
    foreach(S3-K,KL1) do true ),
  keysort(KL1, KL2),
  keyclumped(KL2, KL3),
  ( foreach(S4-Set1,KL3),
    fromto(NegArcs4,NegArcs5,NegArcs7,NegArcs8),
    param(Alphabet)
    do ord_subtract(Alphabet, Set1, CSet1),
      ( foreach(C,CSet1),
        fromto(NegArcs5,[arc(S4,C,fail)|NegArcs6],NegArcs6,NegArcs7),
        param(S4)
        do true
      )
    ),
    ( NegArcs4 == NegArcs8 ->
      NegArcs8 = [],
      append(Sources2, Sinks3, NegSourcesSinks)
    ; ( foreach(A,Alphabet),
      foreach(arc(fail,A,fail),NegArcs8)
      do true
    ),
    append(Sources2, [sink(fail)|Sinks3], NegSourcesSinks)
  ),
  append(NegArcs4, PosArcs, NegArcs).

negaut_counters(PosArcs1, NegSourcesSinks, NegArcs, Sources1, Sinks1, States1, Alphabet, Counters) :-
  ( foreach(S1,Sources1),
    foreach(source(S1),Sources2) do true
  ),
  ( foreach(S2,States1),
    foreach(state(S2),States2) do true
  ),
  append(States2, States1, States3),
  append(Sources2, Sources1, Sources3),
  ord_subtract(Sources3, Sinks3, Sinks4),
  ( foreach(S5,Sources3),
    foreach(source(S5),Sources4) do true
  ),
  append(Sources4, Sources3, Sources5),
  append(Sources3, Sinks4, Sources6),
  foreach(arc(S7,A,fail),NegArcs7)
  do true,
  append(Sources6, [sink(fail)|Sinks4], NegSourcesSinks),
  append(NegArcs7, PosArcs, NegArcs).

negaut_sieve(Sources1, Sinks1, States1, Alphabet) :-
  ord_subtract(States1, Sinks1, Sinks2),
  ( foreach(S1,Sources1),
    foreach(source(S1),Sources2) do true
  ),
  ( foreach(S2,Sinks2),
    foreach(sink(S2),Sinks3) do true
  ),
  ( foreach(arc(S3,K,_),PosArcs),
    foreach(S3-K,KL1) do true
  ),
  keysort(KL1, KL2),
  keyclumped(KL2, KL3),
  ( foreach(S4-Set1,KL3),
    fromto(NegArcs4,NegArcs5,NegArcs7,NegArcs8),
    param(Alphabet)
    do ord_subtract(Alphabet, Set1, CSet1),
      ( foreach(C,CSet1),
        fromto(NegArcs5,[arc(S4,C,fail)|NegArcs6],NegArcs6,NegArcs7),
        param(S4)
        do true
      )
    ),
    ( NegArcs4 == NegArcs8 ->
      NegArcs8 = [],
      append(Sources2, Sinks3, NegSourcesSinks)
    ; ( foreach(A,Alphabet),
      foreach(arc(fail,A,fail),NegArcs8)
      do true
    ),
    append(Sources2, [sink(fail)|Sinks3], NegSourcesSinks)
  ),
  append(NegArcs4, PosArcs, NegArcs).

negaut_graph(Sources1, Sinks1, States1, Alphabet) :-
  ord_subtract(States1, Sinks1, Sinks2),
  ( foreach(S1,Sources1),
    foreach(source(S1),Sources2) do true
  ),
  ( foreach(S2,Sinks2),
    foreach(sink(S2),Sinks3) do true
  ),
  ( foreach(arc(S3,K,_),PosArcs),
    foreach(S3-K,KL1) do true
  ),
  keysort(KL1, KL2),
  keyclumped(KL2, KL3),
  ( foreach(S4-Set1,KL3),
    fromto(NegArcs4,NegArcs5,NegArcs7,NegArcs8),
    param(Alphabet)
    do ord_subtract(Alphabet, Set1, CSet1),
      ( foreach(C,CSet1),
        fromto(NegArcs5,[arc(S4,C,fail)|NegArcs6],NegArcs6,NegArcs7),
        param(S4)
        do true
      )
    ),
    ( NegArcs4 == NegArcs8 ->
      NegArcs8 = [],
      append(Sources2, Sinks3, NegSourcesSinks)
    ; ( foreach(A,Alphabet),
      foreach(arc(fail,A,fail),NegArcs8)
      do true
    ),
    append(Sources2, [sink(fail)|Sinks3], NegSourcesSinks)
  ),
  append(NegArcs4, PosArcs, NegArcs).

negaut_gen(Sources1, Sinks1, States1, Alphabet) :-
  ord_subtract(States1, Sinks1, Sinks2),
  ( foreach(S1,Sources1),
    foreach(source(S1),Sources2) do true
  ),
  ( foreach(S2,Sinks2),
    foreach(sink(S2),Sinks3) do true
  ),
  ( foreach(arc(S3,K,_),PosArcs),
    foreach(S3-K,KL1) do true
  ),
  keysort(KL1, KL2),
  keyclumped(KL2, KL3),
  ( foreach(S4-Set1,KL3),
    fromto(NegArcs4,NegArcs5,NegArcs7,NegArcs8),
    param(Alphabet)
    do ord_subtract(Alphabet, Set1, CSet1),
      ( foreach(C,CSet1),
        fromto(NegArcs5,[arc(S4,C,fail)|NegArcs6],NegArcs6,NegArcs7),
        param(S4)
        do true
      )
    ),
    ( NegArcs4 == NegArcs8 ->
      NegArcs8 = [],
      append(Sources2, Sinks3, NegSourcesSinks)
    ; ( foreach(A,Alphabet),
      foreach(arc(fail,A,fail),NegArcs8)
      do true
    ),
    append(Sources2, [sink(fail)|Sinks3], NegSourcesSinks)
  ),
  append(NegArcs4, PosArcs, NegArcs).

negaut_minimal(Sources1, Sinks1, States1, Alphabet) :-
  ord_subtract(States1, Sinks1, Sinks2),
  ( foreach(S1,Sources1),
    foreach(source(S1),Sources2) do true
  ),
  ( foreach(S2,Sinks2),
    foreach(sink(S2),Sinks3) do true
  ),
  ( foreach(arc(S3,K,_),PosArcs),
    foreach(S3-K,KL1) do true
  ),
  keysort(KL1, KL2),
  keyclumped(KL2, KL3),
  ( foreach(S4-Set1,KL3),
    fromto(NegArcs4,NegArcs5,NegArcs7,NegArcs8),
    param(Alphabet)
    do ord_subtract(Alphabet, Set1, CSet1),
      ( foreach(C,CSet1),
        fromto(NegArcs5,[arc(S4,C,fail)|NegArcs6],NegArcs6,NegArcs7),
        param(S4)
        do true
      )
    ),
    ( NegArcs4 == NegArcs8 ->
      NegArcs8 = [],
      append(Sources2, Sinks3, NegSourcesSinks)
    ; ( foreach(A,Alphabet),
      foreach(arc(fail,A,fail),NegArcs8)
      do true
    ),
    append(Sources2, [sink(fail)|Sinks3], NegSourcesSinks)
  ),
  append(NegArcs4, PosArcs, NegArcs).

negaut_finite(Sources1, Sinks1, States1, Alphabet) :-
  ord_subtract(States1, Sinks1, Sinks2),
  ( foreach(S1,Sources1),
    foreach(source(S1),Sources2) do true
  ),
  ( foreach(S2,Sinks2),
    foreach(sink(S2),Sinks3) do true
  ),
  ( foreach(arc(S3,K,_),PosArcs),
    foreach(S3-K,KL1) do true
  ),
  keysort(KL1, KL2),
  keyclumped(KL2, KL3),
  ( foreach(S4-Set1,KL3),
    fromto(NegArcs4,NegArcs5,NegArcs7,NegArcs8),
    param(Alphabet)
    do ord_subtract(Alphabet, Set1, CSet1),
      ( foreach(C,CSet1),
        fromto(NegArcs5,[arc(S4,C,fail)|NegArcs6],NegArcs6,NegArcs7),
        param(S4)
        do true
      )
    ),
    ( NegArcs4 == NegArcs8 ->
      NegArcs8 = [],
      append(Sources2, Sinks3, NegSourcesSinks)
    ; ( foreach(A,Alphabet),
      foreach(arc(fail,A,fail),NegArcs8)
      do true
    ),
    append(Sources2, [sink(fail)|Sinks3], NegSourcesSinks)
  ),
  append(NegArcs4, PosArcs, NegArcs).
foreach(sink(S2), Sinks3) do true,
    (foreach(Arc1, PosArcs1),
     foreach(Arc2, PosArcs2),
     fromto(NegArcs1, NegArcs2, NegArcs3, NegArcs4),
     foreach(S3-K, KL1),
     param(Sinks1, Counters)
     do Arc1 =.. [arc,S3,K|_],
        (ord_member(S3, Sinks1) -> F=0 ; F=1),
        augment_arc(Arc1, F, Counters, Arc2, NegArcs2, NegArcs3)
    ),
    keysort(KL1, KL2),
    keyclumped(KL2, KL3),
    (foreach(S4-Set1, KL3),
     fromto(NegArcs4, NegArcs5, NegArcs7, NegArcs8),
     param(Alphabet, Counters)
     do ord_subtract(Alphabet, Set1, CSet1),
        (foreach(C, CSet1),
         fromto(NegArcs5, [arc(S4,C,fail,[1|Counters])|NegArcs6], NegArcs6, NegArcs7),
         param(S4, Counters)
         do true
        ),
     append(Sources2, [sink(fail)|Sinks3], NegSourcesSinks)
    ),
    append(NegArcs4, PosArcs2, NegArcs).

augment_arc(arc(S1,K,S2), F, Ctrs, arc(S1,K,S2,[F|Ctrs])) --> [].
augment_arc(arc(S1,K,S2,(P1->Q1 ; P2->Q2)), F, _, arc(S1,K,S2,(P1->[F|Q1] ; P2->[F|Q2]))) --> [].
augment_arc(arc(S1,K,S2,(P1->Q1)), F, Ctrs, arc(S1,K,S2,(P1->[F|Q1]))) --> !,
    {neg_arith(P1, P2)},
    [arc(S1,K,fail,(P2->[1|Ctrs])]).
augment_arc(arc(S1,K,S2,Ctrs), F, _, arc(S1,K,S2,[F|Ctrs])) --> [].

neg_arith(X #= Y, X #\= Y).
neg_arith(X #\= Y, X #= Y).
neg_arith(X #< Y, X #> Y).
neg_arith(X #=< Y, X #> Y).
neg_arith(X #> Y, X #=< Y).
neg_arith(X #>= Y, X #< Y).

orify([], true).
orify([X|L], Disj) :- orify(L, X, Disj).
orify([], X, X).
orify([Y|L], X, (X #\/ Disj)) :- orify(L, Y, Disj).

%----------------------------------------------------------------------------------------------------- ---------------------
union_dom_list_int(Dvars, Union) :-
    ( foreach(V,Dvars),
      foreach(S,Sets)
    do  fd_set(V, S)
    ),
    fdset_union(Sets, U),
    fdset_to_list(U, Union).

union_dom_set([], []).
union_dom_set([V|R], S) :-
    fd_set(V, SetValuesOfV),
    union_dom_set(R, Set),
    fdset_union(SetValuesOfV, Set, S).

same_size([]).
same_size([[-L]|R]) :-
    length(L, N),
same_size(R, N).

same_size([], _).
same_size([[-L]|R], N) :-
    length(L, N),
same_size(R, N).

sort_collection(COL, ATTR, SORTED_COL) :-
    build_key_collection(COL, ATTR, KEY_COL),
    keysort(KEY_COL, SORTED_KEY_COL),
    remove_key_from_collection(SORTED_KEY_COL, SORTED_COL).

build_key_collection([], _, []).
build_key_collection([ITEM|RCOL], ATTR, [KEY-ITEM|R]) :-
    extract_attr_value(ITEM, ATTR, KEY),
    build_key_collection(RCOL, ATTR, R).

extract_attr_value([ATTR-VALUE|_], ATTR, VALUE) :-
extract_attr_value([|ITEM|], ATTR, VALUE) :-
extract_attr_value(ITEM, ATTR, VALUE).

remove_key_from_collection([], []). remove_key_from_collection([-ITEM|R], [ITEM|S]) :-
remove_key_from_collection(R, S).

list_dvar_range([], 0) :- !.
list_dvar_range([X|Y], R) :-
  get_minimum([X|Y], Minimum),
  get_maximum([X|Y], Maximum),
  Min in Minimum..Maximum,
  Max in Minimum..Maximum,
  minimum(Min, [X|Y]),
  maximum(Max, [X|Y]),
  R #= Max-Min+1.

collection_distinct([], _).
collection_distinct([ITEM|R], ATTR) :-
  nth1(ATTR, ITEM, _-A),
  get_attr1(A, L),
  all_different(L),
  collection_distinct(R, ATTR).

collection_increasing_seq(COL, ATTRS) :-
  collection_increasing_seq1(COL, ATTRS, A),
  lex_chain(A, [op(#<)]).

collection_increasing_seq1([], _, []).

collection_increasing_seq1([ITEM|R], ATTRS, [ITEM|S]) :-
  collection_increasing_seq2(ATTRS, ITEM, ITEM_ATTRS),
  collection_increasing_seq1(R, ATTRS, S).

collection_increasing_seq2([], _, []).
collection_increasing_seq2([ATTR|R], ITEM, [A|S]) :-
  nth1(ATTR, ITEM, _-A),
  collection_increasing_seq2(R, ITEM, S).

collection([], _) :- !.
collection([Item|R], Types) :-
  check_item(Types, Item),
  collection(R, Types).

collection([|ITEM|], _, []).
collection([|ITEM|], [ATTR-V]|S),IsValid :-
  get_attr1(ATTR-V, V),
  check_attr(IsValid, V),
  collection([|ITEM|], _, _),
  collection([|ITEM|], ATTR-V, [ATTR-V]|S).
create_collection(R, ATTR, S).
create_collection([], [], _, _, []). create_collection([V1|R1], [V2|R2], ATTR1, ATTR2, [[ATTR1-V1,ATTR2-V2]|S]) :- create_collection(R1, R2, ATTR1, ATTR2, S).
create_collection([], _, _, []). create_collection([[_-L]|R], ATTR1, ATTR2, [[ATTR1-C]|S]) :-
        get_attr1(L, A), create_collection(A, ATTR2, C), create_collection(R, ATTR1, ATTR2, S).

check_item([], []) :- !.
check_item([T|S], [_-V|R]) :-
        check_type(T, V), check_item(S, R).

check_type(atom, V) :- atomic(V), !.
check_type(atom(L), V) :- atomic(V), member(V, L), !.
check_type(int, V) :- integer(V), !.
check_type(int_gteq(VAL), V) :- integer(V), V >= VAL, !.
check_type(int_diff(VAL), V) :- integer(V), V =\= VAL, !.
check_type(dvar, V) :- integer(V), !.
check_type(fdvar, V) :- var(V), !.
check_type(fdvar, V) :- integer(V),!
check_type(fdvar, V) :-
  fd_var(V),
  !.
check_type(dvar_gteq(VAL), V) :-
  integer(V),
  V >= VAL,
  !.
check_type(dvar_gteq(VAL), V) :-
  fd_var(V),
  V #>= VAL,
  !.
check_type(int(Low,Up), V) :-
  integer(V),
  V >= Low,
  V =< Up,
  !.
check_type(dvar(Low,Up), V) :-
  integer(V),
  V >= Low,
  V =< Up,
  !.
check_type(dvar(Low,Up), V) :-
  fd_var(V),
  V #>= Low,
  V #=< Up,
  !.
check_type(fdvar(Low,Up), V) :-
  integer(V),
  V >= Low,
  V =< Up,
  !.
check_type(fdvar(Low,Up), V) :-
  fd_var(V),
  V #>= Low,
  V #=< Up,
  !.
check_type(fdvar(Low,Up), V) :-
  var(V),
  V >= Low,
  V =< Up,
  !.
check_type(col(Types), C) :-
  collection(C, Types),
  !.
check_type(col(Len,Types), C) :-
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

```prolog
length(C, Len),
collection(C, Types), !.
check_type(col_len_gteq(Len,Types), C) :-
  length(C, L),
  L >= Len,
collection(C, Types), !.
check_type(non_empty_col(Types), C) :-
  length(C, L),
  L > 0,
collection(C, Types), !.

check_type(sint, _V) :- % TODO !.
check_type(svar, _V) :- % TODO !.

get_col_attr1([], _, [], []). 
get_col_attr1([([-C]|_]|R], 1, [D|S]) :- !,
  get_attr1(C, D),
get_col_attr1(R, 1, S).
get_col_attr1([([-C]|_]|R], 2, [D|S]) :- !,
  get_attr2(C, D),
get_col_attr1(R, 2, S).
get_col_attr1([([-C]|_]|R], 3, [D|S]) :-
  get_attr3(C, D),
get_col_attr1(R, 3, S).

get_col_attr2([], _, [], []). 
get_col_attr2([[_r,-C|_]|R], 1, [D|S]) :- !,
  get_attr1(C, D),
get_col_attr2(R, 1, S).
get_col_attr2([[_r,-C|_]|R], 2, [D|S]) :- !,
  get_attr2(C, D),
get_col_attr2(R, 2, S).
get_col_attr2([[_r,-C|_]|R], 3, [D|S]) :-
  get_attr3(C, D),
get_col_attr2(R, 3, S).

get_col_attr3([], _, [], []). 
get_col_attr3([[_r,-C|_]|R], 1, [D|S]) :- !,
  get_attr1(C, D),
get_col_attr3(R, 1, S).
get_col_attr3([[_r,-C|_]|R], 2, [D|S]) :- !,
```

get_attr2(C, D),
get_col_attr3(R, 2, S).
get_col_attr3([[_,_,-_C|_]|R], 3, [D|S]) :-
    get_attr3(C, D),
get_col_attr3(R, 3, S).

get_attr12([], []).
get_attr12([[-V1,_-_V2|_]|R], [V1-V2|S]) :-
get_attr12(R, S).

get_attr21([], []).
get_attr21([[-V1,_-_V2|_]|R], [V2-V1|S]) :-
get_attr21(R, S).

get_kattr1([], _, []).
get_kattr1([[-V|_]|R], K, [V-K|S]) :-
    K1 is K+1,
    get_kattr1(R, K1, S).

get_attr1([], []).
get_attr1([[-V|_]|R], [V|S]) :-
get_attr1(R, S).

get_attr2([], []).
get_attr2([[-_-_V|_]|R], [V|S]) :-
get_attr2(R, S).

get_attr3([], []).
get_attr3([[-_-_V|_]|R], [V|S]) :-
get_attr3(R, S).

get_attr4([], []).
get_attr4([[-_-__-_V|_]|R], [V|S]) :-
get_attr4(R, S).

get_attr5([], []).
get_attr5([[-_-__-__-_V|_]|R], [V|S]) :-
get_attr5(R, S).

get_attr6([], []).
get_attr6([[-_-__-__-__-_V|_]|R], [V|S]) :-
get_attr6(R, S).

get_attr7([], []).
get_attr7([[-_-__-__-__-__-_V|_]|R], [V|S]) :-
get_attr7(R, S).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

get_attr8([], []).
get_attr8([[-V] | R], [V | S]) :-
    get_attr8(R, S).

get_attr11([], []). 
get_attr11([[-V] | R], [U | S]) :-
    get_attr1(V, U),
    get_attr11(R, S).

get_proj1([], []).
get_proj1([[A-V] | R], [[A-V] | S]) :-
    get_proj1(R, S).

get_minimum([], 0).
get_minimum([V | R], M) :-
    fd_min(V, Min),
    get_minimum1(R, Min, M).

get_minimum1([], Min, Min).
get_minimum1([V | R], Min, M) :-
    fd_min(V, MinV),
    MinV < Min,
    !,
    get_minimum1(R, MinV, M).

get_maximum([], 0).
get_maximum([V | R], M) :-
    fd_max(V, Max),
    get_maximum1(R, Max, M).

get_maximum1([], Max, Max).
get_maximum1([V | R], Max, M) :-
    fd_max(V, MaxV),
    MaxV > Max,
    !,
    get_maximum1(R, MaxV, M).

get_maximum1([_] | R], Max, M) :-
    get_maximum1(R, Max, M).

get_collection([], _, []).
get_collection([V | R], ATTR, [[ATTR-V] | S]) :-
    get_collection(R, ATTR, S).
gen_varcst([], [], []).  

gen_varcst([V|R], [C|S], [VC|T]) :-  
    VC #= V+C,  
    gen_varcst(R, S, T).

gen_quotient([], _, []).  

gen_quotient([V|R], Size, [Q|T]) :-  
    Sizel is Size-1,  
    Remainder in 0.. Sizel,  
    V #= Size*Q+ Remainder,  
    gen_quotient(R, Size, T).

gen_remainder([], _, []).  

gen_remainder([V|R], M, [Remainder |T]) :-  
    M1 is M-1,  
    Remainder in 0.. M1,  
    V #= M*+ Remainder,  
    gen_remainder(R, M, T).

flattern([], []).  

flattern([L|R], S) :-  
    flattern(R, T),  
    append(L, T, S).

generate_partition_var([], _, [], _, _).  

generate_partition_var([V|R], PVALS, [P|S], MAX) :-  
    P in 0..MAX,  
    gen_part_var(PVALS, 1, V, P),  
    generate_partition_var(R, PVALS, S, MAX).

generate_partition_var([V|R], PVALS, [P|S], MAX, DIFF) :-  
    P in 1..MAX,  
    P \= DIFF,  
    gen_part_var(PVALS, 1, V, P),  
    generate_partition_var(R, PVALS, S, MAX, DIFF).

gen_part_var([], _, _, _).  

gen_part_var([L|R], N, V, P) :-  
    gen_part_var1(L, N, V, P, Vdiff),  
    call(Vdiff #=> P \= N),  
    N1 is N+1,  
    gen_part_var(R, N1, V, P).

gen_part_var1([], _, _, _, 1).  

gen_part_var1([U|R], N, V, P, V \= U \= S) :-
2944  APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

V ≠ U => P ≠ N,
\text{gen}_\text{part}_\text{var1}(R, N, V, P, S).

\text{set}_\text{to}_\text{list}([], []). \\
\text{set}_\text{to}_\text{list}([S], L) :- \\
\text{set}_\text{to}_\text{list}(S, L, []). \\
\text{set}_\text{to}_\text{list}((X,Y)) --> !, \\
\text{set}_\text{to}_\text{list}(X), \\
\text{set}_\text{to}_\text{list}(Y). \\
\text{set}_\text{to}_\text{list}(X) --> [X].

\text{list}_\text{to}_\text{set}([], []). \\
\text{list}_\text{to}_\text{set}([H|T], S) :- \\
\text{list}_\text{to}_\text{set}(T, H, S).

\text{list}_\text{to}_\text{set}([], X, X). \\
\text{list}_\text{to}_\text{set}([H|T], X, (X,S)) :- \\
\text{list}_\text{to}_\text{set}(T, H, S).

\text{count}_\text{var}_\text{notin}_\text{values}([], _, 0) :- \\
\text{count}_\text{var}_\text{notin}_\text{values}([VAR|RVAR], SORTED_\text{VALS}, NOUT) :- \\
\text{fd}_\text{set}(\text{VAR}, S), \\
\text{fdset}_\text{to}_\text{list}(S, L), \\
(\text{ord}_\text{intersect}(L, \text{SORTED}_\text{VALS}) => I=0 ; I=1), \\
\text{count}_\text{var}_\text{notin}_\text{values}(\text{RVAR}, \text{SORTED}_\text{VALS}, N), \\
\text{NOUT is} N+I.

\text{complete}_\text{card}(\text{MIN}, \text{MAX}, _, [], []) :- \\
\text{MIN} > \text{MAX}, \\
!.

\text{complete}_\text{card}(\text{MIN}, \text{MAX}, L, [[\text{var}_\text{OCC}|R], [\text{MIN}_\text{OCC}|S]]) :- \\
\text{MIN} =< \text{MAX}, \\
\text{OCC} \text{ in} 0..L, \\
\text{MIN1 is} \text{MIN}+1, \\
\text{complete}_\text{card}(\text{MIN1}, \text{MAX}, L, R, S).

\text{complete}_\text{card}(\text{MIN}, \text{MIN}, \text{NVARS}, \text{VALS}, \text{NOCCS}, [\text{V-N}]) :- !, \\
\text{complete}_\text{card1}(\text{MIN}, \text{VALS}, \text{NOCCS}, \text{V}_N), \\
(\text{V}_N=[] => \text{V} = \text{MIN}, \text{N} \text{ in} 0..\text{NVARS} ; \text{V}_N=\text{V}-\text{N}).

\text{complete}_\text{card}(\text{MIN}, \text{MAX}, \text{NVARS}, \text{VALS}, \text{NOCCS}, [\text{V-N}|R]) :- \\
\text{MIN} < \text{MAX}, \\
\text{complete}_\text{card1}(\text{MIN}, \text{VALS}, \text{NOCCS}, \text{V}_N), \\
(\text{V}_N=[] => \text{V} = \text{MIN}, \text{N} \text{ in} 0..\text{NVARS} ; \text{V}_N=\text{V}-\text{N}), \\
\text{MIN1 is} \text{MIN} + 1,
complete_card(MIN1, MAX, NVARS, VALS, NOCCS, R).

complete_card1(_, [], [], []) :- !.
complete_card1(MIN, [MIN|_], [NOCC|_], MIN-NOCC) :- !.
complete_card1(MIN, [VAL|R], [_NOCC|S], MN) :-
  MIN =\= VAL,
  complete_card1(MIN, R, S, MN).

complete_card_low_up(MIN, MIN, NVARS, VALS, OMINS, OMAXS, [V-N]) :- !,
  complete_card_low_up1(MIN, VALS, OMINS, OMAXS, V_N),
  (V_N=[] -> V=MIN, N in 0..NVARS ; V_N=V-N).
complete_card_low_up(MIN, MAX, NVARS, VALS, OMINS, OMAXS, [V-N|R]) :-
  MIN < MAX,
  complete_card_low_up1(MIN, VALS, OMINS, OMAXS, V_N),
  (V_N=[] -> V=MIN, N in 0..NVARS ; V_N=V-N),
  MIN1 is MIN + 1,
  complete_card_low_up(MIN1, MAX, NVARS, VALS, OMINS, OMAXS, R).

complete_card_low_up1(_, [], [], [], []) :- !.
complete_card_low_up1(MIN, [MIN|_], [OMIN|_], [OMAX|_], MIN-NOCC) :- !,
  NOCC in OMIN..OMAX.
complete_card_low_up1(MIN, [VAL|R], [_|S], [_|T], MN) :-
  MIN =\= VAL,
  complete_card_low_up1(MIN, R, S, T, MN).

complete_card_consec(LOW, UP, ATMOST, NVAR, [LOW-N|R]) :-
  LOW < UP, !,
  N in 0..ATMOST,
  LOW1 is LOW+1,
  complete_card_consec(LOW1, UP, ATMOST, NVAR, R).
complete_card_consec(LOW, LOW, _, NVAR, [LOW-N]) :-
  N in 0..NVAR.

build_or_var_in_values([], _, true).
build_or_var_in_values([U], V, (V#=U)) :- !.
build_or_var_in_values([U1,U2|R], V, (V#=U1) #\ S) :-
  build_or_var_in_values([U2|R], V, S).

call_term_relop_value(TERM, =, VALUE) :- !,
  call(TERM #= VALUE).
call_term_relop_value(TERM, =\=, VALUE) :- !,
  call(TERM #\= VALUE).
call_term_relop_value(TERM, <-, VALUE) :- !,
  call(TERM #< VALUE).
call_term_relop_value(TERM, >=, VALUE) :- !,
  call(TERM #>= VALUE).
call_term_relop_value(TERM, >, VALUE) :- !,
call(TERM #> VALUE).
call_term_relop_value(TERM, =<, VALUE) :-
call(TERM #=< VALUE).

gen_matrix_bool(MINBINS, MAXBINS, _, []) :-
MINBINS > MAXBINS, !.
gen_matrix_bool(MINBINS, MAXBINS, BINS, [LINE|RLINES]) :-
MINBINS =< MAXBINS,
gen_matrix_bool1(BINS, MINBINS, LINE),
MINBINS1 is MINBINS+1,
gen_matrix_bool(MINBINS1, MAXBINS, BINS, RLINES).

/gen_matrix_bool1([], _, []).
gen_matrix_bool1([BIN|RBINS], IDBIN, [B|R]) :-
BIN #= IDBIN #<=> B,
gen_matrix_bool1(RBINS, IDBIN, R).

common1([], _, [], 0).
common1([V|R], VARS2, [LINE|S], SB+T) :-
common2(VARS2, V, LINE, SUM),
call(SUM #> 0 #<=> SB),
common1(R, VARS2, S, T).

common2([], _, [], 0).
common2([U|R], V, [B|S], B+T) :-
U #= V #<=> B,
common2(R, V, S, T).

gen_cum_tasks([], [], [], [], [task(O,D,E,H,T)|R]) :-
T1 is T+1,
gen_cum_tasks(RO, RD, RE, RH, T1, R).
k_ary_tree([], _, _, _).
k_ary_tree([J|R], INDEXES, SUCCS, K) :-
k_ary_treel(INDEXES, SUCCS, J, Term),
call(Term #=> K),
k_ary_tree(R, INDEXES, SUCCS, K).

k_ary_treel([], [], 0).
k_ary_treel([I|S], [S_I|R], J, B_IJ+T) :-
S_I #= J #\ I #= J #=> B_IJ,
k_ary_treel(S, R, J, T).
ori_dur_end([], [], []).  
ori_dur_end([O|RO], [D|RD], [E|RE]) :-  
    O + D #= E,  
    ori_dur_end(RO, RD, RE).

ori_end([], []).  
ori_end([O|RO], [E|RE]) :-  
    O #=< E,  
    ori_end(RO, RE).

link_index_to_attribute([], [], _, _).  
link_index_to_attribute([ID|RID], [ATT|RATT], Vi, Ai) :-  
    Vi #= ID #<=> Ai #= ATT,  
    link_index_to_attribute(RID, RATT, Vi, Ai).

get_sliding_prod([], _, []).  
get_sliding_prod([V|R], P, [P|S]) :-  
    Q is V*P,  
    get_sliding_prod(R, Q, S).

get_min_list_list_dvar([], []).  
get_min_list_list_dvar([L|R], [Min|S]) :-  
    get_min_list_dvar(L, _, Min),  
    get_min_list_list_dvar(R, S).

get_min_list_dvar([], Min, Min).  
get_min_list_dvar([V|R], Cur, Min) :-  
    fd_min(V, Vmin),  
    (integer(Cur) -> Next is min(Cur,Vmin) ; Next = Vmin),  
    get_min_list_dvar(R, Next, Min).

get_max_list_list_dvar([], []).  
get_max_list_list_dvar([L|R], [Min|S]) :-  
    get_max_list_dvar(L, _, Min),  
    get_max_list_list_dvar(R, S).

get_max_list_dvar([], Max, Max).  
get_max_list_dvar([V|R], Cur, Max) :-  
    fd_max(V, Vmax),  
    (integer(Cur) -> Next is max(Cur,Vmax) ; Next = Vmax),  
    get_max_list_dvar(R, Next, Max).

get_ranges([], [], []).  
get_ranges([A|R], [B|S], [C|T]) :-  
    C is B-A+1,  
    get_ranges(R, S, T).
create_matrix(N, Inf, Sup, MB) :-
    length(MB, N),
    create_matrix1(MB, N, Inf, Sup).

create_matrix1([], _, _, _).
create_matrix1([L|R], N, Inf, Sup) :-
    length(L, N),
    domain(L, Inf, Sup),
    create_matrix1(R, N, Inf, Sup).

count_relop(= , NIN, LIMIT, FLAG) :- NIN #= LIMIT #<=> FLAG.
count_relop(\=, NIN, LIMIT, FLAG) :- NIN \= LIMIT \<=> FLAG.
count_relop(< , NIN, LIMIT, FLAG) :- NIN #< LIMIT #<=> FLAG.
count_relop(\>= , NIN, LIMIT, FLAG) :- NIN #\= LIMIT #\<=> FLAG.
count_relop(> , NIN, LIMIT, FLAG) :- NIN #> LIMIT #<=> FLAG.
count_relop(\=< , NIN, LIMIT, FLAG) :- NIN #\= LIMIT #\<=> FLAG.

used_by_reified([], _, _).
used_by_reified([V|R], VARS1, VARS2) :-
    used_by_reified1(VARS1, V, Term1),
    used_by_reified1(VARS2, V, Term2),
    call(Term1 \=\= Term2),
    used_by_reified1(R, VARS1, VARS2).

used_by_reified1([], _, 0).
used_by_reified1([U|R], V, B+T) :-
    U \= V \<\= B,
    used_by_reified1(R, V, T).

remove_duplicates([], []).
remove_duplicates([X|R], S) :-
    member(X, R),
    !,
    remove_duplicates(R, S).
remove_duplicates([X|R], [X|S]) :-
    remove_duplicates(R, S).

gcc_no_loop1([], _, 0).
gcc_no_loop1([VAR|RVAR], J, BJ+S) :-
    BJ \<\= VAR \= J,
    J1 is J+1,
    gcc_no_loop1(RVAR, J1, S).

gcc_no_loop2(J, N, _, [], _, 0) :-
    J > N, !.
gcc_no_loop2(J, N, I, [VAR|RVAR], VAL, BIJ+S) :-
    J =< N,
    J =\= I, !,
    BIJ #\=< VAR #\= VAL,
    J1 is J+1,
    gcc_no_loop2(J1, N, I, RVAR, VAL, S).

gcc_no_loop2(J, N, I, [\_|RVAR], VAL, 0+S) :-
    J =< N,
    J = I,
    J1 is J+1,
    gcc_no_loop2(J1, N, I, RVAR, VAL, S).

% cond_lex/5 is used in order to state automata associated to constraints
% cond_lex_greatereq, cond_lex_greater, cond_lex_lesseq and cond_lex_less.
% cond_lex/3 is used in order to state the automaton associated to
% constraint cond_lex_cost.
cond_lex(VECTOR1, VECTOR2, PREFERENCE_TABLE, O1, O2) :-
    cond_lex_signature(VECTOR1, VECT1),
    cond_lex_signature(VECTOR2, VECT2),
    % from each item extract a tuple of values and add key at the end
    gen_tuples(PREFERENCE_TABLE, 1, T1),
    % sort in lexicographic order
    sort(T1, T2),
    % to each tuple of value add state variables
    gen_tuples_var(T2, T3),
    % get arity of the tuples
    T1 = [T\_], functor(T, _, N),
    retractall(num_state(_)),
    % initial state number minus 1
    assert(num_state(0)),
    % fix the states variables
    gen_state(1, N, T3),
    % get last state
    num_state(LastS),
    % generate the list of states of the automaton
    gen_states(0, LastS, States),
    % generate the list of transitions of the automaton
    gen_transitions(1, N, T3, Transitions),
    % get number of tuples of preference table
    length(PREFERENCE_TABLE, NbTuples),
    % O1 indicates position of tuple associated to VECTOR1
    % O2 indicates position of tuple associated to VECTOR2
    domain([O1,O2], 1, NbTuples),
    % build signature variables for the automaton computing O1
    append(VECT1,[O1],VECTOR_O1),
    % build signature variables for the automaton computing O2
    append(VECT2, [O2], VECTOR_O2).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

append(VECT2,[O2],VECTOR_O2),
% state automaton that computes O1
automaton(VECTOR_O1, _,
VECTOR_O1,
States,
Transitions,
[], [], []),
% state automaton that computes O2
automaton(VECTOR_O2, _,
VECTOR_O2,
States,
Transitions,
[], [], []).

cond_lex(VECTOR, PREFERENCE_TABLE, 0) :-
  cond_lex_signature(VECTOR, VECT),
  % from each item extract a tuple of values and add key at the end
  gen_tuples(PREFERENCE_TABLE, 1, T1),
  % sort in lexicographic order
  sort(T1, T2),
  % to each tuple of value add state variables
  gen_tuples_var(T2, T3),
  % get arity of the tuples
  T1 = [T|_], functor(T, _, N),
  retractall(num_state(_)),
  % initial state number minus 1
  assert(num_state(0)),
  % fix the states variables
  gen_state(1, N, T3),
  % get last state
  num_state(LastS),
  % generate the list of states of the automaton
  gen_states(0, LastS, States),
  % generate the list of transitions of the automaton
  gen_transitions(1, N, T3, Transitions),
  % get number of tuples of preference table
  length(PREFERENCE_TABLE, NbTuples),
  % O indicates position of tuple associated to VECTOR
  domain([O], 1, NbTuples),
  % build signature variables for the automaton computing O
  append(VECT,[O],VECTOR_O),
  % state automaton that computes O
  automaton(VECTOR_O, _,
VECTOR_O,
States,
Transitions,
cond_lex_signature([], []).  
cond_lex_signature([[var-VAR]|R], [VAR|S]) :-  
cond_lex_signature(R, S).

gen_tuples([], _, []).  
gen_tuples([[-X]|Y], I, [U|V]) :-  
gen_tuple(X, I, U),  
J is I + 1,  
gen_tuples(Y, J, V).

gen_tuple(X, I, U) :-  
gen_tup(X, Y),  
append(Y, [I], Y1),  
append([t], Y1, Z),  
U =.. Z.

gen_tup([], []).  
gen_tup([[-I]|R], [I|S]) :-  
gen_tup(R, S).

gen_tuples_var([], []).  
gen_tuples_var([A|B], [C|D]) :-  
A =.. LA,  
LA = [TA|RA],  
add_var_to_list_elem(RA, RC),  
LC = [TA|RC],  
C =.. LC,  
gen_tuples_var(B, D).

add_var_to_list_elem([], []).  
add_var_to_list_elem([A|RA], [A-_|R]) :-  
add_var_to_list_elem(RA, R).

gen_state(I, N, L) :-  
I < N, !,  
gen_state1(L, [], I, 1, 1),  
J is I + 1,  
gen_state(J, N, L).  
gen_state(N, N, L) :-  
gen_state1(L, [], N, 1, 0).

gen_state1([], _, _, _, _) :- !.  
gen_state1([F|R], [], I, Inc, Inc1) :-  
!,
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

arg(I, F, FI),
FI = _-S1,
num_state(S),
S1 is S + Inc,
retract(num_state(S)),
assert(num_state(S1)),
gen_state1(R, F, I, Inc1, Inc1).
gen_state1([F|R], P, I, Inc, Inc1) :-
arg(I, F, FI),
arg(I, P, PI),
FI = VI-SI1,
PI = UI--_,
UI =\= VI,
\!,
num_state(SI),
SI1 is SI + Inc,
retract(num_state(SI)),
assert(num_state(SI1)),
gen_state1(R, F, I, Inc1, Inc1).
gen_state1([F|R], P, I, Inc, Inc1) :-
I > 1,
J is I - 1,
arg(J, F, FJ),
arg(J, P, PJ),
FJ = _-SJ,
PJ = _-RJ,
SJ =\= RJ,
\!,
arg(I, F, FI),
FI = _-SI1,
num_state(SI),
SI1 is SI + Inc,
retract(num_state(SI)),
assert(num_state(SI1)),
gen_state1(R, F, I, Inc1, Inc1).
gen_state1([F|R], _, I, _, Inc1) :-
arg(I, F, FI),
FI = _-SI,
num_state(SI),
gen_state1(R, F, I, Inc1, Inc1).

gen_states(0, J, [source(0)|R]) :- !,
gen_states(1, J, R).
gen_states(I, J, R /*[node(I)|R]*/) :-
I > 0,
I < J, !,
I1 is I + 1,
gen_states(I1, J, R).
gen_states(J, J, [sink(J)]) :-
J > 0.

gen_transitions(I, N, L, T) :-
I =< N, !,
gen_transitions1(L, [], I, T1),
J is I + 1,
gen_transitions(J, N, L, T2),
append(T1, T2, T).
gen_transitions(I, N, _, []) :-
I > N.

gen_transitions1([F|R], [], 1, [arc(0,V1,S1)|Rarc]) :-
!,
arg(1, F, F1),
F1 = V1-S1,
gen_transitions1(R, F, 1, Rarc).
gen_transitions1([F|R], [], I, [arc(SJ,VI,SI)|Rarc]) :-
I > 1,
!,
J is I - 1,
arg(J, F, FJ),
arg(I, F, FI),
FJ = _-SJ,
FI = VI-SI,
gen_transitions1(R, F, I, Rarc).
gen_transitions1([F|R], P, I, [arc(SJ,VI,SI)|Rarc]) :-
arg(I, F, FI),
arg(I, P, PI),
FI = VI-SI,
PI = UI-RI,
(SI =\= RI -> true ; VI=\=UI),
!,
(I=1 -> SJ=0 ; J is I-1, arg(J,F,FJ), FJ=_-SJ),
gen_transitions1(R, F, I, Rarc).
gen_transitions1([F|R], _, I, Rarc) :-
!,
gen_transitions1(R, F, I, Rarc).
gen_transitions1([], _, _, []).
call(SUM_INTER #=< LIMIT),
I1 is I+1,
sliding_time_window1(RO, RD, I1, ORIGINS, DURATIONS, WINDOW_SIZE, LIMIT).

sliding_time_window2([], [], _, _, _, _, _, 0) :- !.
sliding_time_window2([_|RO], [RD], J, Oi, Di, I, WINDOW_SIZE, LIMIT, min(Di,WINDOW_SIZE)+SUM) :-
    I = J,
    !,
    J1 is J+1,
    sliding_time_window2(RO, RD, J1, Oi, Di, I, WINDOW_SIZE, LIMIT, SUM).
sliding_time_window2([Oj|RO], [_Dj|RD], J, Oi, Di, I, WINDOW_SIZE, LIMIT, SUM) :-
    I =\= J,
    fd_max(Oj, MaxOj),
    fd_max(Dj, MaxDj),
    fd_min(Oi, MinOi),
    MaxEj is MaxOj+MaxDj,
    MaxEj < MinOi,
    !,
    J1 is J+1,
    sliding_time_window2(RO, RD, J1, Oi, Di, I, WINDOW_SIZE, LIMIT, SUM).
sliding_time_window2([Oj|RO], [Dj|RD], J, Oi, Di, I, WINDOW_SIZE, LIMIT, SUM) :-
    I =\= J,
    fd_min(Oj,MinOj),
    fd_max(Oi,MaxOi),
    E is MaxOi+WINDOW_SIZE-1,
    MinOj > E,
    !,
    J1 is J+1,
    sliding_time_window2(RO, RD, J1, Oi, Di, I, WINDOW_SIZE, LIMIT, SUM).

gen_automaton_state(ATOM, I, J, STATE) :-
    number_codes(I, ICODE), atom_codes(IATOM, ICODE),
    number_codes(J, JCODE), atom_codes(JATOM, JCODE),
atom_concat(ATOM, ‘_’, SH),
atom_concat(SH, IATOM, SI),
atom_concat(SI, ‘_’, SJ),
atom_concat(SJ, JATOM, STATE).

check_lesseq([], []).
check_lesseq([U|R], [V|S]) :-
   U =< V,
   check_lesseq(R, S).

get_sum([], 0).
get_sum([V|R], S) :-
   get_sum(R, T),
   S is V+T.

build_sum_var([], 0).
build_sum_var([V|R], S) :-
   build_sum_var(R, T),
   S is V+T.

build_sum_squares_var([], 0).
build_sum_squares_var([V|R], S) :-
   build_sum_squares_var(R, T),
   S is V*V+T.

build_sum_cubes_var([], 0).
build_sum_cubes_var([V|R], S) :-
   build_sum_cubes_var(R, T),
   S is V*V*V+T.

build_prod_var([], 1).
build_prod_var([V|R], S) :-
   build_prod_var(R, T),
   S is V*T.

build_sliding_sums([], _, [], []).
build_sliding_sums([V|R], P, [PV|S]) :-
   PV #= P+V,
   build_sliding_sums(R, PV, S).

period1(0, _, [], _) :- !.
period1(P, L, [R|S]) :-
   P > 0,
   period2(L, 0, P, R),
   P1 is P-1,
   period1(P1, L, S).

period2([], _, 0, []) :- !.
period2([I|P], I, P, [I|R]) :-
   P > 0,
P1 is P-1,
period2([], I, P1, R).
period2([X|Y], I, P, R) :-
  I1 is (I+1) mod P,
  period2(Y, I1, P, S),
  period3(X, I, S, R).

period3(X, 0, [U|V], [W|V]) :- !,
  append([X], U, W).
period3(X, I, [U|V], [U|W]) :-
  I > 0,
  I1 is I-1,
  period3(X, I1, V, W).

period4([], _, _, []).
period4([L|LL], Z, CTR, [B|S]) :-
  period5(L, Z, CTR, R),
  call(R #<=> B),
  period4(LL, Z, CTR, S).

period5([], _, _, 1).
period5([L|R], Z, CTR, T #\ S) :-
  period6(L, Z, CTR, T),
  period5(R, Z, CTR, S).

period6([], _, _, 1) :- !.
period6([_], _, _, 1) :- !.
period6([X,Y|R], 1, =, X#=Y #\ S) :- !,
  period6([Y|R], 1, =, S).
period6([X,Y|R], 1, =\=, X#=Y #\ S) :- !,
  period6([Y|R], 1, =\=, S).
period6([X,Y|R], 1, <, X#<Y #\ S) :- !,
  period6([Y|R], 1, <, S).
period6([X,Y|R], 1, >=, X#>=Y #\ S) :- !,
  period6([Y|R], 1, >=, S).
period6([X,Y|R], 1, >, X#>Y #\ S) :- !,
  period6([Y|R], 1, >, S).
period6([X,Y|R], 1, =<, X#=<Y #\ S) :- !,
  period6([Y|R], 1, =<, S).
period6([X,Y|R], 0, =, (X#=0 #\ X#0 #\ X#=Y) #\ S) :- !,
  period6([Y|R], 0, =, S).
period6([X,Y|R], 0, =\=, (X#=0 #\ Y#=0 #\ X#=Y) #\ S) :- !,
  period6([Y|R], 0, =\=, S).
period6([X,Y|R], 0, <, (X#0 #\ Y#0 #\ X#<Y) #\ S) :- !,
  period6([Y|R], 0, <, S).
period6([X,Y|R], 0, >, (X#0 #\ Y#0 #\ X#>Y) #\ S) :- !,
period6([Y|R], 0, >=, S).
period6([X,Y|R], 0, >, (X#=0 #\ Y#=0 #\ X#>Y) #\ S) :- !,
period6([Y|R], 0, >, S).
period6([X,Y|R], 0, =<, (X#=0 #\ Y#=0 #\ X#<Y) #\ S) :- !,
period6([Y|R], 0, =<, S).
period6([X,Y|R], 2, CTRS, Term #\ S) :- !,
     build_vectors_compare(X, Y, CTRS, Term),
period6([Y|R], 2, CTRS, S).

period7([], _, _, _, 0).
period7([B|R], I, P, N, (N #\ B #\ P#=I) #\ S) :- !,
     I1 is I+1,
     period7(R, I1, P, N #\ #\ B, S).

build_vectors_compare([], [], [], 1) :- !.
build_vectors_compare([X|RX], [Y|RY], [=|RCTR], X#=Y #\ R) :-
build_vectors_compare(RX, RY, RCTR, R).
build_vectors_compare([X|RX], [Y|RY], [\=|RCTR], X#=Y #\ R) :-
build_vectors_compare(RX, RY, RCTR, R).
build_vectors_compare([X|RX], [Y|RY], [<|RCTR], X#<Y #\ R) :-
build_vectors_compare(RX, RY, RCTR, R).
build_vectors_compare([X|RX], [Y|RY], [\>|RCTR], X#>Y #\ R) :-
build_vectors_compare(RX, RY, RCTR, R).
build_vectors_compare([X|RX], [Y|RY], [\=\=|RCTR], X#=\=Y #\ R) :-
build_vectors_compare(RX, RY, RCTR, R).
build_vectors_compare_change([], [], [], 0) :- !.
build_vectors_compare_change([X|RX], [Y|RY], [=|RCTR], X#=Y #\ R) :- !,
build_vectors_compare_change(RX, RY, RCTR, R).
build_vectors_compare_change([X|RX], [Y|RY], [\=\=|RCTR], X#=\=Y #\ R) :- !,
build_vectors_compare_change(RX, RY, RCTR, R).
build_vectors_compare_change([X|RX], [Y|RY], [<|RCTR], X#<Y #\ R) :- !,
build_vectors_compare_change(RX, RY, RCTR, R).
build_vectors_compare_change([X|RX], [Y|RY], [\>|RCTR], X#>Y #\ R) :- !,
build_vectors_compare_change(RX, RY, RCTR, R).
build_vectors_compare_change([X|RX], [Y|RY], [\=\=|RCTR], X#=\=Y #\ R) :- !,
build_vectors_compare_change(RX, RY, RCTR, R).

geost_dims(D, D, [D]) :- !.
geost_dims(D, K, [D|R]) :- D < K,
D1 is D+1,
geost_dims(D1, K, R).

geost1([], [], [], []).
geost1([OID|R], [SID|S], [X|T], [object(OID,SID,X)|U]) :-
geost1(R, S, T, U).

geost2([], [], [], []).
geost2([SID|R], [T|S], [L|U], [sbox(SID,T,L)|V]) :-
geost2(R, S, U, V).

bin_packing1([], _, []).
bin_packing1([[-B,-W]|R], I, [task(B,1,B1,W,I)|RT]) :-
I1 is I+1,
B1 #= B+1,
bin_packing1(R, I1, RT).

nvector_common(NVEC, VECTORS) :-
get_col_attr1(VECTORS, 1, VECTS),
transpose(VECTS, TVECTS),
get_min_list_list_dvar(TVECTS, MINS),
get_max_list_list_dvar(TVECTS, MAXS),
get_ranges(MINS, MAXS, RANGES),
reverse(RANGES, RRANGES),
get_sliding_prod(RRANGES, 1, PRODS),
reverse(MINS, RMINS),
nvector_common1(VECTS, RMINS, PRODS, VARS),
nvalue(NVEC, VARS).

nvector_common1([], _, _, []).
nvector_common1([VECT|R], RMINS, PRODS, [V|S]) :-
reverse(VECT, RVECT),
nvector_common2(RVECT, RMINS, PRODS, Term),
call(V #= Term),
nvector_common1(R, RMINS, PRODS, S).

nvector_common2([], _, _, 0).
nvector_common2([V|R], [MIN|S], [PROD|T], [prod*V-Q+E]) :-
Q is PROD*MIN,
nvector_common2(R, S, T, E).

stretch_lmin([], []) :-
!.
stretch_lmin([0|R], [1|S]) :-
!,
stretch_lmin(R, S).
stretch_lmin([L|R], [L|S]) :-
  L > 0,
  stretch_lmin(R, S).

stretch_reduce_lmax([], _, []).
stretch_reduce_lmax([L|R], N, [M|S]) :-
  M is min(L,N),
  stretch_reduce_lmax(R, N, S).

stretch_gen_states([], [], _, _, [sink(s),source(s)]).
stretch_gen_states([LMIN|LMINs], [LMAX|LMAXs], NVAR, I, STATES) :-
  LMIN =< LMAX,
  (LMIN =< 1, LMAX >= NVAR -> STATES1 = [] ; stretch_gen_states1(LMIN, LMAX, I, STATES1),
  I1 is I+1,
  stretch_gen_states(LMINs, LMAXs, NVAR, I1, STATES2),
  append(STATES1, STATES2, STATES)).

stretch_gen_states1(LCUR, LMAX, _, []) :-
  LCUR > LMAX, !.
stretch_gen_states1(LCUR, LMAX, I, [sink(S)|R]) :-
  LCUR =< LMAX,
  gen_automaton_state('s',I,LCUR,S),
  LCUR1 is LCUR+1,
  stretch_gen_states1(LCUR1, LMAX, I, R).

stretch_gen_transitions(I, M, [], [], _, _, _, [arc(s,0,s)]) :-
  I > M, !.
stretch_gen_transitions(I, M, [LMIN|LMINs], [LMAX|LMAXs], LLMIN, LLMAX, NVAR, TRANSITIONS) :-
  I =< M,
  (LMIN =< 1, LMAX >= NVAR ->
   T0 = [arc(s,I,s)],
   LT1 = [],
   LT2 = []
   ;
   gen_automaton_state('s',I,1,S_I_1),
   T0 = [arc(s,I,S_I_1)],
   stretch_gen_transitions1(1, LMAX, LMIN, I, M, LLMIN, LLMAX, NVAR, LT1),
   LMAX1 is LMAX-1,
   stretch_gen_transitions2(1, LMAX1, I, M, LT2)
  ),
  I1 is I+1,
  stretch_gen_transitions(I1, M, LMINs, LMAXs, LLMIN, LLMAX, NVAR, LT0),
  append(T0, LT0, T1),
  append(T1, LT1, T2),
  append(T2, LT2, TRANSITIONS).
APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

stretch_gen_transitions1(J, LMAX, _, _, _, _, _, [], []) :-
    J > LMAX, !.
stretch_gen_transitions1(J, LMAX, LMIN, I, M, LLMIN, LLMAX, NVAR, TRANSITIONS) :-
    J =< LMAX,
    gen_automaton_state('s',I,J,S_I_J),
    (J =< LMIN -> stretch_gen_transitions11(1, M, I, J, LLMIN, LLMAX, NVAR, T1) ; T1 = LT0),
    (J =< LMIN -> append([arc(S_I_J,0,s)], LT0, T1) ; T1 = LT0),
    append(T1, LT1, TRANSITIONS).

stretch_gen_transitions11(K, M, _, _, _, _, _, [], []) :-
    K > M, !.
stretch_gen_transitions11(K, M, I, J, [LMINs], [LMAXs], NVAR, R) :-
    K = I, !,
    K1 is K+1,
    stretch_gen_transitions11(K1, M, I, J, LMINs, LMAXs, NVAR, R).
stretch_gen_transitions11(K, M, I, J, [LMIN|LMINs], [LMAX|LMAXs], NVAR, [arc(S_I_J,K,S_K_1)|R]) :-
    K =< M,
    K =\= I,
    gen_automaton_state('s',I,J,S_I_J),
    (LMIN =< 1, LMAX >= NVAR ->
        S_K_1 = 's' ;
        gen_automaton_state('s',K,1,S_K_1)
    ),
    K1 is K+1,
    stretch_gen_transitions11(K1, M, I, J, LMINs, LMAXs, NVAR, R).

stretch_gen_transitions2(J, LMAX, _, _, []) :-
    J > LMAX, !.
stretch_gen_transitions2(J, LMAX, I, M, [arc(S_I_J,I,S_I_J1)|R]) :-
    J =< LMAX,
    gen_automaton_state('s',I,J,S_I_J),
    J1 is J+1,
    gen_automaton_state('s',I,J1,S_I_J1),
    stretch_gen_transitions2(J1, LMAX, I, M, R).

symmetric_alldifferent0(NODES, SNODES) :-
symmetric_alldifferent0a(NODES, L),
sort(L, S),
symmetric_alldifferent0a(SNODES, S).
symmetric_alldifferent0a([], []).  
symmetric_alldifferent0a([[index-INDEX, succ-SUCC]|R], [INDEX-SUCC|S]) :-  
symmetric_alldifferent0a(R, S).  

symmetric_alldifferent1([], _, _).  
symmetric_alldifferent1([S|R], I, SUCCS) :-  
symmetric_alldifferent2(SUCCS, 1, S, I),  
I1 is I+1,  
symmetric_alldifferent1(R, I1, SUCCS).  

symmetric_alldifferent2([], _, _, _).  
symmetric_alldifferent2([S|RS], J, S, I) :-  
S #= J #=> Sj #= I,  
J1 is J+1,  
symmetric_alldifferent2(RS, J1, S, I).  

derangement1([], []).  
derangement1([S|R], [I|T]) :-  
S #\= I,  
derangement1(R, T).
Appendix C

Systems Correspondence Tables
C.1 From the Catalog to Choco
<table>
<thead>
<tr>
<th>Catalog</th>
<th>Choco</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs_value</td>
<td>abs</td>
</tr>
<tr>
<td>all_equal</td>
<td>atMostNValue</td>
</tr>
<tr>
<td>allDifferent</td>
<td>allDifferent</td>
</tr>
<tr>
<td>among</td>
<td>among</td>
</tr>
<tr>
<td>among_var</td>
<td>among</td>
</tr>
<tr>
<td>and</td>
<td>reifiedAnd</td>
</tr>
<tr>
<td>arith</td>
<td>eq</td>
</tr>
<tr>
<td></td>
<td>neq</td>
</tr>
<tr>
<td></td>
<td>geq</td>
</tr>
<tr>
<td></td>
<td>gt</td>
</tr>
<tr>
<td></td>
<td>leq</td>
</tr>
<tr>
<td></td>
<td>lt</td>
</tr>
<tr>
<td>atleast</td>
<td>occurrenceMin</td>
</tr>
<tr>
<td>atmost</td>
<td>occurrenceMax</td>
</tr>
<tr>
<td>atmost_nvalue</td>
<td>atMostNValue</td>
</tr>
<tr>
<td>bin_packing</td>
<td>pack</td>
</tr>
<tr>
<td>bin_packing_capa</td>
<td>pack</td>
</tr>
<tr>
<td>clause_and</td>
<td>reifiedAnd</td>
</tr>
<tr>
<td>clause_or</td>
<td>clause</td>
</tr>
<tr>
<td>count</td>
<td>occurrence</td>
</tr>
<tr>
<td>cumulative</td>
<td>cumulativeMax</td>
</tr>
<tr>
<td>cumulative_two_d</td>
<td>geost</td>
</tr>
<tr>
<td>decreasing</td>
<td>increasingNValue</td>
</tr>
<tr>
<td>diffn</td>
<td>geost</td>
</tr>
<tr>
<td>disjoint_tasks</td>
<td>disjoint</td>
</tr>
<tr>
<td>disjunctive</td>
<td>disjunctive</td>
</tr>
<tr>
<td>distance</td>
<td>distanceEQ</td>
</tr>
<tr>
<td>domain</td>
<td>member</td>
</tr>
<tr>
<td>domain_constraint</td>
<td>domainChanneling</td>
</tr>
<tr>
<td>elem</td>
<td>nth</td>
</tr>
<tr>
<td>element</td>
<td>nth</td>
</tr>
<tr>
<td>element_matrix</td>
<td>nth</td>
</tr>
<tr>
<td>eq</td>
<td>eq</td>
</tr>
<tr>
<td>eq_set</td>
<td>eq</td>
</tr>
<tr>
<td>equivalent</td>
<td>ifOnlyIf</td>
</tr>
<tr>
<td>exactly</td>
<td>occurrence</td>
</tr>
<tr>
<td>geost</td>
<td>geost</td>
</tr>
<tr>
<td>geost_time</td>
<td>geost</td>
</tr>
<tr>
<td>geq</td>
<td>geq</td>
</tr>
<tr>
<td>global_cardinality</td>
<td>globalCardinality</td>
</tr>
<tr>
<td>global_cardinality_low_up</td>
<td>globalCardinality</td>
</tr>
<tr>
<td>gt</td>
<td>gt</td>
</tr>
<tr>
<td>imply</td>
<td>reifiedLeftImp</td>
</tr>
<tr>
<td>in</td>
<td>member</td>
</tr>
<tr>
<td>in_interval</td>
<td>member</td>
</tr>
<tr>
<td>Catalog</td>
<td>Choco</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>in_relation</td>
<td>feasPairAC</td>
</tr>
<tr>
<td></td>
<td>infeasPairAC</td>
</tr>
<tr>
<td></td>
<td>relationPairAC</td>
</tr>
<tr>
<td></td>
<td>feasTupleAC</td>
</tr>
<tr>
<td></td>
<td>infeasTupleAC</td>
</tr>
<tr>
<td></td>
<td>relationTupleAC</td>
</tr>
<tr>
<td>in_set</td>
<td>member</td>
</tr>
<tr>
<td>increasing</td>
<td>increasingNValue</td>
</tr>
<tr>
<td>increasing_nvalue</td>
<td>increasingNValue</td>
</tr>
<tr>
<td>inverse</td>
<td>inverseChanneling</td>
</tr>
<tr>
<td>inverse_offset</td>
<td>inverseChanneling</td>
</tr>
<tr>
<td>inverse_set</td>
<td>inverseSet</td>
</tr>
<tr>
<td>leq</td>
<td>leq</td>
</tr>
<tr>
<td>lex_between</td>
<td>lexChainEq</td>
</tr>
<tr>
<td>lex_chain_less</td>
<td>lexChain</td>
</tr>
<tr>
<td>lex_chain_leq</td>
<td>lexChainEq</td>
</tr>
<tr>
<td>lex_greater</td>
<td>lex</td>
</tr>
<tr>
<td>lex_greater_eq</td>
<td>lexEq</td>
</tr>
<tr>
<td>lex_less</td>
<td>lex</td>
</tr>
<tr>
<td>lex_leq</td>
<td>lexEq</td>
</tr>
<tr>
<td>lex_leq_allperm</td>
<td>leximin</td>
</tr>
<tr>
<td>lt</td>
<td>lt</td>
</tr>
<tr>
<td>maximum</td>
<td>max</td>
</tr>
<tr>
<td>minimum</td>
<td>min</td>
</tr>
<tr>
<td>nand</td>
<td>clause</td>
</tr>
<tr>
<td>neq</td>
<td>neq</td>
</tr>
<tr>
<td>nor</td>
<td>reifiedXnor</td>
</tr>
<tr>
<td>not_in</td>
<td>notMember</td>
</tr>
<tr>
<td>or</td>
<td>reifiedOr</td>
</tr>
<tr>
<td>proper_forest</td>
<td>tree</td>
</tr>
<tr>
<td>scalar_product</td>
<td>equation</td>
</tr>
<tr>
<td>set_disjoint</td>
<td>disjoint</td>
</tr>
<tr>
<td>sort</td>
<td>sorting</td>
</tr>
<tr>
<td>stretch_path</td>
<td>stretchPath</td>
</tr>
<tr>
<td>strictly_decreasing</td>
<td>increasingNValue</td>
</tr>
<tr>
<td>strictly_increasing</td>
<td>increasingNValue</td>
</tr>
<tr>
<td>sum_ctr</td>
<td>equation</td>
</tr>
<tr>
<td>tree</td>
<td>tree</td>
</tr>
<tr>
<td>xor</td>
<td>reifiedXor</td>
</tr>
</tbody>
</table>
C.2 From the Catalog to Gecode
<table>
<thead>
<tr>
<th>Catalog</th>
<th>Gecode</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs_value</td>
<td>abs</td>
</tr>
<tr>
<td>all_equal</td>
<td>rel</td>
</tr>
<tr>
<td>alldifferent</td>
<td>linear</td>
</tr>
<tr>
<td>alldifferent_cst</td>
<td>linear</td>
</tr>
<tr>
<td>among</td>
<td>count</td>
</tr>
<tr>
<td>among_seq</td>
<td>sequence</td>
</tr>
<tr>
<td>among_var</td>
<td>count</td>
</tr>
<tr>
<td>and</td>
<td>rel</td>
</tr>
<tr>
<td>arith</td>
<td>rel</td>
</tr>
<tr>
<td>atleast</td>
<td>count</td>
</tr>
<tr>
<td>atmost</td>
<td>count</td>
</tr>
<tr>
<td>bin_packing</td>
<td>binpacking</td>
</tr>
<tr>
<td>bin_packing_capa</td>
<td>binpacking</td>
</tr>
<tr>
<td>circuit</td>
<td>circuit</td>
</tr>
<tr>
<td>clause_and</td>
<td>clause</td>
</tr>
<tr>
<td>clause_or</td>
<td>clause</td>
</tr>
<tr>
<td>count</td>
<td>count</td>
</tr>
<tr>
<td>counts</td>
<td>count</td>
</tr>
<tr>
<td>cumulative</td>
<td>cumulative</td>
</tr>
<tr>
<td>cumulatives</td>
<td>cumulatives</td>
</tr>
<tr>
<td>decreasing</td>
<td>rel</td>
</tr>
<tr>
<td>diffn</td>
<td>nooverlap</td>
</tr>
<tr>
<td>disjunctive</td>
<td>unary</td>
</tr>
<tr>
<td>domain</td>
<td>dom</td>
</tr>
<tr>
<td>domain_constraint</td>
<td>channel</td>
</tr>
<tr>
<td>elem</td>
<td>element</td>
</tr>
<tr>
<td>element</td>
<td>element</td>
</tr>
<tr>
<td>element_matrix</td>
<td>element</td>
</tr>
<tr>
<td>eq</td>
<td>rel</td>
</tr>
<tr>
<td>eq_set</td>
<td>rel</td>
</tr>
<tr>
<td>equivalent</td>
<td>rel</td>
</tr>
<tr>
<td>exactly</td>
<td>count</td>
</tr>
<tr>
<td>geq</td>
<td>rel</td>
</tr>
<tr>
<td>global_cardinality</td>
<td>count</td>
</tr>
<tr>
<td>gt</td>
<td>rel</td>
</tr>
<tr>
<td>imply</td>
<td>rel</td>
</tr>
<tr>
<td>in</td>
<td>rel</td>
</tr>
<tr>
<td>in_interval</td>
<td>dom</td>
</tr>
<tr>
<td>in_intervals</td>
<td>dom</td>
</tr>
<tr>
<td>in_relation</td>
<td>extensional</td>
</tr>
<tr>
<td>in_set</td>
<td>rel</td>
</tr>
<tr>
<td>increasing</td>
<td>rel</td>
</tr>
<tr>
<td>int_value_precede</td>
<td>precede</td>
</tr>
<tr>
<td>Catalog</td>
<td>Gecode</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>int_value_precede_chain</td>
<td>precede</td>
</tr>
<tr>
<td>inverse</td>
<td>channel</td>
</tr>
<tr>
<td>inverse_offset</td>
<td>channel</td>
</tr>
<tr>
<td>leq</td>
<td>rel</td>
</tr>
<tr>
<td>lex</td>
<td>lex</td>
</tr>
<tr>
<td>lex_greater</td>
<td>rel</td>
</tr>
<tr>
<td>lex_greatereq</td>
<td>rel</td>
</tr>
<tr>
<td>lex_less</td>
<td>rel</td>
</tr>
<tr>
<td>lex_leqseq</td>
<td>rel</td>
</tr>
<tr>
<td>link_set_to_booleans</td>
<td>channel</td>
</tr>
<tr>
<td>lt</td>
<td>rel</td>
</tr>
<tr>
<td>maximum</td>
<td>max</td>
</tr>
<tr>
<td>minimum</td>
<td>min</td>
</tr>
<tr>
<td>nand</td>
<td>clause</td>
</tr>
<tr>
<td>neq</td>
<td>rel</td>
</tr>
<tr>
<td>nor</td>
<td>clause</td>
</tr>
<tr>
<td>not_all_equal</td>
<td>rel</td>
</tr>
<tr>
<td>not_in</td>
<td>rel</td>
</tr>
<tr>
<td>nvalue</td>
<td>nvalues</td>
</tr>
<tr>
<td>nvalues</td>
<td>nvalues</td>
</tr>
<tr>
<td>or</td>
<td>rel</td>
</tr>
<tr>
<td>roots</td>
<td>roots</td>
</tr>
<tr>
<td>scalar_product</td>
<td>linear</td>
</tr>
<tr>
<td>set_value_precede</td>
<td>precede</td>
</tr>
<tr>
<td>sort</td>
<td>sorted</td>
</tr>
<tr>
<td>sort_permutation</td>
<td>sorted</td>
</tr>
<tr>
<td>strictly_decreasing</td>
<td>rel</td>
</tr>
<tr>
<td>strictly_increasing</td>
<td>rel</td>
</tr>
<tr>
<td>sum_ctr</td>
<td>linear</td>
</tr>
<tr>
<td>sum_set</td>
<td>weights</td>
</tr>
<tr>
<td>xor</td>
<td>rel</td>
</tr>
</tbody>
</table>
C.3 From the Catalog to JaCoP

<table>
<thead>
<tr>
<th>Catalog</th>
<th>JaCoP</th>
</tr>
</thead>
<tbody>
<tr>
<td>alldifferent</td>
<td>alldifferent</td>
</tr>
<tr>
<td></td>
<td>alldiff</td>
</tr>
<tr>
<td></td>
<td>alldistinct</td>
</tr>
<tr>
<td>among</td>
<td>among</td>
</tr>
<tr>
<td>among_seq</td>
<td>sequence</td>
</tr>
<tr>
<td>among_var</td>
<td>amongvar</td>
</tr>
<tr>
<td>and</td>
<td>andbool</td>
</tr>
<tr>
<td>atleast</td>
<td>count</td>
</tr>
<tr>
<td>atmost</td>
<td>count</td>
</tr>
<tr>
<td>circuit</td>
<td>circuit</td>
</tr>
<tr>
<td>count</td>
<td>count</td>
</tr>
<tr>
<td>cumulative</td>
<td>cumulative</td>
</tr>
<tr>
<td>diffn</td>
<td>diff2</td>
</tr>
<tr>
<td></td>
<td>diff</td>
</tr>
<tr>
<td></td>
<td>disjoint</td>
</tr>
<tr>
<td></td>
<td>disjointconditional</td>
</tr>
<tr>
<td>distance</td>
<td>distance</td>
</tr>
<tr>
<td></td>
<td>distance2</td>
</tr>
<tr>
<td>elem</td>
<td>element</td>
</tr>
<tr>
<td>element</td>
<td>element</td>
</tr>
<tr>
<td>eq</td>
<td>xeqy</td>
</tr>
<tr>
<td>equivalent</td>
<td>eqbool</td>
</tr>
<tr>
<td>exactly</td>
<td>count</td>
</tr>
<tr>
<td>geost</td>
<td>geost</td>
</tr>
<tr>
<td>geost_time</td>
<td>geost</td>
</tr>
<tr>
<td>geq</td>
<td>xgteqy</td>
</tr>
<tr>
<td>global_cardinality</td>
<td>gcc</td>
</tr>
<tr>
<td>gt</td>
<td>xgty</td>
</tr>
<tr>
<td>imply</td>
<td>ifthenbool</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
</tr>
<tr>
<td>in_interval</td>
<td>in</td>
</tr>
<tr>
<td>in_intervals</td>
<td>in</td>
</tr>
<tr>
<td>in_relation</td>
<td>extensionalsupportVA</td>
</tr>
<tr>
<td></td>
<td>extensionalsupportMDD</td>
</tr>
<tr>
<td></td>
<td>extensionalsupportSTR</td>
</tr>
<tr>
<td>leq</td>
<td>xlteqy</td>
</tr>
<tr>
<td>lt</td>
<td>xltb</td>
</tr>
<tr>
<td>maximum</td>
<td>max</td>
</tr>
<tr>
<td>minimum</td>
<td>min</td>
</tr>
<tr>
<td>or</td>
<td>orbool</td>
</tr>
<tr>
<td>power</td>
<td>xexpyeqz</td>
</tr>
<tr>
<td>scalar_product</td>
<td>sumweight</td>
</tr>
<tr>
<td>stretch_path</td>
<td>stretch</td>
</tr>
<tr>
<td>xor</td>
<td>xorbool</td>
</tr>
</tbody>
</table>
### C.4 From the Catalog to MiniZinc

<table>
<thead>
<tr>
<th>Catalog</th>
<th>MiniZinc</th>
</tr>
</thead>
<tbody>
<tr>
<td>all_equal</td>
<td>all_equal</td>
</tr>
<tr>
<td>all_different</td>
<td>all_different</td>
</tr>
<tr>
<td>among</td>
<td>among</td>
</tr>
<tr>
<td>at_least</td>
<td>at_least</td>
</tr>
<tr>
<td>at_most</td>
<td>at_most</td>
</tr>
<tr>
<td>at_most1</td>
<td>at_most1</td>
</tr>
<tr>
<td>bin_packing</td>
<td>bin_packing</td>
</tr>
<tr>
<td>bin_packing_capa</td>
<td>bin_packing_capa</td>
</tr>
<tr>
<td>count</td>
<td>count</td>
</tr>
<tr>
<td>cumulative</td>
<td>cumulative</td>
</tr>
<tr>
<td>decreasing</td>
<td>decreasing</td>
</tr>
<tr>
<td>diffn</td>
<td>diffn</td>
</tr>
<tr>
<td>element</td>
<td>element</td>
</tr>
<tr>
<td>exactly</td>
<td>exactly</td>
</tr>
<tr>
<td>global_cardinality</td>
<td>global_cardinality</td>
</tr>
<tr>
<td>global_cardinality_low_up</td>
<td>global_cardinality_low_up</td>
</tr>
<tr>
<td>in</td>
<td>member</td>
</tr>
<tr>
<td>in_relation</td>
<td>table</td>
</tr>
<tr>
<td>increasing</td>
<td>increasing</td>
</tr>
<tr>
<td>int_value_precede</td>
<td>value_precede</td>
</tr>
<tr>
<td>int_value_precede_chain</td>
<td>value_precede_chain</td>
</tr>
<tr>
<td>inverse</td>
<td>inverse</td>
</tr>
<tr>
<td>inverse_set</td>
<td>inverse_set</td>
</tr>
<tr>
<td>lex2</td>
<td>lex2</td>
</tr>
<tr>
<td>lex_greater</td>
<td>lex_greater</td>
</tr>
<tr>
<td>lex_greatereq</td>
<td>lex_greatereq</td>
</tr>
<tr>
<td>lex_less</td>
<td>lex_less</td>
</tr>
<tr>
<td>lex_lesseq</td>
<td>lex_lesseq</td>
</tr>
<tr>
<td>link_set_to_booleans</td>
<td>link_set_to_booleans</td>
</tr>
<tr>
<td>maximum</td>
<td>maximum</td>
</tr>
<tr>
<td>minimum</td>
<td>minimum</td>
</tr>
<tr>
<td>nvalue</td>
<td>nvalue</td>
</tr>
<tr>
<td>roots</td>
<td>roots</td>
</tr>
<tr>
<td>sliding_sum</td>
<td>sliding_sum</td>
</tr>
<tr>
<td>sort</td>
<td>sort</td>
</tr>
<tr>
<td>strict_lex2</td>
<td>strict_lex2</td>
</tr>
<tr>
<td>sum</td>
<td>sum_pred</td>
</tr>
</tbody>
</table>
C.5 From the Catalog to SICStus
<table>
<thead>
<tr>
<th>Catalog</th>
<th>SICStus</th>
</tr>
</thead>
<tbody>
<tr>
<td>alldifferent</td>
<td>all_different all_distinct</td>
</tr>
<tr>
<td>and</td>
<td>#/\</td>
</tr>
<tr>
<td>arith</td>
<td>#&lt; #=&lt; %&gt; #&gt;= #= #\</td>
</tr>
<tr>
<td>atleast</td>
<td>count</td>
</tr>
<tr>
<td>atmost</td>
<td>count</td>
</tr>
<tr>
<td>circuit</td>
<td>circuit</td>
</tr>
<tr>
<td>count</td>
<td>count</td>
</tr>
<tr>
<td>cumulative</td>
<td>cumulative</td>
</tr>
<tr>
<td>cumulatives</td>
<td>cumulatives</td>
</tr>
<tr>
<td>domain</td>
<td>domain</td>
</tr>
<tr>
<td>domain_constraint</td>
<td>in in_set</td>
</tr>
<tr>
<td>elem</td>
<td>element</td>
</tr>
<tr>
<td>element</td>
<td>element</td>
</tr>
<tr>
<td>eq</td>
<td>#=</td>
</tr>
<tr>
<td>equivalent</td>
<td>#&lt;==&gt;</td>
</tr>
<tr>
<td>exactly</td>
<td>count</td>
</tr>
<tr>
<td>geost</td>
<td>geost</td>
</tr>
<tr>
<td>gt</td>
<td>#&gt;</td>
</tr>
<tr>
<td>imply</td>
<td>#&gt;=</td>
</tr>
<tr>
<td>in</td>
<td>in in_set</td>
</tr>
<tr>
<td>in_interval</td>
<td>in</td>
</tr>
<tr>
<td>in_intervals</td>
<td>in</td>
</tr>
<tr>
<td>in_relation</td>
<td>case relation</td>
</tr>
<tr>
<td></td>
<td>table</td>
</tr>
<tr>
<td>inverse</td>
<td>assignment</td>
</tr>
<tr>
<td>leq</td>
<td>#=&lt;</td>
</tr>
<tr>
<td>lex_between</td>
<td>lex_chain</td>
</tr>
<tr>
<td>lex_chain_less</td>
<td>lex_chain</td>
</tr>
<tr>
<td>lex_chain_lesseq</td>
<td>lex_chain</td>
</tr>
<tr>
<td>lex_greater</td>
<td>lex_chain</td>
</tr>
<tr>
<td>lex_greatereq</td>
<td>lex_chain</td>
</tr>
<tr>
<td>lex_less</td>
<td>lex_chain</td>
</tr>
<tr>
<td>lex_lesseq</td>
<td>lex_chain</td>
</tr>
<tr>
<td>lt</td>
<td>#&lt;</td>
</tr>
<tr>
<td>maximum</td>
<td>maximum</td>
</tr>
<tr>
<td>minimum</td>
<td>minimum</td>
</tr>
</tbody>
</table>
## APPENDIX B. ELECTRONIC CONSTRAINT CATALOGUE

<table>
<thead>
<tr>
<th>Catalog</th>
<th>SICStus</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimum_weight_alldifferent</td>
<td>all_different</td>
</tr>
<tr>
<td></td>
<td>all_distinct</td>
</tr>
<tr>
<td>nand</td>
<td>#/\</td>
</tr>
<tr>
<td>neq</td>
<td>#=</td>
</tr>
<tr>
<td>nor</td>
<td>#/</td>
</tr>
<tr>
<td>nvalue</td>
<td>nvalue</td>
</tr>
<tr>
<td>or</td>
<td>#/\</td>
</tr>
<tr>
<td>scalar_product</td>
<td>scalar_product</td>
</tr>
<tr>
<td>sort</td>
<td>sorting</td>
</tr>
<tr>
<td>sort_permutation</td>
<td>sorting</td>
</tr>
<tr>
<td>sumCtr</td>
<td>scalar_product</td>
</tr>
<tr>
<td>xor</td>
<td>#\</td>
</tr>
</tbody>
</table>
C.6 From Choco to the Catalog
<table>
<thead>
<tr>
<th>Choco</th>
<th>Catalog</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>abs.value</td>
</tr>
<tr>
<td>allDifferent</td>
<td>alldifferent</td>
</tr>
<tr>
<td>among</td>
<td>among</td>
</tr>
<tr>
<td>atMostNValue</td>
<td>all_equal</td>
</tr>
<tr>
<td>clause</td>
<td>clause_and</td>
</tr>
<tr>
<td>cumulativeMax</td>
<td>cumulative</td>
</tr>
<tr>
<td>disjoint</td>
<td>disjoint_tasks</td>
</tr>
<tr>
<td>disjunctive</td>
<td>disjunctive</td>
</tr>
<tr>
<td>distanceEQ</td>
<td>distance</td>
</tr>
<tr>
<td>domainChanneling</td>
<td>domain.constraint</td>
</tr>
<tr>
<td>eq</td>
<td>arith</td>
</tr>
<tr>
<td>equation</td>
<td>scalar_product</td>
</tr>
<tr>
<td>feasPairAC</td>
<td>in_relation</td>
</tr>
<tr>
<td>feasTupleAC</td>
<td>in_relation</td>
</tr>
<tr>
<td>geost</td>
<td>cumulative_two_d</td>
</tr>
<tr>
<td>geq</td>
<td>arith</td>
</tr>
<tr>
<td>globalCardinality</td>
<td>global_cardinality</td>
</tr>
<tr>
<td>gt</td>
<td>arith</td>
</tr>
<tr>
<td>ifOnlyIf</td>
<td>equivalent</td>
</tr>
<tr>
<td>increasingNValue</td>
<td>decreasing</td>
</tr>
<tr>
<td>inffeasPairAC</td>
<td>in_relation</td>
</tr>
<tr>
<td>inffeasTupleAC</td>
<td>in_relation</td>
</tr>
<tr>
<td>inverseChanneling</td>
<td>inverse</td>
</tr>
<tr>
<td>inverseSet</td>
<td>inverse_set</td>
</tr>
<tr>
<td>leq</td>
<td>arith</td>
</tr>
<tr>
<td>lex</td>
<td>lex_greater</td>
</tr>
<tr>
<td>lexChain</td>
<td>lex.chain_less</td>
</tr>
<tr>
<td>Choco</td>
<td>Catalog</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>lexChainEq</td>
<td>lex_between</td>
</tr>
<tr>
<td></td>
<td>lex_chain_leseq</td>
</tr>
<tr>
<td>lexEq</td>
<td>lex_greatereq</td>
</tr>
<tr>
<td></td>
<td>lex_leseq</td>
</tr>
<tr>
<td>leximin</td>
<td>lex_leseq_allperm</td>
</tr>
<tr>
<td>lt</td>
<td>arith</td>
</tr>
<tr>
<td></td>
<td>lt</td>
</tr>
<tr>
<td>max</td>
<td>maximum</td>
</tr>
<tr>
<td>member</td>
<td>domain</td>
</tr>
<tr>
<td></td>
<td>in</td>
</tr>
<tr>
<td></td>
<td>in_interval</td>
</tr>
<tr>
<td></td>
<td>in_set</td>
</tr>
<tr>
<td>min</td>
<td>minimum</td>
</tr>
<tr>
<td>neq</td>
<td>arith</td>
</tr>
<tr>
<td></td>
<td>neq</td>
</tr>
<tr>
<td>notMember</td>
<td>not_in</td>
</tr>
<tr>
<td>nth</td>
<td>elem</td>
</tr>
<tr>
<td></td>
<td>element</td>
</tr>
<tr>
<td></td>
<td>element_matrix</td>
</tr>
<tr>
<td>occurrence</td>
<td>count</td>
</tr>
<tr>
<td></td>
<td>exactly</td>
</tr>
<tr>
<td>occurrenceMax</td>
<td>atmost</td>
</tr>
<tr>
<td>occurrenceMin</td>
<td>atleast</td>
</tr>
<tr>
<td>pack</td>
<td>bin_packing</td>
</tr>
<tr>
<td></td>
<td>bin_packing_capa</td>
</tr>
<tr>
<td>reifiedAnd</td>
<td>and</td>
</tr>
<tr>
<td></td>
<td>clause_and</td>
</tr>
<tr>
<td>reifiedLeftImp</td>
<td>imply</td>
</tr>
<tr>
<td>reifiedOr</td>
<td>clause_or</td>
</tr>
<tr>
<td></td>
<td>or</td>
</tr>
<tr>
<td>reifiedXnor</td>
<td>nor</td>
</tr>
<tr>
<td>reifiedXor</td>
<td>xor</td>
</tr>
<tr>
<td>relationPairAC</td>
<td>in_relation</td>
</tr>
<tr>
<td>relationTupleAC</td>
<td>in_relation</td>
</tr>
<tr>
<td>sorting</td>
<td>sort</td>
</tr>
<tr>
<td>stretchPath</td>
<td>stretch_path</td>
</tr>
<tr>
<td>tree</td>
<td>proper_forest</td>
</tr>
<tr>
<td></td>
<td>tree</td>
</tr>
</tbody>
</table>
C.7 From Gecode to the Catalog
<table>
<thead>
<tr>
<th>Gecode</th>
<th>Catalog</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>abs.value</td>
</tr>
<tr>
<td>atleast</td>
<td>atleast</td>
</tr>
<tr>
<td>atmost</td>
<td>atmost</td>
</tr>
<tr>
<td>binpacking</td>
<td>bin_packing</td>
</tr>
<tr>
<td></td>
<td>bin_packing_capa</td>
</tr>
<tr>
<td>channel</td>
<td>domain_constraint</td>
</tr>
<tr>
<td></td>
<td>inverse</td>
</tr>
<tr>
<td></td>
<td>inverse_offset</td>
</tr>
<tr>
<td>channel</td>
<td>link_set_to_booleans</td>
</tr>
<tr>
<td>circuit</td>
<td>circuit</td>
</tr>
<tr>
<td>clause</td>
<td>clause_and</td>
</tr>
<tr>
<td></td>
<td>clause_or</td>
</tr>
<tr>
<td></td>
<td>nand</td>
</tr>
<tr>
<td></td>
<td>nor</td>
</tr>
<tr>
<td>count</td>
<td>among</td>
</tr>
<tr>
<td></td>
<td>among_var</td>
</tr>
<tr>
<td></td>
<td>atleast</td>
</tr>
<tr>
<td></td>
<td>atmost</td>
</tr>
<tr>
<td></td>
<td>count</td>
</tr>
<tr>
<td></td>
<td>counts</td>
</tr>
<tr>
<td></td>
<td>exactly</td>
</tr>
<tr>
<td></td>
<td>global_cardinality</td>
</tr>
<tr>
<td>cumulative</td>
<td>cumulative</td>
</tr>
<tr>
<td>cumulatives</td>
<td>cumulatives</td>
</tr>
<tr>
<td>dom</td>
<td>domain</td>
</tr>
<tr>
<td></td>
<td>in</td>
</tr>
<tr>
<td></td>
<td>in_interval</td>
</tr>
<tr>
<td></td>
<td>in_intervals</td>
</tr>
<tr>
<td></td>
<td>in_set</td>
</tr>
<tr>
<td>element</td>
<td>elem</td>
</tr>
<tr>
<td></td>
<td>element</td>
</tr>
<tr>
<td>element</td>
<td>element_matrix</td>
</tr>
<tr>
<td>exactly</td>
<td>exactly</td>
</tr>
<tr>
<td>extensional</td>
<td>in_relation</td>
</tr>
<tr>
<td>lex</td>
<td>lex</td>
</tr>
<tr>
<td>linear</td>
<td>alldifferent</td>
</tr>
<tr>
<td></td>
<td>alldifferent_cst</td>
</tr>
<tr>
<td>linear</td>
<td>scalar.product</td>
</tr>
<tr>
<td></td>
<td>sum_ctr</td>
</tr>
<tr>
<td>max</td>
<td>maximum</td>
</tr>
<tr>
<td>min</td>
<td>minimum</td>
</tr>
<tr>
<td>nooverlap</td>
<td>diffn</td>
</tr>
<tr>
<td>nvalues</td>
<td>nvalue</td>
</tr>
<tr>
<td></td>
<td>nvalues</td>
</tr>
<tr>
<td>precede</td>
<td>int_value_precede</td>
</tr>
<tr>
<td></td>
<td>int_value_precede_chain</td>
</tr>
<tr>
<td>precede</td>
<td>set_value_precede</td>
</tr>
<tr>
<td>Gecode</td>
<td>Catalog</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>rel</td>
<td>and</td>
</tr>
<tr>
<td></td>
<td>equivalent</td>
</tr>
<tr>
<td></td>
<td>imply</td>
</tr>
<tr>
<td></td>
<td>or</td>
</tr>
<tr>
<td></td>
<td>xor</td>
</tr>
<tr>
<td>rel</td>
<td>all_equal</td>
</tr>
<tr>
<td></td>
<td>arith</td>
</tr>
<tr>
<td></td>
<td>decreasing</td>
</tr>
<tr>
<td></td>
<td>eq</td>
</tr>
<tr>
<td></td>
<td>geq</td>
</tr>
<tr>
<td></td>
<td>gt</td>
</tr>
<tr>
<td></td>
<td>increasing</td>
</tr>
<tr>
<td></td>
<td>leq</td>
</tr>
<tr>
<td></td>
<td>lex_greater</td>
</tr>
<tr>
<td></td>
<td>lex_greatereq</td>
</tr>
<tr>
<td></td>
<td>lex_less</td>
</tr>
<tr>
<td></td>
<td>lex_lessseq</td>
</tr>
<tr>
<td></td>
<td>lt</td>
</tr>
<tr>
<td></td>
<td>neq</td>
</tr>
<tr>
<td></td>
<td>not_all_equal</td>
</tr>
<tr>
<td></td>
<td>strictly_decreasing</td>
</tr>
<tr>
<td></td>
<td>strictly_increasing</td>
</tr>
<tr>
<td>rel</td>
<td>eq_set</td>
</tr>
<tr>
<td></td>
<td>in</td>
</tr>
<tr>
<td></td>
<td>in_set</td>
</tr>
<tr>
<td></td>
<td>not_in</td>
</tr>
<tr>
<td>roots</td>
<td>roots</td>
</tr>
<tr>
<td>sequence</td>
<td>among_seq</td>
</tr>
<tr>
<td>sorted</td>
<td>sort</td>
</tr>
<tr>
<td></td>
<td>sort_permutation</td>
</tr>
<tr>
<td>unary</td>
<td>disjunctive</td>
</tr>
<tr>
<td>weights</td>
<td>sum_set</td>
</tr>
</tbody>
</table>
## C.8 From JaCoP to the Catalog

<table>
<thead>
<tr>
<th>JaCoP</th>
<th>Catalog</th>
</tr>
</thead>
<tbody>
<tr>
<td>alldiff</td>
<td>alldifferent</td>
</tr>
<tr>
<td>alldifferent</td>
<td>alldifferent</td>
</tr>
<tr>
<td>alldistinct</td>
<td>alldifferent</td>
</tr>
<tr>
<td>among</td>
<td>among</td>
</tr>
<tr>
<td>amongvar</td>
<td>among_var</td>
</tr>
<tr>
<td>andbool</td>
<td>and</td>
</tr>
<tr>
<td>circuit</td>
<td>circuit</td>
</tr>
<tr>
<td>count</td>
<td>at.least</td>
</tr>
<tr>
<td></td>
<td>at.most</td>
</tr>
<tr>
<td></td>
<td>count</td>
</tr>
<tr>
<td></td>
<td>exactly</td>
</tr>
<tr>
<td>cumulative</td>
<td>cumulative</td>
</tr>
<tr>
<td>diff</td>
<td>diffn</td>
</tr>
<tr>
<td>diff2</td>
<td>diffn</td>
</tr>
<tr>
<td>disjoint</td>
<td>diffn</td>
</tr>
<tr>
<td>disjointconditional</td>
<td>diffn</td>
</tr>
<tr>
<td>distance</td>
<td>distance</td>
</tr>
<tr>
<td>distance2</td>
<td>distance</td>
</tr>
<tr>
<td>element</td>
<td>elem</td>
</tr>
<tr>
<td>eqbool</td>
<td>equivalent</td>
</tr>
<tr>
<td>extensionalsupportMDD</td>
<td>in.relation</td>
</tr>
<tr>
<td>extensionalsupportSTR</td>
<td>in.relation</td>
</tr>
<tr>
<td>extensionalsupportVA</td>
<td>in.relation</td>
</tr>
<tr>
<td>gcc</td>
<td>global_cardinality</td>
</tr>
<tr>
<td>geost</td>
<td>geost</td>
</tr>
<tr>
<td></td>
<td>geost_time</td>
</tr>
<tr>
<td>ifthenbool</td>
<td>imply</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
</tr>
<tr>
<td></td>
<td>in.interval</td>
</tr>
<tr>
<td></td>
<td>in.intervals</td>
</tr>
<tr>
<td>max</td>
<td>maximum</td>
</tr>
<tr>
<td>min</td>
<td>minimum</td>
</tr>
<tr>
<td>orbool</td>
<td>or</td>
</tr>
<tr>
<td>sequence</td>
<td>among_seq</td>
</tr>
<tr>
<td>stretch</td>
<td>stretch_path</td>
</tr>
<tr>
<td>sumweight</td>
<td>scalar_product</td>
</tr>
<tr>
<td>xeqy</td>
<td>eq</td>
</tr>
<tr>
<td>xexpyeqz</td>
<td>power</td>
</tr>
<tr>
<td>xgteqy</td>
<td>geq</td>
</tr>
<tr>
<td>xgt</td>
<td>gt</td>
</tr>
<tr>
<td>xltgeqy</td>
<td>leq</td>
</tr>
<tr>
<td>xlt</td>
<td>lt</td>
</tr>
<tr>
<td>xorbool</td>
<td>xor</td>
</tr>
</tbody>
</table>
## C.9 From MiniZinc to the Catalog

<table>
<thead>
<tr>
<th>MiniZinc</th>
<th>Catalog</th>
</tr>
</thead>
<tbody>
<tr>
<td>all_different</td>
<td>allDifferent</td>
</tr>
<tr>
<td>all_equal</td>
<td>all_equal</td>
</tr>
<tr>
<td>among</td>
<td>among</td>
</tr>
<tr>
<td>at_least</td>
<td>at_least</td>
</tr>
<tr>
<td>at_most</td>
<td>at_most</td>
</tr>
<tr>
<td>at_most1</td>
<td>at_most1</td>
</tr>
<tr>
<td>bin_packing</td>
<td>bin_packing</td>
</tr>
<tr>
<td>bin_packing_capa</td>
<td>bin_packing_capa</td>
</tr>
<tr>
<td>count</td>
<td>count</td>
</tr>
<tr>
<td>cumulative</td>
<td>cumulative</td>
</tr>
<tr>
<td>decreasing</td>
<td>decreasing</td>
</tr>
<tr>
<td>diffn</td>
<td>diffn</td>
</tr>
<tr>
<td>element</td>
<td>element</td>
</tr>
<tr>
<td>exactly</td>
<td>exactly</td>
</tr>
<tr>
<td>global_cardinality</td>
<td>global_cardinality</td>
</tr>
<tr>
<td>global_cardinality_low_up</td>
<td>global_cardinality_low_up</td>
</tr>
<tr>
<td>increasing</td>
<td>increasing</td>
</tr>
<tr>
<td>inverse</td>
<td>inverse</td>
</tr>
<tr>
<td>inverse_set</td>
<td>inverse_set</td>
</tr>
<tr>
<td>lex2</td>
<td>lex2</td>
</tr>
<tr>
<td>lex_greater</td>
<td>lex_greater</td>
</tr>
<tr>
<td>lex_greatereq</td>
<td>lex_greatereq</td>
</tr>
<tr>
<td>lex_less</td>
<td>lex_less</td>
</tr>
<tr>
<td>lex_less_eq</td>
<td>lex_less_eq</td>
</tr>
<tr>
<td>link_set_to_bools</td>
<td>link_set_to_bools</td>
</tr>
<tr>
<td>maximum</td>
<td>maximum</td>
</tr>
<tr>
<td>member</td>
<td>in</td>
</tr>
<tr>
<td>minimum</td>
<td>minimum</td>
</tr>
<tr>
<td>nvalue</td>
<td>nvalue</td>
</tr>
<tr>
<td>roots</td>
<td>roots</td>
</tr>
<tr>
<td>sliding_sum</td>
<td>sliding_sum</td>
</tr>
<tr>
<td>sort</td>
<td>sort</td>
</tr>
<tr>
<td>strict_lex2</td>
<td>strict_lex2</td>
</tr>
<tr>
<td>sum_pred</td>
<td>sum</td>
</tr>
<tr>
<td>table</td>
<td>in_relation</td>
</tr>
<tr>
<td>value_precede</td>
<td>int_value_precede</td>
</tr>
<tr>
<td>value_precede_chain</td>
<td>int_value_precede_chain</td>
</tr>
</tbody>
</table>
### C.10 From SICStus to the Catalog

<table>
<thead>
<tr>
<th>SICStus</th>
<th>Catalog</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#/\</code></td>
<td><code>and</code></td>
</tr>
<tr>
<td></td>
<td><code>nand</code></td>
</tr>
<tr>
<td><code>#&lt;</code></td>
<td><code>arith</code></td>
</tr>
<tr>
<td></td>
<td><code>lt</code></td>
</tr>
<tr>
<td><code>#&lt;&gt;=</code></td>
<td><code>equivalent</code></td>
</tr>
<tr>
<td><code>#=</code></td>
<td><code>arith</code></td>
</tr>
<tr>
<td></td>
<td><code>eq</code></td>
</tr>
<tr>
<td><code>#=&lt;</code></td>
<td><code>arith</code></td>
</tr>
<tr>
<td></td>
<td><code>leq</code></td>
</tr>
<tr>
<td><code>#=&gt;</code></td>
<td><code>imply</code></td>
</tr>
<tr>
<td><code>#&gt;</code></td>
<td><code>arith</code></td>
</tr>
<tr>
<td></td>
<td><code>gt</code></td>
</tr>
<tr>
<td><code>#&gt;=</code></td>
<td><code>arith</code></td>
</tr>
<tr>
<td></td>
<td><code>geq</code></td>
</tr>
<tr>
<td><code>#/\</code></td>
<td><code>xor</code></td>
</tr>
<tr>
<td><code>#/=</code></td>
<td><code>nor</code></td>
</tr>
<tr>
<td></td>
<td><code>or</code></td>
</tr>
<tr>
<td><code>#/=</code></td>
<td><code>arith</code></td>
</tr>
<tr>
<td></td>
<td><code>neq</code></td>
</tr>
<tr>
<td><code>all_different</code></td>
<td><code>alldifferent</code></td>
</tr>
<tr>
<td></td>
<td><code>minimum_weight_alldifferent</code></td>
</tr>
<tr>
<td><code>all_distinct</code></td>
<td><code>alldifferent</code></td>
</tr>
<tr>
<td></td>
<td><code>minimum_weight_alldifferent</code></td>
</tr>
<tr>
<td><code>assignment</code></td>
<td><code>inverse</code></td>
</tr>
<tr>
<td><code>case</code></td>
<td><code>in_relation</code></td>
</tr>
<tr>
<td><code>circuit</code></td>
<td><code>circuit</code></td>
</tr>
<tr>
<td><code>count</code></td>
<td><code>atleast</code></td>
</tr>
<tr>
<td></td>
<td><code>atmost</code></td>
</tr>
<tr>
<td></td>
<td><code>count</code></td>
</tr>
<tr>
<td></td>
<td><code>exactly</code></td>
</tr>
<tr>
<td><code>cumulative</code></td>
<td><code>cumulative</code></td>
</tr>
<tr>
<td><code>cumulatives</code></td>
<td><code>cumulatives</code></td>
</tr>
<tr>
<td><code>domain</code></td>
<td><code>domain</code></td>
</tr>
<tr>
<td><code>element</code></td>
<td><code>elem</code></td>
</tr>
<tr>
<td></td>
<td><code>element</code></td>
</tr>
<tr>
<td><code>geost</code></td>
<td><code>geost</code></td>
</tr>
<tr>
<td><code>global_cardinality</code></td>
<td><code>global_cardinality</code></td>
</tr>
<tr>
<td></td>
<td><code>global_cardinality_with_costs</code></td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>domain.constraint</code></td>
</tr>
<tr>
<td></td>
<td><code>in</code></td>
</tr>
<tr>
<td></td>
<td><code>in_interval</code></td>
</tr>
<tr>
<td></td>
<td><code>in_intervals</code></td>
</tr>
<tr>
<td><code>in_set</code></td>
<td><code>domain.constraint</code></td>
</tr>
<tr>
<td></td>
<td><code>in</code></td>
</tr>
</tbody>
</table>
### SICStus Catalogue

<table>
<thead>
<tr>
<th>SICStus</th>
<th>Catalog</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lex_chain</code></td>
<td><code>lex_between</code></td>
</tr>
<tr>
<td></td>
<td><code>lex_chain_less</code></td>
</tr>
<tr>
<td></td>
<td><code>lex_chain_le</code></td>
</tr>
<tr>
<td></td>
<td><code>lex_greater</code></td>
</tr>
<tr>
<td></td>
<td><code>lex_greater_eq</code></td>
</tr>
<tr>
<td></td>
<td><code>lex_less</code></td>
</tr>
<tr>
<td></td>
<td><code>lex_less_eq</code></td>
</tr>
<tr>
<td><code>maximum</code></td>
<td><code>maximum</code></td>
</tr>
<tr>
<td><code>minimum</code></td>
<td><code>minimum</code></td>
</tr>
<tr>
<td><code>nvalue</code></td>
<td><code>nvalue</code></td>
</tr>
<tr>
<td><code>relation</code></td>
<td><code>in_relation</code></td>
</tr>
<tr>
<td><code>scalar_product</code></td>
<td><code>scalar_product</code></td>
</tr>
<tr>
<td></td>
<td><code>sum_ctr</code></td>
</tr>
<tr>
<td><code>sorting</code></td>
<td><code>sort</code></td>
</tr>
<tr>
<td></td>
<td><code>sort_permutation</code></td>
</tr>
<tr>
<td><code>table</code></td>
<td><code>in_relation</code></td>
</tr>
</tbody>
</table>
Bibliography


[18] F. W. Barnes. Packing the maximum number of $m \times n$ tiles in a $p \times q$ rectangle. *Discrete Mathematics*, 26:93–100, 1979. 276


[282] P. Ouellet and C.-G. Quimper. The Multi-Inter-Distance Constraint. In 22th Int. Joint Conf. on Artificial Intelligence (IJCAI-11), 2011. 1400, 1401


Over-Constrained Problems. In T. Walsh, editor, Principles and Practice of
Springer-Verlag, 2001. 5, 1726, 1727, 1730, 1731

224

[296] J. Pitrat. MALICE, notre collègue. In Colloque Métacognition de Berder,
pages 4–19, September 2001. In French. iii, 1036

report, Paris VI University, France, 2008. iii, 144, 252, 1036


[299] E. Poder. Programmation par contraintes et ordonnancement de tâches avec
consommation variable de ressource. PhD thesis, Blaise-Pascal - Clermont II
University, France, October 2002. In French. 186

of the Compulsory Part of a Task with Varying Duration and Varying Resource
100, 186

Gent, editor, Principles and Practice of Constraint Programming (CP’2009),
volume 5732 of LNCS, pages 669–683. Springer-Verlag, 2009. 305

Printing. 324

tional Conference on Intelligent Systems (SPICIS), pages 256–261, Singapore,
November 1994. 99

In 15th National Conference on Artificial Intelligence (AAAI-98), pages 359–

International Workshop on Symmetry in Constraint Satisfaction Problems
(SymCon-03), 2003. held along CP-2003. 1827

Beek, editor, Principles and Practice of Constraint Programming (CP’2005),
volume 3709 of LNCS, pages 477–489. Springer-Verlag, 2005. 1827


[403] W. J. van Hoeve and A. Sabharwal. Two Set-Constraints for Modeling and Efficiency. In Sixth International Workshop on Constraint Modelling and Reformulation (ModRef’07), Providence, USA, 2007. 550, 551, 1840


Index
INDEX

1...

Ågren M., iii, 1018, 1276, 1280
Östergård P., 216
3-SAT. 147, 553, 705, 1037, 1468, 1937
3-dimensional-matching. 147, 1223, 1717

A

abs. 420
abs_value, 112, 153, 155, 177, 199, 233, 283, 286, 420, 1000, 1030
absolute_value, 420
abstract interpretation, 147, 1016, 1598
accessibility, 1955
Aggoun A., iii, 4, 99, 186, 242, 317, 594, 786
aggregate. 148
Aho A. V., 1784
air traffic management. 150, 431, 1213, 1401
ALICE, i, 98, 435, 829
alignment, 150, 1529
all different, 151, 439, 442, 444, 447, 451, 454, 458, 463, 467, 1063, 1213, 1499, 1588, 1675, 1679, 1727, 1731, 1855, 1960
all_differ_from_at_least_k_pos, 18, 115, 177, 191, 209, 215, 221, 265, 328, 329, 336, 1401
all_equal, 88, 108, 191, 333, 426, 752, 858, 1000, 1132, 1450, 1468, 1714, 1716, 1722
all_incomparable, 191, 209, 265, 328, 329, 336, 428, 1130
all_incomparables. 428
all_min_dist, 29, 38, 86, 113, 150, 180, 191, 209, 233, 257, 301, 314, 333, 430, 439, 454, 877, 913, 914, 920, 1401
all_null_intersect, 442
all_perm. 474
all_permutations, 474
ALL_VERTICES. 75, 669
alldiff, 434
alldiff_between_sets, 442
alldiff_cst, 446
alldiff_except_0, 450
alldiff_interval, 454
alldiff_modulo, 458
alldiff_on_intersection, 462
alldiff_on_sets, 442
alldiff_on_tuples, 1268
INDEX

alldiff_partition. 466
alldiff_same_value. 470
alldifferent_consecutive_values. 108, 151, 215, 279, 314, 333, 437, 439, 444, 752, 1588
alldifferent_cst. 110, 144, 151, 153, 178, 179, 191, 198, 215, 266, 268, 269, 271, 314, 333, 439, 446
alldifferent_interval. 113, 151, 153, 168, 171, 191, 245, 271, 314, 333, 431, 439, 450
alldifferent_modulo. 113, 151, 153, 168, 171, 191, 262, 271, 314, 433, 439, 458
alldifferent_on_multisets. 100
alldifferent_on_sets. 442
alldifferent_on_tuples. 1268
alldifferent_partition. 120, 151, 153, 191, 245, 271, 277, 314, 333, 439, 466, 1124
alldifferent_same_value. 56, 63, 127, 168, 171, 233, 285, 314, 439, 470
alldistinct. 434
alldistinct_between_sets. 442
alldistinct_cst. 446
alldistinct_except_0. 450
alldistinct_interval. 454
alldistinct_modulo. 458
alldistinct_on_intersection. 462
alldistinct_on_sets. 442
alldistinct_on_tuples. 1268
alldistinct_partition. 466
alldistinct_same_value. 470
allperm. 109, 148, 178, 191, 249, 256, 274, 314, 328, 329, 332, 474, 474, 1267, 1281, 1311, 1311, 1316, 1815
alpha-acyclic constraint network(2). 151, 479, 486, 491, 495, 499, 535, 547, 763, 767, 868, 1013, 1079, 1090, 1683
alpha-acyclic constraint network(3). 152, 1079, 1090, 1206
Althaus E., iii, 662, 1216, 1570, 1824, 1874
Alvarez-Valdes R., 277
Amazon, see n-Amazon
Amil hastre J., ii
INDEX

assignment dimension removed, 84
assignment to the same set of values, 163, 596, 601, 820, 877, 950, 959, 1022, 1029
at least, 168, 535, 621, 1506
at most, 168, 547, 625, 629, 1401, 1508
atleast, 14, 15, 31, 86, 125, 151, 153, 168, 169, 171, 221–223, 333, 450, 479, 486, 527, 534, 546, 1012, 1168, 1206, 1383, 1480, 1506, 1584, 1619, 1683, 1959
atleast_nvalue, 113, 153, 178, 203, 220, 221, 270, 322, 334, 510, 538, 552, 553, 1402, 1446, 1450, 1468, 1524, 1675, 1679
atleast_nvector, 115, 203, 217, 220, 221, 270, 322, 334, 336, 542, 557, 1288, 1485, 1532, 1533, 1870, 1886
atmost, 32, 86, 125, 151, 153, 168, 169, 171, 191, 333, 479, 527, 534, 535, 546, 789, 835, 1012, 1508, 1619
atmost1, 110, 180, 188, 191, 283, 550
atmost_nvalue, 113, 147, 180, 191, 203, 220, 270, 322, 334, 538, 539, 552, 1468, 1714, 1716, 1722, 1727, 1731
atmost_nvector, 115, 191, 203, 217, 220, 270, 322, 334, 336, 543, 556, 1288, 1485, 1536, 1537
atmost_sliding, 1400
atour, 662, 1874
attached to cost variant, 85
automaton with array of counters, 171, 439, 451, 454, 458, 463, 471, 527, 561, 571, 575, 595, 621, 625, 789, 899, 1037, 1180, 1185, 1189, 1339, 1365, 1369, 1468, 1625, 1919
automaton with counters, 171, 479, 486, 491, 495, 499, 523, 535, 547, 633, 641, 651, 661, 672, 763, 767, 849, 853, 863, 868, 927, 1013, 1079, 1090, 1101, 1161, 1206, 1255, 1259, 1323, 1579, 1683, 1709, 1941
automaton without counters, 172, 510, 515, 518, 590, 677, 681, 723, 749, 858, 941, 950, 955, 959, 963, 967, 971, 979, 983, 1010, 1058, 1104, 1107, 1110, 1125, 1132, 1137, 1143, 1169, 1173, 1273, 1285, 1289, 1293, 1299, 1305, 1312, 1349, 1379, 1383, 1387, 1403, 1418, 1433, 1437, 1447, 1451, 1454, 1458, 1518, 1520, 1525, 1575, 1660, 1793, 1804, 1812, 1816, 1820, 1903, 1911, 1941, 1962
autoref, 173, 1037

B

Bacchus F., 297, 478
balance_cycle, 116, 184, 187, 208, 210, 233, 237, 238, 271, 279, 322, 561, 566, 830
balance_interval, 125, 159, 169, 171, 173, 220, 233, 245, 286, 333, 560, 570
balance_modulo, 125, 159, 169, 171, 173, 220, 233, 262, 286, 333, 560, 574
balance_partition, 127, 159, 173, 220, 233, 277, 286, 333, 560, 578, 1124
balance_path, 116, 187, 210, 233, 237, 238, 271, 277, 332, 561, 582, 1567
balanced assignment, 173, 561, 571, 575, 579, 1349
balanced tree, 174, 1890
Baptiste P., 430, 786, 912
Barichard V., 434
Barnes F. W., 276
Barnier N., 1208
Bartak R., 273
Baues G., iii
Becket R., 404
Beeri C., 175
Berger C., 5, 52, 57, 434
Berge-acyclic constraint network, 174, 479, 511, 515, 518, 633, 661, 677, 681, 723, 727, 731, 735, 739, 749, 983, 1010, 1058, 1104, 1111, 1137, 1143, 1169, 1173, 1273, 1285, 1289, 1293, 1299, 1305, 1312, 1403, 1447, 1525, 1575, 1709, 1804, 1812, 1903, 1911, 1962
Berliner H. J., 662
Berthier F., 99
Bessi`ere C., iii, 5, 100, 154, 181, 376, 434, 478, 538, 552, 704, 1040, 1208, 1466, 1618, 1726, 1730, 1936
between, 478, 1272
between_min_max, 113, 169, 172, 182, 221, 290, 590, 797, 799, 1107
bin_packing_capa, 121, 159, 160, 163, 192, 283, 295, 595, 600, 1157
bin_packing_load, 601
binary constraint, 177, 420, 930, 932, 963, 967, 979, 1000, 1002, 1004, 1030, 1032, 1098, 1116, 1125, 1262, 1264, 1326, 1410, 1412, 1522, 1616, 1656, 1672, 1793, 1848
binary_tree, 116, 187, 233, 237, 238, 271, 332, 602, 1566–1568, 1886
Bininda-Emonds O. R., 1784
bioinformatics, 177, 423, 1660, 1788
bipartite, 110, 178, 188, 210, 237, 328, 606, 1128, 1874
bipartite matching, 178, 439, 442, 447, 539, 899, 1269
bipartite matching in convex bipartite graphs, 179, 439, 447
INDEX

Bleuzen-Guernalec N., 434, 1772
Bockmayr A., 662, 1570, 1824, 1874
Bohlin M., 4
Boolean channel, 179, 941
Boolean constraint, 180, 510, 511, 676, 677, 680, 681, 1010, 1104, 1402, 1403, 1446, 1447, 1524, 1525, 1962
Bordeaux L., 5
border, 180, 1583
bound(D) consistency, 181
bound(Z) consistency, 181
bound-consistency, 180, 431, 439, 551, 553, 1037, 1042, 1155, 1213, 1401, 1468, 1625, 1636, 1691, 1714, 1717, 1773, 1840, 1843, 1919
bound_alldiff, 434
bound_alldifferent, 434
bound_distinct, 434
Bourdais S., 78, 1034, 1120, 1574
Bourreau É., iii, 4, 181, 310, 317, 828, 834, 842
Bouwkamp C. J., 317
Brand S., iii, 404, 502, 1842
Brisset P., iii, 1208
Bron C., 257, 430, 434, 684, 912
business rules, 181, 830, 877, 1021

calendar, 144, 160, 184, 192, 263, 283, 301–303, 330, 610, 789, 820, 877, 914, 1021
Cambazard H., iii, 594, 1848
Caprara A., 310
card_matrix, 688
card_set, 50
card_var gcc, 1034, 1035
cardinality_atmost_partition, 127, 148, 153, 168, 178, 233, 265, 277, 286, 333, 628, 1037, 1106
cardinality_matrix, 688, 689
cardinality_on_attributes_values, 1466
Carillon J.-P., 78, 958
Carlier J., 332, 912
Carlson B., 99, 103
Carlsson M., i, ii, 2, 4, 5, 78, 99, 100, 103, 172, 182, 203, 242, 252, 279, 292, 305, 317, 324, 326, 404, 502, 656, 786, 818, 872, 898, 1018, 1024, 1268, 1272, 1276, 1280, 1292, 1298, 1304, 1310, 1406, 1414, 1424, 1428, 1466, 1612, 1690, 1844
Carrauilla M. A., 872
case, 100, 189, 216, 233, 305, 315, 612, 959, 1120
Caseau Y., 100, 786
INDEX

Cayley A., 1604
CC, 75, 1670, 1684
centered cyclic(1) constraint network(1), 182, 590, 941, 1107, 1349, 1379, 1383, 1454, 1518, 1520
centered cyclic(2) constraint network(1), 182, 950, 959, 963, 967, 979, 1125, 1387, 1793
centered cyclic(3) constraint network(1), 183, 971, 1418
Chabert G., iii, 306, 1484
CHAIN, 53, 1091
Chan P., iii, 310
change_partition, 127, 148, 178, 233, 266, 270, 277, 286, 331, 633, 656, 1124
channel, 1188, 1189, 1194, 1195
channel routing, 183, 744
channelling constraint, 184, 619, 941, 1190, 1195, 1199, 1203, 1318, 1319, 1625
Charlier P., iii
Charman P., 224
CHARME, 99, 1034
Cheadle A. M., 99
Cheng K. C.K., 100
Cheng K.-H., 872
CHIP, 99, 202, 446, 479, 534, 546, 632, 662, 787, 829, 834, 835, 842, 843, 882, 886, 958, 970, 978, 1076, 1078, 1188, 1189, 1292, 1298, 1304, 1310, 1348, 1378, 1450, 1612, 1690, 1735, 1869
Choco, 2, 99, 99, 103, 144, 959, 1792
choquet, 100, 344
Chu G., 5
INDEX

Chvátal V., 662
CIRCUIT, 53, 673, 1801
circuit, 184, 567, 663, 825, 831, 1855
circuit, 39, 60, 62, 87, 110, 184, 210, 237–239, 250, 271, 279, 282, 321, 439, 662, 667, 829, 830, 1567, 1824, 1874
circuit_cluster, 117, 185, 237, 271, 279, 322, 437, 439, 663, 666, 830, 1476, 1477
circular sliding cyclic(1) constraint network(2), 184, 672
circular_change, 53, 123, 169, 171, 184, 208, 233, 270, 286, 331, 633, 672
clause, 676, 680, 1402, 1446
clause_and, 127, 153, 169, 172, 174, 180, 221, 290, 510, 676, 680
clause_or, 127, 153, 169, 172, 174, 180, 216, 221, 290, 676, 680, 1524
Clautiaux F., iii, 332
CLIQUE(<), 432, 476, 784, 884, 889, 915, 917, 919, 1069, 1220, 1270, 1531, 1662, 1728, 1771, 1900
CLIQUE(≠), 424, 429, 686, 879, 896, 924, 1560, 1596, 1606, 1718, 1856, 1875
clique, 116, 188, 233, 237, 257, 328, 344, 684, 1128, 1318
CLIQUE(Comparison), 54
cluster, 185, 667
Coffman E. G., 594
Cohn A. G., 288, 754, 770, 776, 902, 1006, 1164, 1354, 1562
Coletta R., 5
collection generator, 41, 42, 42, 52
Collet R., 934
Colmerauer A., 434, 1772
colored_cumulative, 692
colored_cumulatives, 698
colored_matrix, 88, 90, 136, 233, 256, 283, 286, 329, 331, 688, 1037, 1212, 1213, 1625
coloured, 185, 527, 694, 700, 835, 1180
coloured_cumulative, 83, 117, 185, 192, 270, 295, 301, 330, 342, 344, 692, 700, 789, 909, 1467, 1468, 1476, 1477, 1880
coloured_cumulatives, 121, 160, 162, 163, 185, 192, 270, 295, 301, 330, 342, 344, 694, 698, 789, 820, 1467, 1468, 1477
coloured_matrix, 658
combinatorial object, 139, 439, 444, 503, 523, 567, 583, 641, 663, 667, 759, 830, 835, 863, 866, 877, 936, 993, 1021, 1058, 1079, 1090, 1101, 1121, 1161, 1189, 1213, 1223, 1227, 1231, 1236, 1239, 1255, 1259, 1398, 1433, 1437, 1494, 1496, 1567, 1571, 1579, 1583, 1585, 1587, 1588, 1595, 1613, 1625, 1631, 1636, 1645, 1649, 1653, 1660, 1675, 1679, 1683, 1687, 1691, 1773, 1780, 1804, 1812, 1855, 1861, 1865, 1870, 1919, 1941
common keyword, 86
common_interval, 135, 148, 178, 188, 233, 245, 266, 286, 705, 708
common_modulo, 135, 148, 178, 188, 233, 262, 266, 286, 705, 712
common_partition, 135, 148, 178, 188, 234, 266, 277, 286, 705, 716, 1124
compare_and_count, 192, 221, 283, 720, 763
comparison swapped, 86
complexity, 139, 553, 705, 789, 820, 877, 895, 914, 1037, 1213, 1223, 1468, 1486, 1717, 1937, 1960
compulsory part, 185, 694, 700, 789, 798, 804, 809, 820, 877, 914
cond_lex_cost, 36, 127, 153, 169, 172, 174, 202, 249, 274, 284, 290, 336, 722, 727, 731, 735, 739, 959, 1121
cond_lex_greater, 129, 153, 169, 174, 249, 274, 284, 336, 723, 726, 731, 735, 739, 1121, 1293
cond_lex_greatereq, 129, 153, 169, 174, 249, 274, 284, 336, 723, 727, 730, 735, 739, 1121, 1299
cond_lex_less, 129, 153, 169, 174, 249, 274, 284, 336, 723, 727, 733, 734, 739, 1121, 1305
cond_lex_lesseq, 129, 153, 169, 174, 249, 274, 284, 336, 723, 727, 731, 735, 738, 1121, 1311
conditional constraint, 186, 1675, 1679
Condon A. E., 422, 868
configuration, 186, 273
configuration problem, 186, 975
connect_points, 54, 135, 183, 234, 236, 246, 322, 328, 742
connected, 110, 187, 188, 237, 328, 746, 1128, 1824, 1852
connected component, 187, 463, 567, 583, 587, 603, 641, 746, 831, 835, 843, 1058, 1079, 1216, 1329, 1473, 1567, 1605, 1870, 1886, 1890, 1893
Connors D. P., 1208
consecutive loops are connected, 187, 1079, 1804, 1812
consecutive values, 187, 1344, 1345, 1374, 1375, 1462, 1463
consecutive_groups_of_ones, 22, 106, 119, 144, 153, 169, 172, 174, 252, 253, 290, 748, 1079
CONSECUTIVE_LOOPS_ARE_CONNECTED, 69
consecutive_values, 108, 283, 314, 333, 426, 444, 752, 1058, 1468
constant_sum, 1834
constraint between three collections of variables, 188, 759, 1780
constraint between two collections of variables, 188, 463, 705, 709, 713, 718, 1625, 1631, 1636, 1641, 1645, 1649, 1653, 1739, 1743, 1748, 1751, 1755, 1759, 1763, 1767, 1773, 1919, 1925, 1929, 1933, 1937
constraint involving set variables, 188, 442, 551, 607, 685, 746, 856, 895, 936, 1004, 1073, 1128, 1199, 1216, 1318, 1319, 1499, 1503, 1506, 1508, 1511, 1515, 1570, 1571, 1605, 1619, 1667, 1824, 1827, 1840, 1848, 1852, 1861, 1865, 1874
INDEX


constraint on the intersection, 189, 463, 705, 1473, 1641


constructive disjunction, 189, 877, 914, 1911

contact, 190, 1539, 1903

container, 161, 162, 1027

contains, 754

contains_box, 192, 236, 252, 288, 754, 771, 777, 903, 1008, 1166, 1356, 1441, 1564

Costejean E., iii, 78, 478, 494, 502, 828, 866, 872, 1854, 1868

contiguity, 1058

contractibility, 273

contractible, 191

convex, 197, 798, 1058

convex bipartite graph, 198, 439, 447, 1468

convex hull relaxation, 198, 1831

Conway packing problem, 199, 878, 1022

core, 199, 439, 789, 830, 877, 914, 959, 1037, 1395, 1468, 1773

Cormen T. H., 782

Corn M. R., 422, 868

Cornelissen T., 284

correspondence, 129, 148, 178, 188, 214, 266, 279, 437, 758, 1625, 1780
INDEX

cost filtering constraint, 202, 723, 1055, 1395, 1845, 1960
cost matrix, 202, 1055, 1395
cost variant, 86
cost_gcc, 1052
cost_ordered_global_cardinality, 1541
cost_regular, 100
Costa M.-C., 434
Costas arrays, 202, 439
Coté M.-C., ii, 100, 250
count, 107, 131, 148, 151, 153, 169, 171, 192, 203, 221, 333, 478, 479, 514, 527, 534, 546, 720, 762, 766, 767, 1012, 1034, 1036, 1338, 1339, 1368, 1369, 1468, 1476, 1490, 1618, 1619
counting constraint, 203, 479, 480, 486, 491, 495, 499, 507, 539, 543, 553, 557, 720, 763, 767, 890, 891, 1013, 1036, 1037, 1041, 1042, 1143, 1149, 1255, 1259, 1339, 1369, 1407, 1415, 1429, 1459, 1468, 1473, 1477, 1481, 1486, 1491, 1503, 1511, 1515, 1533, 1537, 1545, 1619, 1683
counts, 131, 148, 151, 153, 169, 171, 178, 192, 203, 221, 266, 333, 479, 527, 528, 763, 766
Cousin X., 1178
Cousot P., 147
Cousot R., 147
coveredby, 770
coveredby_sboxes, 236, 252, 288, 756, 770, 777, 903, 1008, 1166, 1356, 1441, 1564
covers, 776
covers_sboxes, 192, 236, 252, 288, 756, 771, 776, 903, 1008, 1166, 1356, 1441, 1564
Croft H. T., 315
crossable unavailability period, 301
crossing, 23, 50, 117, 148, 234, 236, 252, 266, 286, 782, 1066, 1067, 1899
Cui Z., 288, 754, 770, 776, 902, 1006, 1164, 1354, 1562
cumulative longest hole problems, 203, 789
cumulative_convex, 48, 117, 185, 192, 197, 278, 295, 301, 330, 437, 590, 789, 794, 1835
cumulative_max, 786
cumulative_product, 117, 185, 192, 285, 295, 301, 330, 342, 789, 802, 1602
cumulative_trapeze, 100, 186
cumulative_two, 117, 185, 192, 214, 236, 283, 287, 344, 595, 789, 808, 809, 877
cumulative_with_level_of_priority, 121, 192, 214, 295, 301, 330, 342, 789, 812, 1835
cumultives, 11, 39, 47, 84, 124, 144, 160, 162–164, 167, 185, 192, 210, 214, 284, 295, 299, 301, 324, 330, 331, 337, 342, 344, 619, 698, 700, 789, 818, 877, 1835
Cunningham S., 4
cutset, 65, 117, 148, 184, 209, 215, 237, 262, 266, 285, 344, 824, 1128
CYCLE, 54
cycle, 208, 567, 831, 1855
cycle_card_on_path, 136, 185, 187, 237, 238, 271, 279, 297, 308, 495, 830, 834
cycle_or_accessibility, 126, 224, 234, 236, 237, 322, 830, 838, 1481
cycle_resource, 122, 187, 214, 237, 238, 295, 322, 345, 830, 842

cyclic, 208, 672, 849, 853, 1800


Cymer R., iii

D

dag, 9, 110, 188, 237, 856, 1128
data constraint, 208, 950, 955, 959, 963, 967, 971, 975, 979, 983, 987, 993, 997, 1121, 1206, 1418, 1425, 1793, 1831

Daubern P., 4
deadlock breaking, 209, 825
debruyne R., iii
dece F., iii

dechter R., 39, 152, 239
decomposition, 209, 423, 428, 431, 503, 515, 518, 523, 858, 877, 883, 888, 895, 914, 916, 918, 941, 1021, 1027, 1132, 1213, 1219, 1223, 1227, 1231, 1236, 1239, 1243, 1247, 1251, 1269, 1277, 1281, 1319, 1549, 1600, 1619, 1660, 1687, 1691, 1816, 1820, 1861, 1865, 1955
decomposition-based violation measure, 210, 426, 439, 1717, 1727
deepest_valley, 113, 169, 171, 257, 297, 307, 862, 1101, 1941
degree of diversity of a set of solutions, 211, 1277, 1727
demand profile, 210, 820, 1631, 1636
deemassey S., iii, iv, 100, 786
denley T. M. J., 434
denmat T., 1016, 1598
deo N., 662
derangement, 110, 153, 237, 271, 279, 314, 439, 830, 866
derived collection, 42, 42–45, 52, 74, 75, 77, 214, 214, 402, 527, 759, 809, 814, 820, 843, 941, 959, 971, 979, 997, 1063, 1107, 1110, 1121, 1125, 1293, 1299, 1305, 1312, 1319, 1387, 1418, 1425, 1454, 1699, 1780, 1880, 1893, 1899
deviation, 100, 173, 320
deville Y., 100, 173, 174, 189, 326, 894, 934, 1826

dFS-bottleneck, 210, 439, 567, 583, 607, 663, 831, 1037, 1042, 1567, 1625, 1919

di Battista G., 1898

diedrich F., 305
diff, 1284
diff2, 872, 873
differ_from_at_least_k_pos, 55, 127, 151, 169, 171, 221, 329, 333, 336, 423, 424, 868
difference, 214, 1063, 1843
difference between pairs of variables, 214, 1269
different, 1284
diffn, 9, 14, 52, 90, 107, 109, 144, 156, 160–164, 181, 185, 186, 189, 192, 199, 202, 209, 216,
224, 225, 236, 241, 246, 263, 269, 275–277, 279, 282, 287, 293, 294, 296–302, 309,
310, 312, 315, 320, 324, 326, 331, 332, 402, 431, 439, 619, 788, 789, 808, 809, 820,
872, 877, 882, 883, 886, 888, 914, 1018, 1021, 1024, 1027, 1213, 1269, 1277, 1281,
diffn_column, 115, 192, 209, 236, 238, 275, 282, 882, 888, 1906, 1907
diffn_include, 115, 192, 209, 236, 275, 282, 887, 883, 886, 1914, 1915
diffst, 1954, 1955
Dinchbas M., iii, 99, 1208
directed acyclic graph, 215, 825
disequality, 215, 423, 439, 442, 444, 447, 899, 993, 1063, 1213, 1219, 1285, 1412, 1451, 1454,
1499, 1588, 1619, 1675, 1679, 1727, 1731, 1855
disinequality, 437
disj, 110, 188, 209, 295, 299, 301, 894, 914, 1128
1218–1220, 1284
disjoint1, 872, 873
disjoint2, 243, 872, 873
disjoint_boxes, 192, 236, 252, 288, 756, 771, 777, 902, 1008, 1166, 1356, 1441, 1564
disjoint_tasks, 122, 192, 269, 301, 330, 346, 694, 899, 908
disjunction, 216, 518, 681, 877, 914, 916, 918, 950, 959, 1022, 1029, 1525
disjunctive, 110, 185, 189, 192, 199, 202, 209, 216, 250, 257, 280, 295, 298, 299, 301, 314,
342, 431, 439, 619, 788, 789, 877, 895, 912, 916, 918, 1600
disjunctive_or_same_end, 110, 192, 209, 216, 295, 301, 342, 914, 916, 918
disjunctive_or_same_start, 110, 192, 209, 216, 295, 301, 342, 914, 916, 918
DISTANCE. 69, 924, 928
distance, 123, 155, 234, 283, 286, 330, 431, 920, 922, 926, 1264, 1709
distance_between, 70, 131, 234, 285, 286, 922, 927
distance_change, 131, 169, 171, 234, 285, 286, 308, 633, 923, 926
distinct, 11, 434
distribute, 1034
distribution, 1034, 1035
div, 930
div_or, 932
divisible, 112, 153, 155, 177, 283, 930, 932, 1656
divisible_or, 112, 155, 177, 283, 930, 932
dom, 938, 1106, 1106, 1110, 1128
dom_reachability, iii, 134, 188, 237, 283, 934, 1567, 1570
dom_reified, 1114
domain, 125, 192, 216, 245, 283, 333, 938, 940, 1107, 1110, 1890
domain_channel, 216, 941
domain consistency, 154
domain definition, 216, 515, 939, 1107, 1110, 1111, 1118, 1454
domain_constraint, 45, 116, 153, 169, 172, 179, 182, 184, 209, 214, 216, 250, 290, 940,
1318, 1618, 1619, 1862, 1866
dominating queens, 216, 1469
domination, 217, 543, 557, 1469, 1477, 1486, 1491, 1845
Dooms G., iii, 7, 9, 99, 100, 606, 746, 856, 1852, 1884
double counting, 1036
INDEX

double_lEx, 1266
Dowsland K. A., 277
dual model, 217, 1190, 1195, 1199, 1203
Ducassé M., 1016, 1598
Duck G. J., 404
Dudeney H. E., 5
Duijvestijn A. J. W., 315, 317
duplicated variables, 218, 1037, 1213, 1293, 1300, 1305, 1312, 1659, 1800
Dupont P., 100, 173, 174, 894, 1826
dynamic graph constraint
 assign_and_counts, 528
 assign_and_nvalues, 532
 bin_packing, 597
 circuit_cluster, 669
 coloured_cumulative, 695
coloured_cumulatives, 701
cumulative, 790
cumulative_convex, 799
cumulative_product, 805
cumulative_with_level_of_priority, 815
cumulatives, 821
cycle_card_on_path, 836
cycle_or_accessibility, 840
indexed_sum, 1158
interval_and_count, 1181
interval_and_sum, 1186
minimum_greater_than, 1388
next_element, 1419
next_greater_element, 1426
shift, 1670
sliding_card_skip0, 1684
sliding_time_window, 1696
sliding_time_window_sum, 1705
track, 1881
dynamic programming, 218, 633, 789, 1709, 1800, 1804
dynamic wavelength routing, 249

d dynam ing programming, 218, 633, 789, 1709, 1800, 1804
dynamic wavelength routing, 249

dynam  

E

Eades P., 1898
ECLAIR, 99
ECLiPSe, 99, 404, 406
egcc, 1034, 1035
Elbassioni K. M., 1208, 1222, 1238
elem_from_to, 121, 153, 155, 169, 172, 208, 290, 330, 335, 950, 954, 959
elem_matrix, 970
element_from_to, 954
element_greatereq, 121, 153, 155, 169, 172, 177, 182, 208, 250, 291, 330, 335, 950, 959, 962, 967, 974
element_lessseq, 121, 153, 155, 169, 172, 177, 182, 208, 250, 291, 330, 335, 950, 959, 963, 966, 974, 1027
element_matrix, 12, 136, 153, 155, 169, 172, 183, 208, 214, 256, 291, 330, 950, 959, 970
element_product, 132, 155, 186, 208, 221, 234, 286, 330, 335, 950, 959, 963, 967, 974
element_sparse, 128, 153, 155, 169, 172, 177, 182, 208, 214, 291, 315, 329, 330, 335, 950, 959, 978, 996, 997
element_var, 958
elementm, 127, 153, 169, 172, 174, 208, 221, 291, 308, 330, 950, 982
elements, 121, 153, 208, 234, 286, 287, 305, 329, 330, 950, 959, 986, 990, 992
elements_alldiff, 990
elements_alldifferent, 121, 155, 208, 215, 234, 279, 330, 439, 950, 959, 987, 990
elements_alldistinct, 990
elements_sparse, 128, 153, 208, 214, 305, 315, 329, 330, 950, 959, 979, 996
Elf M., 662, 1570, 1824, 1874
empty intersection, 218, 899, 1219
entailment, 219, 439, 495, 1042, 1349, 1379, 1454
Eppstein D., 434
eq, 112, 153, 155, 177, 234, 283, 286, 420, 426, 1000, 1002, 1004, 1010, 1030, 1098, 1262, 1288, 1326, 1410, 1656
eq.set, 19, 123, 153, 155, 177, 234, 283, 286, 1000, 1002, 1032, 1264, 1412
equation, 1658
EQUIVALENCE, 69, 540, 544, 554, 558, 563, 572, 576, 580, 1144, 1470, 1478, 1488, 1492
equivalence, 220, 539, 543, 553, 557, 561, 571, 575, 579, 1143, 1339, 1369, 1407, 1415, 1429, 1451, 1459, 1469, 1477, 1486, 1491, 1731
Eran H., 1208
Erschler J., 786
Euler knight, 220, 439, 831
Euler L., 5
exactly, 125, 148, 151, 153, 169–171, 203, 234, 286, 333, 479, 535, 546, 1012
excluded, 221, 1454
extended_global_cardinality, 1034
extended_sortedsness, 1778
Extensible, 221
INDEX

extension, 224, 1121
extension, 1120
extensional, 1120
extensional support, 1120
extensional support Madd, 1120
extensional support Str, 1120
extensional support Va, 1120

F

FaCile, 99
facilities location problem, 224, 839, 1845
Fages F., iii, 824
Fages J.-G., iii, 1884
Fagin R., 152, 175
Fahle T., 684
Falconer K. J., 315
Falkenhainer B., 1678
Falkowski B.-J., 269
Faltings B., 273
Favaron O., 376
feastuple ac, 1120
Fekete S. P., 297
Fekete S. P., 297
Feldman J., 4
Festa P., 262

Fink A., 278, 794
Flajolet O., 1328
INDEX

Flamm C., 1660
Flener P., iii, iv, 292, 474, 982, 1266, 1316, 1784, 1814, 1884
floor planning problem, 224, 877, 1021, 1277
flow, 227, 439, 503, 1037, 1042, 1045, 1049, 1499, 1511, 1515, 1625, 1631, 1636, 1691, 1861, 1865, 1919
Focacci F., 890, 1394
forced_shift_stretch, 1803, 1804
Ford Jr. L. R., 688
Fotso L. P., 786
frequency allocation problem, 233, 431
Freuder E. C., 239
Freund A., 1208
Friedman E., 276, 309, 312
Frisch A. M., 78, 474, 1266, 1292, 1298, 1304, 1310, 1316, 1814
Frutos A. G., 422, 868
Frühwirth T., 1292, 1298, 1304, 1310
Fulkerson D. R., 688

G

Galinier P., 78, 1034, 1120, 1574
Gambini I., 872
Gandibleux X., iii
Gansner E. R., 104, 405
García de la Banda M. J., 5, 404
Garey M. R., 430, 594, 934, 1898
gcc, 201, 1034, 1040
gcc_low_up, 228, 1040
gcc_low_up_no_loop, 228, 1044
gcc_no_loop, 1048
gcc, 1052
gcd, 123, 147, 155, 234, 283, 286, 330, 1016, 1598
Gecode, ii, 2, 99, 99, 103, 404, 676, 680, 895, 936, 938, 950, 958, 1106, 1110, 1128, 1189, 1194, 1195, 1659
Gehring H., 872
Gendron B., ii, 100, 250
generalisation, 86
generalized arc-consistency, 154
INDEX

Gent I. P., 99, 296, 434, 1784
gometry, 141, 744, 756, 772, 778, 810, 839, 877, 883, 888, 904, 909, 1008, 1021, 1029, 1067, 1166, 1356, 1442, 1529, 1549, 1552, 1555, 1559, 1564, 1591, 1595, 1660, 1899, 1903, 1907, 1911, 1915, 1955
Georget Y., iii
Georgiadis L., 934
gost_time, 134, 156, 160, 162, 163, 209, 216, 236, 252, 269, 283, 324, 331, 877, 1021, 1024, 1442, 1955
geq, 112, 153, 155, 177, 283, 420, 1000, 1030, 1032, 1098, 1262, 1298, 1326, 1410, 1672
geq_cast, 123, 153, 155, 177, 283, 1002, 1030, 1032, 1264
Gervet C., 550
Ginsberg M. L., 250, 890
Gittleman J. L., 1784
global_cardinality_closed, 1036
global_cardinality_low_up_closing, 1041
global_cardinality_low_up_no_loop, 133, 227–230, 333, 1042, 1044, 1049, 1635, 1886
global_cardinality_no_loop, 89, 127, 227, 234, 286, 333, 1036, 1045, 1048, 1886
Glover F., 179, 434
golomb, 47, 108, 151, 192, 214, 215, 237, 439, 1062, 1820
Golomb ruler, 237, 439, 1063
Golomb S. W., 310, 872, 1062, 1594
Golynski A., 5, 1034
Gomes C., 688
Gondran M., 57, 185
Gotlieb A., 1016, 1598
Grabisch M., 100, 344
Graf T., 99
Grandcolas S., 297
graph colouring, 237, 439, 1175, 1213
graph constraint, 237, 567, 583, 587, 603, 607, 663, 667, 685, 746, 825, 830, 831, 835, 839, 843, 856, 866, 936, 1067, 1073, 1189, 1195, 1203, 1216, 1329, 1567, 1571, 1605, 1788, 1824, 1827, 1852, 1855, 1870, 1874, 1886, 1890, 1893
Graph invariants:

- MAX, NCC, 354
- MAX, NSCC, 354
- MIN, NCC, 354
- MIN, NSCC, 354
- NARC, 354
- NCC, 355
- NSCC, 355
- NSINK, 355
- NSOURCE, 355
- NVERTEX, 355

- MAX, NCC, MAX, NSCC, 356
- MAX, NCC, MIN, NCC, 356
- MAX, NCC1, MIN, NCC1, 386
- MAX, NCC2, MIN, NCC2, 386
- MAX, NCC, NARC, 356
- MAX, NCC1, NCC2, 386
- MAX, NCC2, NCC1, 386
- MAX, NCC, NSINK, 357
- MAX, NCC, NSOURCE, 357
- MAX, NCC, NVERTEX, 357
- MAX, NSCC, MIN, NSCC, 357
- MAX, NSCC, NARC, 358
- MAX, NSCC, NVERTEX, 358
- MIN, NCC, MIN, NSCC, 358
- MIN, NCC, NARC, 359
- MIN, NCC, NCC, 359
- MIN, NCC1, NCC2, 387
- MIN, NCC2, NCC1, 387
- MIN, NCC, NVERTEX, 359
- MIN, NSCC, NARC, 360
- MIN, NSCC, NVERTEX, 360

- NARC, NARC, 387
- NARC, NCC, 360
- NARC, NSCC, 360
- NARC, NSINK, 361
- NARC, NSOURCE, 361
- NARC, NVERTEX, 361
- NCC1, NCC2, 387
- NCC, NSCC, 362
- NCC, NVERTEX, 362
- NSCC, NSINK, 363
- NSCC, NSOURCE, 363
- NSCC, NVERTEX, 363
- NSINK, NVERTEX, 364
- NSOURCE, NVERTEX, 364
- NVERTEX1, NVERTEX2, 387
- MAX, NCC1, MIN, NCC1, MIN, NCC2, 388
- MAX, NCC2, MIN, NCC2, MIN, NCC1, 389
- MAX, NCC, MIN, NCC, NARC, 365
INDEX

MAX_NCC, MIN_NCC, NCC, 365
MAX_NCC, MIN_NCC, N_VERTEX, 365
MAX_NCC1, MIN_NCC1, N_VERTEX2, 389
MAX_NCC2, MIN_NCC2, N_VERTEX1, 389
MAX_NCC, NARC, NCC, 366
MAX_NCC, NARC, N_VERTEX, 367
MAX_NCC, NCC, NSINK, 368
MAX_NCC, NCC, NSOURCE, 368
MAX_NSSC, MIN_NSSC, NARC, 369
MAX_NSSC, MIN_NSSC, NSCC, 369
MAX_NSSC, MIN_NSSC, N_VERTEX, 369
MAX_NSSC, NSCC, N_VERTEX, 370
MIN_NCC1, NARC2, NCC1, 390
MIN_NCC, NARC, N_VERTEX, 370
MIN_NCC, NCC, N_VERTEX, 372
MIN_NSSC, NARC, N_VERTEX, 372
MIN_NSSC, NCC, N_VERTEX, 372
MIN_NSSC, NSCC, N_VERTEX, 372
NARC, NCC, N_VERTEX, 373
NARC, NSSC, N_VERTEX, 374
NARC, NSINK, N_VERTEX, 376
NARC, NSOURCE, N_VERTEX, 377
NSCC, NSINK, NSOURCE, 377
NSINK, NSOURCE, N_VERTEX, 378
MAX_NCC1, MIN_NCC1, MIN_NCC2, NCC1, 391
MAX_NCC2, MIN_NCC2, MIN_NCC1, NCC2, 391
MAX_NCC1, MIN_NCC1, MIN_NCC2, N_VERTEX2, 392
MAX_NCC2, MIN_NCC2, MIN_NCC1, N_VERTEX1, 392
MAX_NCC, MIN_NCC, NARC, NCC, 379
MAX_NCC, MIN_NCC, NCC, N_VERTEX, 380
MAX_NCC, NARC, NSOURCE, N_VERTEX, 380
MAX_NSSC, MIN_NSSC, NARC, NSCC, 380
MAX_NSSC, MIN_NSSC, NSCC, N_VERTEX, 381
MIN_NCC, NARC, NCC, N_VERTEX, 381
NARC, NCC, NSCC, N_VERTEX, 382
NARC, NSINK, NSOURCE, N_VERTEX, 384
MAX_NCC1, MAX_NCC2, MIN_NCC1, MIN_NCC2, NCC1, 393
MAX_NCC1, MAX_NCC2, MIN_NCC1, MIN_NCC2, NCC2, 396
MAX_NCC, MIN_NCC, NARC, NCC, N_VERTEX, 385
MIN_NCC, NARC, NCC, NSCC, N_VERTEX, 385
MAX_NCC1, MAX_NCC2, MIN_NCC1, MIN_NCC2, NCC1, NCC2, 398

greater, 1292
greatereq, 1298

graph partitioning constraint, 238, 567, 583, 587, 603, 663, 830, 831, 835, 843, 1067, 1329, 1567, 1855, 1870, 1874, 1886, 1890, 1893

graph_crossing, 117, 234, 236–238, 252, 286, 783, 830, 1066, 1329, 1886, 1899

graph_isomorphism, iii, 129, 188, 237, 283, 1072, 1073, 1827

Graphviz, 104
INDEX

Gresh D. L., 1208
GRID, 54
GRID([SIZE1, SIZE2, SIZE3]), 745
Grinberg E. Ya., 662
group_skip_isolated_item, 53, 136, 151, 152, 169, 171, 234, 271, 297, 322, 331, 641, 1079, 1088, 1106, 1804
groups of values, see assignment to the same set of values
gt, 112, 153, 155, 177, 283, 1000, 1030, 1098, 1292, 1326, 1410
guillotine cut, 238, 883, 1907
Guo Q., 186, 872
Guy R. K., 315

H

Haggkvist R., 434
Hall interval, 238, 439, 1037
Hamiltonian, 239, 663, 1874
Hamming distance, 213
Han Hoogeveen J. A., iii
Hanák D., iii
Hansen P., 376
Harary F., 1898
hard version, 86
Hardy G. H., 946
Harvey W., 99, 1292, 1298, 1304, 1310, 1658
Harvey W. D., 250, 890
Hebrard E., iii, 100, 144, 213, 376, 478, 538, 552, 704, 1466, 1618, 1714, 1716, 1720, 1726, 1730, 1936
Hellsten L., 632, 1708, 1798, 1802
Henriksen J. G., 5
Henz M., 1854
Hermenier F., iii, 144, 1142
Hernández B. M., 184
heuristics, 142, 239, 877, 891, 950, 959, 1022, 1055, 1189, 1195, 1203, 1277, 1281, 1293, 1300, 1305, 1312, 1835
heuristics and Berge-acyclic constraint network, 239
heuristics and lexicographical ordering, 241, 1277, 1281, 1293, 1300, 1305, 1312
heuristics for two-dimensional rectangle placement problems, 241, 877, 1022
highest_peak, 113, 169, 171, 297, 307, 863, 1100, 1579
Hnich B., 78, 100, 213, 376, 474, 478, 538, 552, 704, 1266, 1292, 1298, 1304, 1310, 1316, 1466, 1618, 1814, 1936
Hoda S., 5
Hofacker I. L., 1660
Hooker J. N., i, 4, 5, 434, 662, 786, 962, 966
Hopcroft J., 662
INDEX

Hopper E., 320
Hungarian method for the assignment problem, 243, 1395
hybrid-consistency, 243, 1605, 1619
hyper arc-consistency, 154
hypergraph, 244, 503, 523, 1529, 1613, 1675, 1679, 1687, 1691

I

IF/PROLOG, 99
ifthen, 1104
Ilog, 442
Ilog CP Optimizer, 787, 912
Ilog Solver, 99, 1869
implied by, 87
implies, 87
implies (if swap arguments), 87
implies (items to collection), 87
in, 55, 113, 153, 164, 169, 172, 182, 214, 216, 221, 244, 291, 333, 479, 481, 590, 629, 630, 939, 1079, 1080, 1091, 1106, 1110, 1118, 1124, 1126, 1128, 1349, 1379, 1454, 1504
in_attr, 10
in_interval, 123, 153, 169, 172, 174, 214, 216, 245, 291, 292, 333, 405, 939, 1107, 1110, 1114, 1116, 1118, 1128
in_interval_reified, 131, 153, 177, 283, 292, 333, 437, 439, 1110, 1114
in_intervals, 116, 153, 216, 221, 245, 283, 333, 1110, 1118
in_list, 10
in_reified, 1114
in_relation, 36, 45, 120, 153, 208, 214, 221, 224, 293, 332, 722, 723, 726, 730, 734, 738, 959, 1120, 1944
in_same_partition, 126, 153, 169, 172, 177, 182, 214, 221, 277, 291, 334, 467, 468, 579, 580, 657, 658, 718, 719, 1106, 1107, 1124, 1407, 1408, 1653, 1654, 1749, 1764, 1933, 1934
in_set, 112, 188, 244, 283, 334, 607, 608, 685, 686, 746, 747, 826, 856, 857, 890, 892, 895, 896, 1106, 1107, 1110, 1128, 1199, 1200, 1216, 1217, 1320, 1499, 1500, 1503, 1504, 1506–1509, 1511, 1512, 1515, 1516, 1518, 1520, 1570, 1572, 1606, 1620, 1825, 1831, 1832, 1849, 1852, 1853, 1861, 1862, 1865, 1866, 1874, 1875
included, 244, 1107, 1128
inclusion, 244, 1239, 1243, 1247, 1919, 1925, 1933, 1937
incomparable, 106, 119, 283, 329, 336, 428, 429, 1130
incomparables, 1130
incompatible pairs of values, 245, 467
incr_sum, 1842
increasing_gcc, 1136
increasing_gcc_low_up, 1136
INDEX

increasing_global_cardinality, 120, 153, 159, 169, 172, 174, 256, 274, 291, 328, 334, 1042, 1132, 1156, 1542
increasing_global_cardinality_low_up, 1136
increasing_nvalue_chain, 116, 203, 270, 274, 1143, 1148, 1468, 1545
increasing_seq, 11
increasing_sum, 113, 155, 180, 274, 283, 324, 328, 1132, 1154, 1835
increasing_sum_ctr, 1154
increasing_sum_eq, 1154
increments_sum, 1842
indexed_sum, 121, 159, 335, 595, 601, 1156, 1835
indexing an array by a decision variable, see array constraint
indistinguishable values, 245, 1169, 1175, 1667
inequality_sum, 100
inflexion, 78–80, 113, 169, 171, 297, 307, 1058, 1160, 1578, 1579, 1940, 1941
inside, 1164
inside_sboxes, 192, 236, 252, 288, 756, 771, 777, 903, 1008, 1164, 1356, 1442, 1564
int_value_precede, 125, 148, 153, 169, 172, 174, 192, 245, 274, 291, 328, 335, 534, 1168, 1173, 1667
inter_distance, 430
intersection graph, 175, 176, 240
interval, 245, 455, 491, 571, 709, 939, 1111, 1118, 1180, 1185, 1227, 1243, 1429, 1645, 1739, 1755, 1925
interval_and_count, 85, 133, 159, 160, 162, 169, 171, 185, 192, 245, 295, 330, 331, 344, 495, 1178, 1185
interval_and_sum, 126, 159, 160, 162, 169, 171, 192, 245, 295, 330, 331, 344, 1180, 1184, 1835
inverse, 101, 110, 144, 153, 169, 171, 184, 217, 234, 237, 239, 266, 267, 269, 279, 280, 286, 337, 339–341, 830, 1188, 1195, 1198, 1199, 1202, 1203, 1855
inverse_channeling, 1188
inverse_in_range, 1202
inverse_offset, 126, 153, 184, 217, 234, 237, 239, 1189, 1194
inverse_range, 1202
inverse_set, 16, 121, 184, 188, 217, 299, 1128, 1189, 1198, 1203
inverse_within_range, 55, 106, 119, 178, 184, 217, 237, 239, 266, 328, 1189, 1199, 1202
inverse_offset, 286
Isermann H., 276
Italiano G. F., iii, 321, 322, 1884
ith_pos_different_from_0, 125, 152, 169, 171, 208, 221, 246, 330, 534, 1206

J

Jünger M., 662, 1570, 1824, 1874
Jackson J., 1528
INDEX

Jackson J. R., 912
JaCoP, iii, 2, 99, 99, 103, 762, 873, 1121, 1189, 1348, 1378, 1467, 1659, 1834
Jansen K., 305
Jaulin L., 306, 1484
Jefferson C., 99
Jensen J. L., 5
Johnson D. S., 430, 594, 934, 1898
joker value, 246, 451, 486, 744, 853, 1206, 1383, 1481, 1585, 1859, 1960
Jouglet A., iii
Jussien N., iii, 4, 434, 1798
Jørgensen M. E., 5

K

k_alldiff, 1208
k_alldifferent, 109, 150, 151, 159, 180, 181, 192, 209, 215, 218, 237, 249, 250, 275, 279, 300, 323, 329, 334, 435, 439, 690, 874, 877, 1021, 1208, 1468, 1631
k_alldistinct, 1208
k_cut, 116, 187, 188, 237, 250, 1128, 1216, 1318
k_diff, 538, 1467
k_disjoint, 34–36, 109, 192, 209, 215, 218, 329, 334, 899, 1218
k_same, 21, 109, 147, 193, 209, 220, 265, 279, 314, 329, 1222, 1226, 1227, 1230, 1231, 1234, 1235, 1239, 1625
k_same_interval, 115, 193, 209, 245, 279, 314, 329, 1223, 1226, 1243, 1645
k_same_modulo, 115, 193, 209, 262, 279, 314, 329, 1223, 1230, 1247, 1649
k_same_partition, 120, 193, 209, 277, 279, 314, 329, 1223, 1234, 1251, 1653
k_used_by, 12, 109, 193, 209, 244, 265, 314, 329, 1223, 1238, 1242, 1243, 1246, 1247, 1250, 1251, 1919
k_used_by_interval, 115, 193, 209, 244, 245, 314, 329, 1227, 1239, 1242, 1925
k_used_by_modulo, 115, 193, 209, 244, 262, 314, 329, 1231, 1239, 1246, 1929
k_used_by_partition, 120, 193, 209, 277, 314, 329, 1235, 1239, 1250, 1933
Kadioglu S., 239
Kalé L. V., 269
Kameugne R., 786
Kasper T., 662, 1570, 1824, 1874
Katriel I., iii, 5, 100, 210, 1034, 1208, 1222, 1238, 1604, 1622, 1630, 1634, 1884, 1918
Katsirelos G., 181, 434, 1040, 1208, 1466
Kaya L. G., 662
Keber R., 276
Kerbosch J., 257, 430, 434, 684, 912
Kirkman T. P., 5
Klarlund N., 5
Klee measure problem, 246, 877
Klee V., 872
knight, 220, 266
Koalog. 99
Kocjan W., iii, 1860, 1864
Korf R. E., 310
Kreuger P., iii, 1860, 1864
Kuchcinski K., iii, 872
Kuhn H. W., 243, 1394
Kutz M., 1208, 1222, 1238
Kızıltan Z., 78, 100, 376, 474, 478, 538, 552, 1266, 1292, 1298, 1304, 1310, 1316, 1466, 1618, 1622, 1814, 1936

L

L´opez-Ortiz A., 5, 430, 434, 1034
Labbé M., 838, 1868
labelling by increasing cost, 246, 950, 959
Labreuche C., 100, 344
Laburthe F., 99, 100, 103, 786
Laderux F., 1208
Larrosa J., 1826
Latin square, 249, 1213
Laura L., 321, 322, 1884
Lauri`ere J.-L., i, 98, 434, 662, 828
Law Y. C., 245, 1168, 1172, 1666
Le Pape C., 786, 912
Lecoutre C., iii, 404
Lee J. H. M., 1168, 1172, 1666
Leiserson C. E., 782
length_first_sequence, 113, 169, 171, 203, 307, 334, 1254, 1259
length_last_sequence, 113, 169, 171, 203, 307, 334, 1255, 1258
less, 1304
less_eq, 1310
Levy H., 824
lex, 1292, 1304
lex2, 109, 249, 256, 274, 283, 328, 329, 474, 1266, 1281, 1311, 1815
lex_alldiff, 1268
lex_alldifferent, 87, 109, 153, 178, 193, 209, 214, 221, 329, 336, 439, 447, 663, 866, 877, 1021, 1189, 1268, 1277, 1284, 1855
lex_alldistinct, 1268
lex_between, 129, 153, 169, 172, 174, 193, 249, 274, 291, 328, 329, 336, 1272, 1277, 1281, 1293, 1299, 1305, 1311
lex_chain, 1276, 1280, 1292, 1298, 1304, 1310
lex_chain_less, 109, 144, 153, 193, 209, 211, 221, 223–225, 241, 249, 256, 274, 328, 329,
INDEX

336, 789, 877, 1021, 1269, 1273, 1276, 1281, 1293, 1299, 1305, 1311, 1814, 1815

lex_chain_leqseq 109, 153, 193, 197, 209, 241, 249, 256, 274, 328, 329, 336, 474, 789, 877, 1021, 1137, 1267, 1273, 1277, 1280, 1293, 1299, 1305, 1311, 1532, 1533, 1536, 1537, 1545

lex_different 106, 119, 154, 169, 172, 174, 215, 221, 287, 291, 329, 336, 899, 1269, 1270, 1284, 1288, 1293, 1299, 1305, 1311, 1779

lex_equal 106, 119, 148, 154, 169, 172, 174, 178, 193, 266, 291, 336, 543, 544, 557, 558, 1284, 1288, 1299, 1311, 1485, 1488, 1492, 1625, 1944

lex_geq 1298


lex_greatereq 87, 106, 119, 154, 169, 172, 174, 193, 197, 214, 218, 241, 249, 256, 265, 274, 291, 328, 336, 731, 1273, 1277, 1281, 1284, 1288, 1293, 1298, 1305, 1311, 1773

lex_leq 1310


lex_lesseq_allperm 106, 119, 193, 249, 256, 274, 283, 328, 329, 336, 474, 476, 1311, 1316

lexeq 1298, 1310

lexicographic order 249, 474, 475, 723, 727, 731, 735, 739, 1267, 1273, 1277, 1281, 1293, 1299, 1300, 1305, 1311, 1312, 1316, 1317, 1815

leximin 1316

Le Huédé F., 100, 344

Li K., 872

limited discrepancy search 250, 891

line-segments intersection 252, 783, 1067, 1899

linear 1658, 1659, 1834

linear programming 250, 503, 663, 789, 941, 963, 967, 1216, 1319, 1571, 1691, 1824, 1831, 1874

Linhares A., 794

link_set_to_booleans, 118, 184, 188, 209, 214, 250, 299, 334, 442, 685, 941, 1128, 1216, 1318, 1570, 1618, 1619, 1824, 1861, 1862, 1865, 1866, 1874

Liu Q., 422, 868

Lloyd E. L., 824

Lock H. C. R., 786

Lodi A., 310, 1394

logic 252, 756, 771, 772, 777, 778, 903, 904, 1008, 1021, 1166, 1356, 1442, 1555, 1564, 1591, 1903, 1907, 1911, 1915

logigraphe 252, 749

longest_change 61, 123, 169, 171, 234, 271, 286, 307, 331, 633, 1322

LOOP, 54, 1059, 1080, 1684, 1801, 1805

Lopez P., 786

Lorca X., iii, 306, 1142, 1484, 1604, 1784, 1884

Low D. W., 824

lt 112, 154, 155, 177, 283, 1000, 1030, 1098, 1262, 1326, 1410

Lubiw A., 1266
Lucas E., 5

M

Müller T., 1854
Müller-Hannemann M., 594
Métivier J.-P., 100
Macho-Gonzalez S., 273
Maculet R., 224, 226
magic hexagon, 254, 439, 1055
magic series, 255, 1037
magic square, 255, 439, 1055
Mahéo M., 376
Mahajan M., 1208, 1222, 1238
Maher M. J., iii, 78, 195, 273, 404, 502, 1058, 1690
Maier D., 175
MALICE, 1036
map, 126, 187, 234, 237, 238, 286, 830, 1067, 1328, 1886
Marcovitch J., 99, 1034
Marriott K., 404
Marte M., iii, 1878
Martello S., 310, 594
Martin J., ii, iii, 2, 182, 252, 404
Martin M., 326
Martin P., 1208
Marx D., 1714, 1716, 1720, 1726, 1730
matching, 255, 1855
matrix, 256, 475, 690, 971, 1267, 1815
matrix, 970
matrix model, 256, 475, 690, 1267, 1815
matrix symmetry, 256, 474, 475, 1137, 1267, 1277, 1281, 1293, 1300, 1305, 1311, 1312, 1316, 1317, 1815
max, 1348
MAX_DRG, 60
MAX_ID, 60, 584, 604, 622, 626, 630, 664, 1568, 1789, 1871, 1875, 1961
max_index, 116, 257, 274, 1332, 1361
max_n, 125, 234, 257, 274, 286, 288, 1334, 1349, 1365
MAX_NCC, 61, 464, 642, 1080, 1324, 1801, 1805
MAX_NSCC, 61, 440, 443, 448, 452, 456, 459, 468, 472, 584, 588, 604, 826, 857, 1064, 1091, 1214, 1340, 1346, 1500, 1568, 1589, 1715, 1724, 1789, 1887, 1891, 1894
max_nvalue, 113, 159, 169, 171, 203, 220, 234, 257, 286, 334, 479, 763, 1036, 1338, 1369, 1468
MAX_OD, 61, 1875
max_size_set_of_consecutive_var, 113, 187, 234, 257, 286, 334, 1344, 1462
maximum, 257, 1332, 1335, 1339, 1345, 1349, 1352, 1518
maximum clique, 257, 431, 439, 685, 914
maximum number of occurrences, 257, 1339
maximum modulo, 125, 234, 257, 262, 274, 286, 1349, 1352, 1392
maxint, 257, 863, 1365, 1379, 1392
maxval, 15
McGregor J. J., 1072, 1826
Mcinnis M. J., 1208
Medjdoub B., 224, 225
meet, 1354
meet sboxes, 2, 193, 236, 252, 283, 288, 404, 756, 771, 777, 903, 1008, 1166, 1354, 1442, 1564
Mehlhorn K., 5, 434, 662, 1570, 1772, 1824, 1874
Mehta D., 4
member, 1106, 1106, 1128
Menana J., iii, 100
Menschner K., 872
Mercier L., 786
metro, 258, 1264
metro map, 258
Meyer M., 872
Miguel I., 78, 99, 474, 1266, 1292, 1298, 1304, 1310, 1316, 1814
Milano M., 100, 1394
Mildner P., iii
min, 1378
MIN_DRG, 61
MIN_ID, 61, 1875
min_index, 116, 260, 274, 1332, 1360
min_n, 125, 169, 171, 234, 257, 260, 274, 286, 288, 507, 1335, 1364, 1369, 1379, 1468
MIN_NCC, 62, 642, 1080, 1801, 1805
MIN_NSCC, 62, 664, 1091, 1370, 1376, 1825, 1875
min_nvalue, 113, 159, 169, 171, 203, 220, 234, 260, 262, 286, 334, 479, 763, 1036, 1339, 1368, 1468
MIN_OD, 62, 1875
min_size_set_of_consecutive_var, 113, 159, 187, 235, 260, 286, 334, 1374, 1462
min_weight_alldiff, 1394
min_weight_alldifferent, 1394
min_weight_alldistinct, 1394
minimum, 260, 1361, 1365, 1369, 1375, 1379, 1383, 1387, 1392, 1418, 1425, 1520
minimum cost flow, 261, 1727, 1751
minimum feedback vertex set, 262, 825
minimum hitting set cardinality, 262, 1468
minimum number of occurrences, 262, 1369
minimum_distance, 430
minimum_except_0, 125, 170, 172, 182, 235, 246, 257, 260, 274, 286, 291, 534, 1379, 1382
minimum_greater_than, 125, 148, 170, 172, 182, 214, 260, 274, 291, 1379, 1386, 1418, 1424, 1425
minimum_modulo, 125, 235, 257, 260, 262, 274, 286, 1352, 1379, 1392
minimum_spanning_tree, 100
minimum_weight_alldiff, 1394
minimum_weight_alldistinct, 1394
Minion, 99
MiniZinc, 2, 99, 103, 601, 762, 788, 873, 899, 1036, 1041, 1106, 1121, 1831
Minoux M., 57
minval, 14
miscellaneous, 142, 641, 1079, 1090, 1323, 1683, 1899
Mittal S., 1678
mod, 1616
modelling exercises, 144, 439, 447, 596, 601, 619, 749, 820, 877, 914, 950, 959, 1022, 1029, 1037, 1190, 1264, 1277, 1709, 1727, 1870
modulo, 1616
Monaci M., 310
Monette J.-N., 894
Monfroy É., 5, 1208
Moukrim A., 332
Mozart, 99, 936
multi-site employee scheduling with calendar constraints, 619, 877, 1021, 1022
multi_all_min_dist, 1400
multi_all_min_distance, 1400
multi_contiguity, 1398
multi_global_contiguity, 108, 193, 283, 297, 1058, 1079, 1398
multi_inter_distance, 125, 150, 168, 180, 193, 283, 301, 334, 431, 625, 789, 1400
multicost_regular, 100
multiset, 265, 1223, 1239, 1625, 1631, 1636, 1919
multiset ordering, 265, 1293, 1300, 1305, 1312
Museux N., iii
INDEX

Müller T., 189

N

n-Amazon, 266, 439, 447, 1190, 1709
n-queen, 269, 439, 447, 1190
Nagao T., 253
NARC_NO_LOOP, 63, 472
Narodytska N., 181, 434, 502, 1040, 1208, 1466, 1690
Naveh Y., 1208
nbchanges, 632
NCC, 63, 604, 642, 747, 832, 836, 840, 844, 1059, 1080, 1217, 1330, 1474, 1534, 1538, 1546, 1560, 1568, 1596, 1606, 1789, 1871, 1887, 1891, 1894
nclass, 127, 203, 220, 221, 235, 270, 277, 286, 322, 334, 1124, 1406, 1415, 1429, 1459, 1468
ncross, 1066
negation, 88
Nelissen J., 276, 872
neq. 18, 88, 112, 154, 155, 177, 283, 329, 439, 1000, 1030, 1098, 1262, 1326, 1410, 1412, 1450
neq_cst. 123, 154, 155, 177, 215, 283, 1002, 1410, 1412
nequivalence, 125, 193, 203, 220, 222, 235, 270, 286, 322, 334, 1407, 1414, 1429, 1459, 1468
Nethercote N., 404
next_element, 132, 170, 172, 183, 208, 214, 260, 291, 330, 1379, 1386, 1418, 1425
next_greater_element, 125, 208, 214, 260, 274, 330, 1379, 1386, 1418, 1448
Ng M. P., 1784
Ngo-Kateau Y., 786
Nieuwenhuis R., 5
Nightingale P., 296, 434
ninterval, 17, 125, 193, 203, 220, 235, 245, 270, 286, 322, 334, 1407, 1415, 1428, 1459, 1468
no cycle, 265, 1605
no_cycle, 100
non-crossable unavailability period, 301
non-deterministic automaton, 269, 479, 633, 1709
non-overlapping, 269, 877, 909, 1021, 1027, 1029, 1441, 1442, 1555, 1559, 1591, 1903, 1911
non-resumable task, 301
non_increasing_size, 12
non_overlap, 1440
non_overlap_or_same_end, 916
non_overlap_or_same_start, 918
non_overlapping, 1440
nonogram, see logigram
North S. C., 104, 405
not_alldiff, 1770
not_alldifferent, 1770
not_alldistinct, 1770
not_distinct, 1770
not_in, 113, 154, 170, 172, 182, 193, 214–216, 219, 221, 291, 333, 334, 1079, 1080, 1107, 1454, 1801, 1805
npair, 19, 36, 116, 193, 203, 220, 235, 270, 275, 286, 322, 334, 1407, 1415, 1429, 1458, 1468
npoin, 1484
npoints, 1484, 1490
NSCC, 63, 540, 544, 554, 558, 669, 745, 1091, 1144, 1408, 1416, 1430, 1452, 1460, 1464, 1470, 1478, 1482, 1488, 1492, 1732
nset_of_consecutive_values, 113, 187, 235, 286, 322, 334, 1344, 1374, 1462
NSINK, 63, 706, 710, 714, 719, 1126, 1626, 1632, 1637, 1642, 1646, 1650, 1654, 1774, 1920, 1926, 1930, 1934
NSINK_NSOURCE, 64, 1740, 1744, 1749, 1752, 1756, 1760, 1764, 1768
NSOURCE, 64, 508, 706, 710, 714, 719, 998, 1126, 1626, 1632, 1637, 1642, 1646, 1650, 1654, 1774, 1846, 1920, 1926, 1930, 1934
nth, 946, 958
NTREE, 64, 568, 669, 832, 836, 840, 844, 867, 1330, 1396
number of changes, 270, 633, 651, 657, 661, 672, 849, 853, 1709
number of distinct equivalence classes, 270, 539, 543, 553, 557, 1143, 1407, 1415, 1429, 1459, 1469, 1477, 1486, 1491
number of distinct values, 270, 531, 539, 553, 694, 700, 1143, 1149, 1468, 1469, 1473, 1477, 1481
nurse scheduling, 240
nval, 15
nvalue, 28, 40–42, 63, 72, 73, 84, 86, 114, 147, 170, 171, 180, 181, 193, 198–201, 203, 216, 217, 220, 235, 236, 262, 270, 286, 287, 322, 334, 426, 439, 479, 486, 531, 538, 539, 552, 553, 561, 693, 694, 699, 700, 752, 763, 830, 1036, 1142, 1143, 1145, 1149, 1212, 1213, 1334, 1338, 1339, 1364, 1365, 1369, 1406, 1407, 1414, 1415, 1428,
INDEX

1429, 1450, 1458, 1459, 1466, 1472, 1473, 1476, 1477, 1480, 1481, 1484, 1485, 1731, 1845, 1879–1881
nvalue_on_intersection, 127, 187, 189, 193, 203, 235, 270, 286, 463, 705, 1468, 1472, 1641
nvalues, 123, 193, 203, 220, 222, 270, 322, 334, 530–532, 667, 669, 692, 694, 695, 698, 700, 701, 1468, 1476, 1480, 1481
nvalues_except_0, 123, 193, 203, 222, 246, 270, 322, 334, 531, 534, 839, 840, 1468, 1477, 1480
nvectors, 36, 124, 193, 203, 217, 220, 222, 270, 322, 334, 336, 1288, 1484, 1485, 1490
NVERTEX. 65, 686, 826, 840, 844, 994, 1038, 1043, 1046, 1050, 1056, 1080, 1091, 1138, 1512, 1516, 1543, 1560, 1596, 1606, 1632, 1637, 1871, 1894
nvisible, 1494, 1496
nvisible_from_end, 114, 235, 286, 297, 1494, 1496
nvisible_from_left, 1496
nvisible_from_right, 1494
nvisible_from_start, 114, 235, 286, 297, 1494, 1496

O

O'Sullivan B., 213, 243, 310, 594, 1714, 1716, 1720, 1726, 1730
obscure, 271, 641, 1079, 1090, 1323, 1683, 1899
occurrencemax, 762
occurrencemin, 762
occurrence, 762
ogcc. 1510
Older W. J., 1622, 1772
one succ. 271
one_factor, 1854
one_machine, 912
ONESUCC, 70, 440, 443, 448, 456, 459, 468, 469, 568, 584, 604, 664, 669, 832, 836, 844, 867, 1500, 1506, 1589
one_succ, 439, 442, 447, 451, 455, 458, 467, 567, 583, 587, 603, 663, 667, 830, 831, 835, 866, 1395, 1567, 1588, 1886
open automaton constraint, 272, 1518, 1520
open constraint, 273, 439, 479, 535, 546, 1349, 1379, 1499, 1503, 1506, 1508, 1511, 1515, 1518, 1520, 1679
open_alldiff. 1498
open_alldistinct. 1498
open_among, 133, 188, 193, 203, 235, 273, 334, 479, 1106, 1128, 1502, 1506, 1508, 1511
open_atleast, 132, 168, 188, 222, 273, 334, 535, 1128, 1503, 1506, 1508, 1511
open_atmost, 132, 168, 188, 193, 273, 334, 546, 1128, 1503, 1506, 1508, 1511
open_distinct, 1498
open_gcc. 1510
open_global_cardinality, 129, 159, 189, 203, 227, 273, 334, 1037, 1041, 1128, 1499, 1502,
INDEX

1503, 1506, 1508, 1510, 1515
open_global_cardinality_low_up, 129, 159, 189, 203, 227, 273, 334, 1036, 1042, 1128, 1499, 1511, 1514
open_maximum, 116, 170, 172, 182, 260, 272–274, 291, 1349, 1518, 1520
open_minimum, 90, 116, 170, 172, 182, 260, 272–274, 291, 1379, 1518, 1520, 1889, 1890
Oplobedu A., 99, 1034
opposite_sign, 112, 154, 155, 177, 283, 1522, 1656
orchard, 54, 117, 150, 235, 236, 244, 286, 1528
ORDER, 65, 1333, 1336, 1350, 1362, 1366, 1380, 1384, 1393
order_constraint, 274, 475, 723, 727, 731, 735, 739, 858, 1132, 1137, 1143, 1149, 1155, 1169, 1173, 1267, 1273, 1277, 1281, 1293, 1300, 1305, 1312, 1317, 1332, 1335, 1349, 1352, 1361, 1365, 1379, 1383, 1386, 1387, 1392, 1425, 1518, 1520, 1533, 1537, 1542, 1545, 1600, 1667, 1815, 1816, 1820
ordered_atleast_npoint, 1532
ordered_atleast_npoints, 1532
ordered_atleast_nvector, 115, 203, 274, 328, 336, 543, 1281, 1305, 1311, 1485, 1532, 1533, 1537, 1545
ordered_atleast_nvectors, 1532
ordered_atmost_npoint, 1536
ordered_atmost_npoints, 1536
ordered_atmost_nvector, 115, 194, 203, 274, 328, 336, 557, 1281, 1305, 1311, 1485, 1533, 1536, 1537, 1545
ordered_atmost_nvectors, 1536
ordered_gcc, 1540
ordered_global_cardinality, 116, 154, 159, 194, 274, 334, 789, 1036, 1042, 1137, 1540
ordered_npoint, 1544
ordered_npoints, 1544
ordered_nvector, 115, 194, 203, 235, 274, 328, 336, 1143, 1149, 1281, 1305, 1311, 1485, 1533, 1537, 1544, 1545
ordered_nvectors, 1544
ordgcc, 1540
Ortega J., 1208
orth_link_ori_size_end, 110, 194, 209, 235, 275, 286, 877, 879, 1545, 1552, 1554, 1559, 1560, 1902, 1906, 1910, 1914
orth_on_the_ground, 117, 236, 275, 1548, 1552, 1591, 1592
orth_on_top_of_orth, 128, 236, 252, 269, 275, 1548, 1554, 1591, 1592
orthotope, 275, 431, 439, 820, 877, 883, 888, 914, 1549, 1552, 1555, 1559, 1591, 1903, 1907, 1911, 1915
orths_are_connected, 109, 190, 236, 269, 275, 331, 877, 1548, 1558, 1902, 1903
Ottosson G., 99, 103, 962, 966, 974
Ouellet P., 1400
overlap, 1562
overlap_sboxes, 194, 236, 252, 288, 756, 771, 777, 903, 1008, 1166, 1356, 1442, 1562
INDEX

overlapping alldifferent, 275, 1213

P

Péridy L., 912
Pachet F., 632, 1466
packing almost squares, 276, 877, 1022
Paige R., 5
pair, 275, 651, 1459, 1896
pair_atmost1, 550
pallet loading, 276, 877, 1022
Pardalos P. M., 262
Parreno F., 277
part of system of constraints, 88
partition, 277, 467, 579, 629, 657, 718, 1124, 1125, 1236, 1251, 1407, 1653, 1748, 1763, 1812, 1933
Partridge, 277, 877, 1022
PATH, 54, 427, 504, 634, 642, 652, 658, 850, 854, 859, 928, 1059, 1080, 1133, 1150, 1224, 1228, 1232, 1237, 1240, 1244, 1248, 1252, 1278, 1282, 1324, 1426, 1534, 1538, 1546, 1601, 1614, 1684, 1688, 1692, 1710, 1774, 1781, 1795, 1805, 1817, 1821
path, 277, 583, 936, 1022, 1570, 1571
path, 89, 116, 187, 210, 235, 237, 238, 271, 277, 332, 582, 603, 663, 936, 1128, 1318, 1567, 1570, 1571, 1870
PATH_FROM_TO, 66, 1294, 1301, 1306, 1313, 1572, 1789
path_from_to, 126, 189, 237, 250, 277, 936, 1128, 1318, 1570, 1870
PATH_LENGTH, 75
PATH_LENGTH(PATH_LEN), 836
PATH_N, 55, 1676
pattern, 36, 78, 120, 154, 170, 172, 174, 194, 291, 308, 331, 403, 633, 1079, 1574, 1687, 1800, 1804, 1812
pattern sequencing, 278, 798
peak, 114, 170, 171, 194, 297, 307, 1100, 1101, 1161, 1432, 1433, 1437, 1578, 1941
Pearl J., 39, 239
Pearson J., iii, iv, 292, 474, 1266, 1268, 1316, 1660, 1814
pentomino, 279, 877, 1021, 1022, 1595
period, 123, 180, 194, 235, 279, 283, 286, 297, 301, 331, 1582, 1584–1587
period_except_0, 123, 194, 235, 246, 279, 283, 287, 297, 301, 331, 534, 1583, 1584
period_vectors, 124, 194, 235, 279, 283, 287, 297, 336, 1583, 1586
periodic, 279, 1583, 1585, 1587
permutation, 279, 439, 444, 567, 641, 663, 667, 759, 830, 866, 993, 1189, 1213, 1223, 1227, 1231, 1236, 1588, 1625, 1631, 1636, 1645, 1649, 1653, 1773, 1780, 1855
permutation, 108, 151, 215, 271, 314, 334, 444, 1588
permutation channel, 280, 1190
Pesant G., ii, iii, 5, 78, 100, 172, 174, 279, 430, 434, 502, 632, 1034, 1120, 1142, 1574, 1708, 1798, 1802, 1854
Petersen J., 434
Petit T., ii, iii, 5, 78, 172, 305, 1142, 1154, 1540, 1726, 1730, 1734
Pfefferkorn C. E., 224
Phi-tree, 280, 789, 914
phylogeny, 282, 1788
pick-up delivery, 282, 831
Pinson E., 912
Pinto C., 297
Pitrat J., iii, 144, 252, 1034, 1036
place_in_pyramid, 101, 115, 236, 252, 269, 275, 877, 1548, 1552, 1554, 1555, 1590
placement problem, iv, 104, 160, 161, 185, 186, 241, 294, 326
placement space, 224, 225, 242, 275, 276, 289, 326, 332
planarity test, 282, 663
planning, 273, 305
Poder E., iii, 100, 186, 203, 242, 279, 317, 326, 786, 1018, 1024, 1268, 1582, 1584, 1734
polygon, 282, 877
polyomino, 111, 236, 279, 322, 1594
positioning constraint, 282, 883, 888, 1907, 1915
power, 123, 147, 155, 235, 283, 287, 330, 1016, 1598
precede, 1168, 1172
precedence, 110, 154, 209, 274, 914, 1132, 1168, 1172, 1600
PRED, 75, 840
pref_alldifferent_ctr, 100
pref_alldifferent_var, 100
pref_global_cardinality_low_up_ctr, 100
pref_global_cardinality_low_up_var, 100
preferences, 284, 723, 727, 731, 735, 739
Preparata F. P., 324
PROD, 67, 1603
prod, 15
producer-consumer, 284, 789, 820
PRODUCT(=), 519, 869, 1286, 1290, 1557, 1904, 1908, 1916, 1945
PRODUCT(CLIQUE, LOOP, =), 472
PRODUCT(PATH, VOID), 1294, 1301, 1306, 1313
product, 285, 804, 1602
PRODUCT(Comparison), 55
product ctr, 67, 123, 148, 155, 194, 285, 803, 805, 1602, 1608, 1835

INDEX
INDEX

program verification, 285, 825
propagator group, 176
proper_forest, 116, 187, 189, 235, 237, 243, 265, 328, 332, 333, 1128, 1604, 1886
Prosser P., 1784
proximity constraint, 285, 471, 923, 927
Prud’homme C., iii
PSTricks, 104
Puget J.-F., 99, 434, 1686, 1826
puzzles, 145, 439, 447, 749, 789, 831, 877, 950, 959, 1022, 1037, 1055, 1063, 1190, 1213, 1469, 1595, 1709

Q

quadtree, 287, 809, 877
queen, see n-queen
Quesada L., 4
Quesada L. O., iii, 934
Quimper C.-G., 5, 100, 181, 430, 434, 442, 502, 1034, 1040, 1208, 1268, 1400, 1466, 1690

R

Régin J.-C., iii, 5, 78, 100, 173, 174, 273, 430, 434, 538, 620, 624, 684, 688, 1034, 1040, 1052, 1154, 1466, 1498, 1510, 1514, 1540, 1686, 1726, 1730, 1826, 1854, 1884
Rampon J.-X., i
Randell D. A., 288, 754, 770, 776, 902, 1006, 1164, 1354, 1562
randomized filtering algorithm, 210
RANGE, 67, 445, 1610
range, 288, 1608
range, 14
range, 100, 1618
range consistency, 154
range_ctr, 68, 123, 155, 194, 222, 288, 1602, 1608, 1669, 1670, 1835
RANGE_DRG, 65, 1891
RANGE_NCC, 65, 568, 584, 588
RANGE_NSCC, 65, 563, 572, 576, 580
rank, 288, 1335, 1365
Rauhe T., 5
Razgon I., 1714, 1716, 1720, 1726, 1730
rectangle clique partition, 289, 1486
Refalo P., 940
regret based heuristics, 289, 950, 959, 1055, 1835
regret based heuristics in matrix problems, 290, 1055, 1835
regular, 100, 250, 279, 636, 1712
reified automaton constraint, 290, 510, 515, 518, 590, 595, 677, 681, 723, 749, 858, 941, 950, 955, 959, 963, 967, 971, 979, 983, 1010, 1055, 1104, 1110, 1125, 1132, 1137, 1143, 1169, 1173, 1273, 1285, 1289, 1293, 1299, 1305, 1312, 1349, 1356, 1379, 1383, 1387, 1403, 1418, 1433, 1451, 1454, 1518, 1520, 1525, 1575, 1660, 1793, 1804, 1812, 1816, 1820, 1903, 1911, 1962
reified constraint, 292, 1116
rel, 426, 434, 510, 514, 1030, 1098, 1104, 1262, 1292, 1298, 1304, 1310, 1326, 1410, 1524, 1525
related, 88
related to a common problem, 88
relation, 293, 1121, 1861, 1865
relaxation, 293, 451, 877, 1021, 1613, 1714, 1717, 1722, 1727, 1731, 1736, 1739, 1743, 1748, 1751, 1755, 1759, 1763, 1767, 1845, 1960
relaxation dimension, 294, 877, 1022
relaxed sliding sum, 17, 136, 244, 293, 297, 308, 314, 1612, 1691, 1835
remainder, 123, 155, 177, 235, 283, 287, 1616
require at least, 13
required, 12
Resende M. G. C., 262
resetting the domain of a variable, 305
resource constraint, 295, 596, 601, 694, 700, 789, 797, 798, 803, 804, 814, 820, 843, 895, 914, 916, 918, 1180, 1185, 1736, 1880, 1893
resumable task, 301
rgcc, 1035
Ribeiro C., 872
Richoux F., iii
Richter Y., 1208
Rivest R. L., 782
Rivreau D., 310, 912
Roach J. A., 1902
Rochart G., iii, 4, 434, 1798
Rochon du Verdier F., 872
Roditty L., 934
Rodriguez-Martín I., 838, 1868
root concept, 89
Rousseau L.-M., ii, 5, 100, 250, 502, 1034, 1884
Roussel O., 404
row and column lex, 1266
Roy B., 434
Roy P., 632, 1466
Rueher M., 5, 100, 1884
run of a permutation, \textit{296, 641}

\textbf{S}

Sabharwal A., \textit{502, 550}
Saclé J.-F., \textit{376}
Sadek R., \textit{279, 317, 326, 1018, 1024, 1268}
Sadjad S. B., \textit{1034}
Sagiv Y., \textit{1784}
Saidy H. R. D., \textit{304}
same and gcc, \textit{1630}
same and global cardinality, \textit{130, 159, 188, 194, 210, 220, 227, 265, 280, 334, 1036, 1212, 1213, 1624, 1625, 1630, 1635}
same and global cardinality_low up, \textit{130, 154, 159, 180, 188, 194, 210, 220, 227, 232, 265, 280, 334, 1042, 1045, 1625, 1631, 1634}
same and intersect, \textit{916}
same and non_overlap, \textit{916}
same gcc, \textit{1630}
same intersection, \textit{106, 119, 188, 189, 463, 705, 1473, 1625, 1640}
same modulo, \textit{127, 148, 188, 262, 280, 314, 329, 1230–1232, 1625, 1648, 1742, 1743, 1929}
same partition, \textit{129, 148, 188, 277, 280, 314, 329, 1124, 1234, 1235, 1237, 1625, 1652, 1746, 1747, 1933}
same sign, \textit{112, 154, 155, 177, 283, 930, 1000, 1522, 1656}
same size, \textit{13}
same start or disjunctive, \textit{918}
same start or non_overlap, \textit{918}
same with cardinalities, \textit{1630}
Samet H., \textit{808, 872}
Sandholm A., \textit{5}
Sanlaville E., \textit{100, 186}
Sanner A. M. W., \textit{422, 868}
Santaroni F., \textit{321, 322, 1884}
Saraswat V., \textit{189}
SAT, \textit{296, 439, 480, 877}
Saubion F., \textit{1208}
Savéant P., \textit{100, 344}
Savalle X., \textit{11}
Sbihi M., \textit{1018, 1276, 1280}
scalar product, \textit{297, 1055}
scalar product, \textit{23, 124, 148, 155, 194, 218, 222, 283, 324, 1157, 1185, 1658, 1834, 1835}
Schaus P., \textit{11, 100, 173, 174, 1778}
scheduling, 160, 182, 185, 186, 203, 273, 301
scheduling constraint, 263, 301, 431, 619, 694, 700, 789, 798, 804, 814, 820, 895, 909, 914, 916, 918, 1401, 1583, 1585, 1669, 1736
scheduling with machine choice, calendars and preemption, 619, 820, 877, 1021, 1022
Scheithauer G., 276
Schepers J., 297
Schiex T., iii
Schimpf J., 99, 1658
Schmitz L., 269
Schulte C., iii, 934
Schur number, 306, 1840
Schutt A., 786
Schwarz U. M., 305
Schwenk A. J., 1898
Scott J., 786
Sedgewick R., 1328
Sellmann M., 239, 1394
semantic links, 84
seq.bin, 633, 1143, 1709
sequence, 297, 503, 523, 641, 835, 863, 1058, 1079, 1090, 1101, 1161, 1255, 1259, 1398, 1433, 1437, 1494, 1496, 1579, 1583, 1585, 1587, 1613, 1660, 1675, 1679, 1683, 1687, 1691, 1804, 1812, 1941
sequence, 502, 1690
sequence dependent set-up, 298, 877, 914, 950, 959, 1870
sequence folding, 110, 170, 172, 177, 209, 236, 291, 297, 1660
sequencing with release times and deadlines, 299, 789, 820, 877, 895, 914
set channel, 299, 1199, 1319
set packing, 300, 1213
set.value.precede, 125, 189, 194, 245, 274, 283, 328, 335, 1168, 1666
sgcc, 1630, 1864
Shamos M. I., 324
shared table, 305, 987, 997
Shaw P., 594
Shearer J. B., 1062
Shen K., 99
shift, 126, 301, 330, 331, 1608, 1668, 1695
shift of concept, 89
Shikaku, 300, 877, 1022
Shmoys D. B., 1208
shortest path, 258
Shufet J. A., 662
SICStus, 2, 99, 99, 103, 243, 403, 873, 1121, 1189, 1779
sign, 50
sign, 1672
sign.of, 112, 154, 155, 177, 235, 283, 287, 1030, 1672
signature
INDEX

AUTOMATON
   and, 510
   change_vectors, 660
   clause_and, 676
   clause_or, 680
   cond_lex_cost, 722
   cond_lex_greater, 726
   cond_lex_greataeq, 730
   cond_lex_less, 734
   cond_lex_lessseq, 738
   consecutive_groups_of_ones, 748
   deepest_valley, 862
   elem_from_to, 954
   elementn, 982
   equivalent, 1010
   highest_peak, 1100
   imply, 1104
   inflexion, 1160
   int_value_precede, 1168
   int_value_precede_chain, 1172
   ith_pos_different_from_0, 1206
   length_first_sequence, 1254
   length_last_sequence, 1258
   lex_between, 1272
   nand, 1402
   no_peak, 1432
   no_valley, 1436
   nor, 1446
   nvisible_from_end, 1494
   nvisible_from_start, 1496
   open_maximum, 1518
   open_minimum, 1520
   or, 1524
   pattern, 1574
   peak, 1578
   stretch_path_partition, 1810
   valley, 1940
   xor, 1962

CC(\texttt{NSINK, NSOURCE}), PRODUCT
   same_intersection, 1640

CLIQUE
   bipartite, 606
   symmetric, 1852

CLIQUE, SUCC
   sliding_time_window, 1694

DISTANCE, CLIQUE(\neq)
   distance_between, 922

DISTANCE, PATH; AUTOMATON
   distance_change, 926

LOGIC
contains_sboxes, 754
coveredby_sboxes, 770
covers_sboxes, 776
disjoint_sboxes, 902
equal_sboxes, 1006
inside_sboxes, 1164
meet_sboxes, 1354
non_overlap_sboxes, 1440
overlap_sboxes, 1562

MAX_ID, MAX_NSCC, NCC, CLIQUE
  binary_tree, 602
  path, 1566

MAX_ID, MAX_NSCC, NCC, PATH_FROM_TO, CLIQUE
  stable_compatibility, 1784

MAX_ID, MAX_NSCC, RANGE_NCC, CLIQUE
  balance_path, 582

MAX_ID, MIN_NSCC, CLIQUE
  circuit, 662

MAX_ID, NCC, NVERTEX, CLIQUE
  temporal_path, 1868

MAX_ID, PRODUCT
  cardinality_atleast, 620
  cardinality_atmost, 624
  cardinality_atmost_partition, 628

MAX_ID, SUM, PRODUCT
  weighted_partial_alldiff, 1958

MAX_NCC, CIRCUIT, LOOP, V
  stretch_circuit, 1798

MAX_NCC, MIN_NCC, NARC, NCC, PATH; AUTOMATON
  change_continuity, 638

MAX_NCC, MIN_NCC, NCC, NVERTEX, PATH, LOOP; MAX_NCC, MIN_NCC, PATH, LOOP; AUTOMATON
  group, 1076

MAX_NCC, PATH; AUTOMATON
  longest_change, 1322

MAX_NCC, PATH, LOOP, V; AUTOMATON
  stretch_path, 1802

MAX_NCC, PRODUCT
  alldifferent_on_intersection, 462

MAX_NSCC, CLIQUE
  soft_all_equal_min_var, 1720

MAX_NSCC, CLIQUE
  alldifferent, 434
  alldifferent_between_sets, 442
  alldifferent_cst, 446
  alldifferent_except_0, 450
  alldifferent_interval, 454
  alldifferent_modulo, 458
  alldifferent_partition, 466
  golomb, 1062
  open_alldifferent, 1498
INDEX

permutation, 1588
soft_all_equal_max_var, 1714
MAX_NSCC, CLIQUE
max_nvalue, 1338
max_size_of_consecutive_var, 1344
MAX_NSCC, CLIQUE, ∨
k_alldifferent, 1208
MAX_NSCC, MIN_NSCC, NSCC, NVERTEX, CHAIN, AUTOMATON

group_skip_isolated_item, 1088
MAX_NSCC, NARC, NO_LOOP, PRODUCT(CLIQUE, LOOP, =)
alldifferent_same_value, 470
MAX_NSCC, NCC, CLIQUE
tree, 1884
MAX_NSCC, NCC, NVERTEX, CLIQUE; NVERTEX, CLIQUE, ∨
tree_resource, 1892
MAX_NSCC, NCC, RANGE, DRG, CLIQUE
tree_range, 1888
MAX_NSCC, NVERTEX, CLIQUE
cutset, 824
MAX_NSCC, RANGE, NCC, CLIQUE
balance_tree, 586
MIN_NSCC, CLIQUE
min_nvalue, 1368
min_size_of_consecutive_var, 1374
strongly_connected, 1824
NARC, CIRCUIT, AUTOMATON
circular_change, 672
NARC, CLIQUE(<)
all_min_dist, 430
diffn_column, 882
diffn_include, 886
disjunctive, 912
disjunctive_or_same_end, 916
disjunctive_or_same_start, 918
k_disjoint, 1218
lex_alldifferent, 1268
some_equal, 1770
NARC, CLIQUE(≠)
all_differ_from_at_least_k_pos, 422
all_incomparable, 428
disj, 894
soft_all_equal_min_ctr, 1716
NARC, CLIQUE(<)
soft_alldifferent_ctr, 1726
NARC, CLIQUE
inverse, 1188
inverse_offset, 1194
place_in_pyramid, 1590
NARC, CLIQUE(<)
allperm, 474
crossing, 782
graph_crossing, 1066
orchard, 1528
two_layer_edge_crossing, 1898
NARC, CLIQUE(≠)
symmetric_alldifferent, 1854
NARC, CLIQUE(≠): MAX_ID, MAX_OD, MIN_ID, MIN_NSCC, MIN_OD, CLIQUE(≠)
tour, 1874
NARC, NVERTEX, CLIQUE(≠)
clique, 684
NARC, PATH
all_equal, 426
among_seq, 502
k_same, 1222
k_same_interval, 1226
k_same_modulo, 1230
k_same_partition, 1234
k_used_by, 1238
k_used_by_interval, 1242
k_used_by_modulo, 1246
k_used_by_partition, 1250
lex_chain_less, 1276
lex_chain_lessseq, 1280
precedence, 1600
sliding_distribution, 1686
sliding_sum, 1690
NARC, PATH
change_partition, 656
relaxed_sliding_sum, 1612
NARC, PATH_1
size_max_starting_seq_alldifferent, 1678
NARC, PATH_1; AUTOMATON
arith_sliding, 522
NARC, PATH_N
size_max_seq_alldifferent, 1674
NARC, PATH; AUTOMATON
decreasing, 858
increasing, 1132
strictly_decreasing, 1816
strictly_increasing, 1820
NARC, PATH; AUTOMATON
change, 632
change_pair, 650
cyclic_change, 848
cyclic_change_joker, 852
smooth, 1708
NARC, PATH; NARC, PATH
increasing_nvalue_chain, 1148
NARC, PATH; NARC, PRODUCT; AUTOMATON
INDEX

stage_element, 1792

NARC, PATH; NARC, PRODUCT; SUCC
next_greater_element, 1424

NARC, PATH; NCC, PATH
ordered_atleast_nvector, 1532

NARC, PATH; NCC, PATH
ordered_atmost_nvector, 1536

NARC, PATH; NCC, PATH
ordered_nvector, 1544

NARC, PRODUCT
in_relation, 1120

NARC, PRODUCT
disjoint, 898

NARC, PRODUCT
correspondence, 758
element_product, 974
elements, 986
inverse_set, 1198
link_set_to_booleans, 1318
roots, 1618
symmetric_cardinality, 1860
symmetric_gcc, 1864

NARC, PRODUCT(=)
orth_on_top_of_orth, 1554
two_orth_column, 1906
two_orth_include, 1914
vec_eq_tuple, 1944

NARC, PRODUCT(=); AUTOMATON
differ_from_at_least_k_pos, 868
lex_different, 1284

NARC, PRODUCT; AUTOMATON
between_min_max, 590
element_sparse, 978

NARC, PRODUCT; AUTOMATON
not_in, 1454

NARC, PRODUCT(=); AUTOMATON
arith_or, 518
lex_equal, 1288
two_orth_are_in_contact, 1902

NARC, PRODUCT; AUTOMATON
among_low_up, 494
counts, 766
domain_constraint, 940
elem, 946
element, 958
element_greatereq, 962
element_leseq, 966
element_matrix, 970
in, 1106
in_interval, 1110
NARC, PRODUCT; NARC, PATH
sort permutation, 1778
NARC, PRODUCT, SUCC; AUTOMATON
minimum_greater_than, 1386
next_element, 1418
NARC, SELF
open_atleast, 1506
orth_link_ori siz_end, 1548
NARC, SELF
open_atmost, 1508
NARC, SELF
discrepancy, 890
open_among, 1502
orth_on_the_ground, 1552
NARC, SELF; AUTOMATON
arith, 514
atleast, 534
NARC, SELF; AUTOMATON
atmost, 546
NARC, SELF; AUTOMATON
among, 478
among_diff_r, 486
among_interval, 490
among_modulo, 498
count, 762
everything, 1012
NARC, SELF; CLIQUE, CC
shift, 1668
NARC, SELF; CLIQUE, SUCC
sliding_time_window_sum, 1702
NARC, SELF; MAX_NSCC, CLIQUE
dag, 856
NARC, SELF; NARC, CLIQUE(≠)
diffn, 872
NARC, SELF; NARC, CLIQUE(<); AUTOMATON
sequence_folding, 1660
NARC, SELF; NARC, PRODUCT
disjoint_tasks, 908
NARC, SELF; NCC, NVERTEX, CLIQUE(≠)
orphs_are_connected, 1558
NARC, SELF; PRODUCT, ∨, SUCC
coloured_cumulatives, 698
cumulative_with_level_of_priority, 812
cumulatives, 818
NARC, SELF; PRODUCT, SUCC
coloured_cumulative, 692
cumulative, 786
cumulative_convex, 794
cumulative_product, 802
track, 1878
INDEX

NARC, SYMMETRIC_PRODUCT (=) AUTOMATON
two_orth_do_not_overlap, 1910
NCC, CLIQUE
k_cut, 1216
NCC, CLIQUE
connected, 746
NCC, NTREE, CLIQUE
cycle, 828
NCC, NTREE, CLIQUE
map, 1328
NCC, NTREE, CLIQUE; NVERTEX, CLIQUE, PRED
cycle_or_accessibility, 838
NCC, NTREE, CLIQUE, PATH_LENGTH
cycle_card_on_path, 834
NCC, NTREE, NVERTEX, CLIQUE; NVERTEX, CLIQUE, ∀
cycle_resource, 842
NCC, NVERTEX, CLIQUE (≠)
polyomino, 1594
proper_forest, 1604
NCC, PATH, LOOP; AUTOMATON
global_contiguity, 1058
NCC, PRODUCT
nvalue_on_intersection, 1472
NSCC, CLIQUE
atleast_nvalue, 538
atleast_nvector, 542
soft_alldifferent_var, 1730
NSCC, CLIQUE
atmost_nvalue, 552
atmost_nvector, 556
NSCC, CLIQUE
nclass, 1406
nequivalence, 1414
ninterval, 1428
npair, 1458
nset_of_consecutive_values, 1462
nvalue, 1466
nvalues, 1476
nvalues_except_0, 1480
nvector, 1484
nvectors, 1490
NSCC, CLIQUE; AUTOMATON
not_all_equal, 1450
NSCC, CLIQUE; AUTOMATON
increasing_nvalue, 1142
NSCC, GRID ([SIZE1, SIZE2, SIZE3])
connect_points, 742
NSCC, NTREE, CLIQUE, ALL_VERTICES
circuit_cluster, 666
NSINK_NSOURCE, PRODUCT
INDEX

soft_same_interval_var, 1738
soft_same_modulo_var, 1742
soft_same_partition_var, 1746
soft_same_var, 1750
soft_used_by_interval_var, 1754
soft_used_by_modulo_var, 1758
soft_used_by_partition_var, 1762
soft_used_by_var, 1766

**NSINK, CC({NSINK, NSOURCE}), PRODUCT**
used_by, 1918
used_by_interval, 1924
used_by_modulo, 1928
used_by_partition, 1932

**NSINK, NSOURCE, CC({NSINK, NSOURCE}), PRODUCT**
same, 1622
same_interval, 1644
same_modulo, 1648
same_partition, 1652

**NSINK, NSOURCE, CC({NSINK, NSOURCE}), PRODUCT; NARC, PATH**
sort, 1772

**NSINK, NSOURCE, CC({NSINK, NSOURCE}), PRODUCT; NVERTEX, SELF, ∀**
same_and_global_cardinality, 1630
same_and_global_cardinality_low_up, 1634

**NSINK, NSOURCE, PRODUCT**
common, 704
common_interval, 708
common_modulo, 712
common_partition, 716

**NSINK, NSOURCE, PRODUCT; AUTOMATON**
in_same_partition, 1124

**NSINK, PRODUCT**
uses, 1936

**NSOURCE, PRODUCT**
among_var, 506
elements_sparse, 996

**NSOURCE, SUM, PRODUCT**
sum_of_weights_of_distinct_values, 1844

**NTREE, CLIQUE**
derangement, 866

**NTREE, RANGE, NCC, CLIQUE**
balance_cycle, 566

**NTREE, SUM, WEIGHT, ARC, CLIQUE**
minimum_weight_alldifferent, 1394

**NVERTEX, PRODUCT**
elements_alldifferent, 990

**NVERTEX, SELF, ∀**
ordered_global_cardinality, 1540

**NVERTEX, SELF, ∀**
global_cardinality, 1034
global_cardinality_low_up, 1040
INDEX

open_global_cardinality, 1510
open_global_cardinality_low_up, 1514
NVERTEX SELF; AUTOMATON
  increasing_global_cardinality, 1136
NVERTEX SELF; NARC SELF
  global_cardinality_low_up_no_loop, 1044
  global_cardinality_no_loop, 1048
NVERTEX SELF; SUM_WEIGHT_ARC PRODUCT
  global_cardinality_with_costs, 1052
ORDER CLIQUE
  max_index, 1332
  max_n, 1334
  maximum_modulo, 1352
  min_index, 1360
  min_n, 1364
  minimum_modulo, 1392
ORDER CLIQUE; AUTOMATON
  maximum, 1348
  minimum, 1378
  minimum_except_0, 1382
PATH_FROM_TO CLIQUE
  path_from_to, 1570
PATH_FROM_TO PRODUCT(PATH, VOID); AUTOMATON
  lex_greater, 1292
  lex_greatereq, 1298
  lex_less, 1304
  lex_leq, 1310
PATH_LOOP CC AUTOMATON
  sliding_card, 1682
PREDEFINED
  abs_value, 420
  atmost1, 550
  bin_packing_capa, 600
  calendar, 610
  colored_matrix, 688
  compare_and_count, 720
  consecutive_values, 752
  cumulative_two_d, 808
  distance, 920
  divisible, 930
  divisible_or, 932
  dom_reachability, 934
  domain, 938
  eq, 1000
  eq_cst, 1002
  eq_set, 1004
  gcd, 1016
  geost, 1018
  geost_time, 1024
  geq, 1030
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>geq_cst</td>
<td>1032</td>
</tr>
<tr>
<td>graph_isomorphism</td>
<td>1072</td>
</tr>
<tr>
<td>gt</td>
<td>1098</td>
</tr>
<tr>
<td>in_interval_reified</td>
<td>1114</td>
</tr>
<tr>
<td>in_intervals</td>
<td>1118</td>
</tr>
<tr>
<td>in_set</td>
<td>1128</td>
</tr>
<tr>
<td>incomparable</td>
<td>1130</td>
</tr>
<tr>
<td>increasing_sum</td>
<td>1154</td>
</tr>
<tr>
<td>leq</td>
<td>1262</td>
</tr>
<tr>
<td>leq_cst</td>
<td>1264</td>
</tr>
<tr>
<td>lex2</td>
<td>1266</td>
</tr>
<tr>
<td>lex_lesseq_allperm</td>
<td>1316</td>
</tr>
<tr>
<td>lt</td>
<td>1326</td>
</tr>
<tr>
<td>multi_global_contiguity</td>
<td>1398</td>
</tr>
<tr>
<td>multi_inter_distance</td>
<td>1400</td>
</tr>
<tr>
<td>neq</td>
<td>1410</td>
</tr>
<tr>
<td>neq_cst</td>
<td>1412</td>
</tr>
<tr>
<td>opposite_sign</td>
<td>1522</td>
</tr>
<tr>
<td>period</td>
<td>1582</td>
</tr>
<tr>
<td>period_except_0</td>
<td>1584</td>
</tr>
<tr>
<td>period_vectors</td>
<td>1586</td>
</tr>
<tr>
<td>power</td>
<td>1598</td>
</tr>
<tr>
<td>remainder</td>
<td>1616</td>
</tr>
<tr>
<td>same_sign</td>
<td>1656</td>
</tr>
<tr>
<td>scalar_product</td>
<td>1658</td>
</tr>
<tr>
<td>set_value_precede</td>
<td>1666</td>
</tr>
<tr>
<td>sign_of</td>
<td>1672</td>
</tr>
<tr>
<td>soft_cumulative</td>
<td>1734</td>
</tr>
<tr>
<td>strict_lex2</td>
<td>1814</td>
</tr>
<tr>
<td>subgraph_isomorphism</td>
<td>1826</td>
</tr>
<tr>
<td>sum_cubes_ctr</td>
<td>1838</td>
</tr>
<tr>
<td>sum_free</td>
<td>1840</td>
</tr>
<tr>
<td>sum_of_increments</td>
<td>1842</td>
</tr>
<tr>
<td>sum_squares_ctr</td>
<td>1850</td>
</tr>
<tr>
<td>symmetric_alldifferent_except_0</td>
<td>1858</td>
</tr>
<tr>
<td>twin</td>
<td>1896</td>
</tr>
<tr>
<td>visible</td>
<td>1946</td>
</tr>
<tr>
<td>PROD_SELF</td>
<td></td>
</tr>
<tr>
<td>product_ctr</td>
<td>1602</td>
</tr>
<tr>
<td>PRODUCT, ∀, SUCC</td>
<td></td>
</tr>
<tr>
<td>indexed_sum</td>
<td>1156</td>
</tr>
<tr>
<td>PRODUCT, SUCC</td>
<td></td>
</tr>
<tr>
<td>assign_and_counts</td>
<td>526</td>
</tr>
<tr>
<td>assign_and_nvalues</td>
<td>530</td>
</tr>
<tr>
<td>bin_packing</td>
<td>594</td>
</tr>
<tr>
<td>interval_and_count</td>
<td>1178</td>
</tr>
<tr>
<td>interval_and_sum</td>
<td>1184</td>
</tr>
<tr>
<td>RANGE_NSCC, CLIQUE</td>
<td></td>
</tr>
<tr>
<td>balance</td>
<td>560</td>
</tr>
<tr>
<td>balance_interval</td>
<td>570</td>
</tr>
</tbody>
</table>
INDEX

balance_modulo, 574
balance_partition, 578

RANGE, SELF
alldifferent_consecutive_values, 444
range_ctr, 1608

SUM_WEIGHT_ARC, PRODUCT
sliding_time_window_from_start, 1698

SUM, PRODUCT
sum, 1830
SUM, SELF
sum_ctr, 1834
sum_set, 1848

SYMMETRIC_PRODUCT
inverse_within_range, 1202

similarity, 632
Simonis H., iii, iv, 4, 99, 144, 243, 258, 276, 284, 309, 310, 317, 742, 812, 1208, 1264
Simons B. B., 430

size_max_seq_alldifferent, 55, 72, 114, 151, 186, 215, 235, 244, 287, 297, 308, 344, 437, 439, 539, 1499, 1674, 1678, 1679
size_max_starting_seq_alldifferent, 55, 114, 151, 186, 215, 235, 244, 273, 287, 297, 308, 344, 437, 439, 539, 1499, 1675, 1678

size_maximal_sequence_alldiff, 1674
size_maximal_sequence_alldifferent, 1674
size_maximal_sequence_alldistinct, 1674
size_maximal_starting_sequence_alldiff, 1678
size_maximal_starting_sequence_alldifferent, 1678
size_maximal_starting_sequence_alldistinct, 1678

ski assignment problem, 435
Skiena S., 52
SLAM problem, 306, 1486
Slaney J. K., 404
slice encoding, 305

sliding cyclic(1) constraint network(1), 306, 858, 1132, 1433, 1437, 1451, 1816, 1820
sliding cyclic(1) constraint network(2), 307, 633, 641, 849, 853, 863, 1101, 1161, 1255, 1259, 1379, 1709, 1941
sliding cyclic(1) constraint network(3), 307, 633, 641, 1323
sliding cyclic(2) constraint network(2), 308, 651, 927
sliding sequence constraint, 308, 503, 523, 835, 983, 1575, 1613, 1675, 1679, 1683, 1687, 1691, 1695, 1699, 1703, 1800, 1804, 1812

sliding_atmost, 1400
sliding_card_skip0, 133, 151, 170, 171, 271, 297, 308, 331, 479, 495, 534, 1036, 1682
sliding_distribution, 128, 194, 209, 244, 297, 308, 329, 503, 1040–1042, 1575, 1686, 1691, 1800, 1804
sliding_sum, 54, 132, 180, 194, 209, 227, 244, 250, 297, 308, 324, 329, 1613, 1687, 1690, 1835
sliding_time_window, 126, 194, 308, 330, 1669, 1694, 1698, 1699, 1702, 1703
sliding_time_window_from_start, 132, 194, 214, 308, 330, 1695, 1696, 1698
sliding_time_window_sum, 126, 194, 308, 324, 330, 331, 1695, 1702, 1835
Sloane N. J. A., 434, 828, 1884
smallest rectangle area, 310, 878, 1022
smallest square for packing consecutive dominoes, 309, 877, 1022
smallest square for packing rectangles with distinct sizes, 312, 878, 1022
Smith B. M., 1062, 1784
Smith L. M., 422, 868
Smolka G., 99
smooth, 125, 144, 170, 171, 174, 175, 194, 235, 266–270, 287, 307, 331, 633, 920, 1708
Soffa M. L., 824
soft constraint, 314, 1499, 1613, 1714, 1716, 1717, 1721, 1722, 1727, 1731, 1736, 1739, 1743, 1748, 1751, 1755, 1759, 1763, 1767, 1960
soft cumulative, 1540
soft variant, 89
soft_all_equal_max_ctr. 1726
soft_all_equal_max_var. 114, 154, 180, 293, 314, 334, 335, 426, 553, 1714, 1716, 1721, 1727, 1731
soft_all_equal_min_ctr. 114, 147, 180, 210, 293, 314, 334, 426, 553, 1714, 1716, 1721, 1727, 1731
soft_all_equal_min_var. 114, 154, 293, 314, 324, 334, 335, 426, 553, 1714, 1716, 1720, 1727, 1731
soft_alldiff_ctr. 1726
soft_alldiff_max_ctr. 1716
soft_alldiff_max_var. 552
soft_alldiff_min_ctr. 1726
soft_alldiff_min_var. 1730
soft_alldiff_var. 1730
soft_alldifferent. 86
soft_alldifferent_ctr. 114, 144, 151, 210, 211, 215, 261, 293, 314, 334, 435, 439, 553, 1714, 1716, 1721, 1726, 1731
soft_alldifferent_max_ctr. 1716
soft_alldifferent_max_var. 552
soft_alldifferent_min_ctr. 1726
soft_alldifferent_min_var. 1730
soft_alldifferent_var. 89, 114, 151, 215, 220, 293, 314, 322, 334, 335, 435, 439, 553, 1468, 1714, 1716, 1721, 1727, 1730, 1960
soft_alldistinct_ctr. 1726
soft_alldistinct_max_ctr. 1716
soft_alldistinct_max_var. 552
soft_alldistinct_min_ctr. 1726
soft_alldistinct_min_var. 1730
soft_alldistinct_var. 1730
soft_cumulative. 132, 283, 293, 295, 301, 314, 330, 789, 1734
soft_gcc_val. 100
soft_gcc_var. 100
soft_regular. 100
soft_same. 1750
soft_same_interval. 1738
soft_same_interval_var. 133, 188, 245, 293, 314, 335, 1645, 1738, 1755
soft_same_modulo. 1742
soft_same_modulo_var. 133, 188, 262, 293, 314, 335, 1649, 1742, 1759
soft_same_partition. 1746
soft_same_partition_var. 134, 188, 277, 293, 314, 335, 1124, 1653, 1746, 1763
soft_same_var, 64, 127, 188, 261, 293, 314, 335, 1625, 1739, 1743, 1747, 1750, 1767
soft_used_by, 1766
soft_used_by_interval, 1754
soft_used_by_interval_var, 133, 188, 245, 293, 314, 335, 1739, 1754, 1925
soft_used_by_modulo, 1758
soft_used_by_modulo_var, 133, 188, 262, 293, 314, 335, 1743, 1758, 1929
soft_used_by_partition, 1762
soft_used_by_partition_var, 134, 188, 277, 293, 314, 335, 1124, 1747, 1762, 1933
soft_used_by_var, 127, 188, 293, 314, 335, 1751, 1766, 1919
Solnon C., 1072, 1826
some_different, 435, 1208
some_equal, 108, 222, 314, 334, 439, 1770
Somogyi Z., 404
Soriano P., 1854
Sorlin S., 1072, 1826
sort, 314, 1773, 1780
sort, 70, 106, 107, 119, 180, 188, 199, 235, 280, 287, 314, 315, 344, 437, 439, 560, 1299, 1316, 1624, 1625, 1772, 1773, 1778, 1780
sort based reformulation, 431, 439, 444, 447, 451, 454, 458, 467, 471, 475, 752, 866, 914, 1223, 1227, 1231, 1236, 1239, 1243, 1247, 1251, 1588, 1625, 1645, 1649, 1653, 1770, 1919, 1925, 1929, 1933
sort based reformulation, 314
sort_permutation, 17, 129, 188, 214, 235, 280, 314, 437, 439, 758, 759, 959, 1132, 1284, 1773, 1778
sorted, 1772, 1778
sortedness, 1772, 1778
sorting, 1772, 1778
span, 748, 1798, 1799, 1802, 1803, 1811, 1811
sparse functional dependency, 315, 979, 997
sparse table, 315, 979, 997
specialisation, 89
sport timetabling, 315, 1855, 1859
spread, 100, 173, 174, 320, 324
squared squares, 315, 789, 877, 1022
Sriskandarajah C., 666
stable_compatibility, 67, 111, 177, 237, 282, 332, 1784, 1886
Stadler P. F., 1660
stageelem, 1792
stage_element, 121, 154, 170, 172, 177, 182, 208, 222, 235, 287, 291, 330, 950, 959, 1792
stage_elt, 1792
statistics, 320
Steel M., 1784
Steel M. A., 1784
Stergiou K., 1062
Stille W., 594
Stirling number of first kind, 829
Strash D., 434
stretch, 345, 633, 1709, 1798, 1802, 1810
stretch_circuit, 120, 154, 208, 218, 308, 331, 401, 1079, 1575, 1687, 1798, 1800, 1803, 1804
stretch_path, 120, 154, 170, 172, 174, 187, 218, 291, 297, 308, 331, 401, 641, 1079, 1090, 1254, 1258, 1575, 1687, 1799, 1800, 1802, 1810, 1812
stretch_path_partition, 120, 154, 170, 172, 174, 187, 277, 291, 297, 309, 331, 1575, 1804, 1810
strict_lex2, 109, 249, 256, 274, 283, 328, 329, 474, 1267, 1277, 1311, 1814
strip packing, 320, 877, 1022
strong articulation point, 321, 1886
strong bridge, 321, 663, 831
strongly connected component, 322, 539, 543, 553, 557, 567, 667, 744, 831, 839, 843, 1090, 1143, 1407, 1415, 1429, 1459, 1463, 1469, 1477, 1481, 1486, 1491, 1595, 1731, 1824
strongly_connected, 110, 189, 237, 250, 322, 663, 746, 1128, 1318, 1824
Stuckey P. J., 5, 404, 502
Subbarayan S., 186
subgraph_isomorphism, iii, 129, 189, 237, 283, 328, 1073, 1826
subset sum, 323, 1960
SUCC, 75, 528, 532, 597, 695, 701, 790, 799, 805, 815, 821, 1158, 1181, 1186, 1388, 1419, 1426, 1696, 1705, 1881
Sudoku, 323, 439, 1213
Sulanke T., 1172
SUM, 68, 1832, 1836, 1846, 1849, 1961
sum, 324, 1155, 1659, 1691, 1703, 1831, 1835, 1838, 1843, 1848, 1850
sum, 14
sum, 133, 198, 208, 235, 250, 324, 959, 1128, 1830, 1834, 1835, 1848
sumCtr, 50, 68, 76, 77, 123, 144, 148, 155, 194, 222, 289, 290, 299, 324, 329, 523, 595, 597, 789, 790, 797, 799, 814, 815, 820, 821, 1054, 1154, 1155, 1157, 1158, 1179, 1180, 1185, 1186, 1254, 1258, 1602, 1608, 1613, 1614, 1659, 1691, 1692, 1703, 1705, 1831, 1834, 1838, 1848, 1850
sum_cubes, 1838
sum_cubesCtr, 123, 148, 155, 194, 195, 222, 283, 324, 1835, 1838, 1850
sum_incr, 1842
sum_increments, 1842
sum_of_cubes, 1838
sum_of_cubesCtr, 1838
sum_of_increments, 114, 180, 195, 214, 283, 324, 1842
sum_of_squares, 1850
sum_of_squaresCtr, 1850
sum_pred, 1830, 1831
sum_set, 131, 155, 177, 189, 324, 1128, 1831, 1835, 1848
sum_squares, 1850
sum_squaresCtr, 123, 148, 155, 195, 222, 283, 324, 1835, 1838, 1850
INDEX

sum_weight, 1658, 1659
SUM_WEIGHT_ARC, 69, 1056, 1396, 1700
svc, 1630
svd, 1844
sweep, 324, 820, 877, 1021, 1027, 1029, 1722, 1955
Swinkels G. M., 1622, 1772
symm_alldiff, 1854
symm_alldiff_except_0, 1858
symm_alldifferent, 1854
symm_alldifferent_except_0, 1858
symm_alldistinct, 1854
symm_alldistinct_except_0, 1858
SYMMETRIC, 70, 424, 429, 608, 686, 745, 747, 1204, 1606, 1853
symmetric, 328, 423, 428, 607, 685, 744, 746, 1203, 1605, 1852
symmetric, 110, 189, 237, 328, 746, 1128, 1855
symmetric_alldiff, 1854
symmetric_alldiff_except_0, 1858
symmetric_alldifferent, 110, 151, 154, 184, 208, 215, 237, 238, 255, 280, 315, 331, 401,
439, 830, 1189, 1619, 1854, 1858, 1859
symmetric_alldifferent_except_0, 110, 246, 283, 315, 331, 1855, 1858
symmetric_alldistinct, 1854
symmetric_alldistinct_except_0, 1858
symmetric_cardinality, 122, 159, 189, 209, 227, 293, 331, 1036, 1128, 1318, 1860, 1865
symmetric_gcc, 30, 122, 159, 189, 209, 227, 293, 331, 346, 1036, 1128, 1318, 1861, 1864
SYMMETRIC_PRODUCT, 55, 1204
SYMMETRIC_PRODUCT(=), 1912
SYMMETRIC_PRODUCT(Comparison), 55
symmetry, 146, 328, 475, 723, 727, 731, 735, 739, 1021, 1022, 1137, 1143, 1155, 1169, 1175,
1267, 1273, 1277, 1281, 1293, 1300, 1305, 1312, 1317, 1533, 1537, 1545, 1667,
1815, 1827
system of constraints, 89, 329, 423, 428, 439, 475, 503, 690, 987, 997, 1037, 1213, 1219, 1223,
1227, 1231, 1235, 1236, 1239, 1243, 1247, 1251, 1267, 1269, 1273, 1277, 1281,
1687, 1691, 1815
Szczygiel T., 872
Szeder P., iii
Szymanek R., iii, 872
Szymanski T., 1784

T

table, 330, 950, 955, 959, 963, 967, 975, 979, 993, 997, 1020, 1206, 1418, 1425, 1793

table, 1120
Tack G., iii, 404, 934
Taghavi-Fard M. T., 304
Tallys H. Yunes, 1830
Tamarit J. M., 277
Tamassia R., 1898
INDEX

Tarjan R. E., 430, 662
temporal constraint, 330, 619, 694, 700, 789, 798, 804, 814, 820, 909, 1180, 1185, 1669, 1695, 1699, 1703, 1736, 1880
temporal_path, 117, 144, 187, 235, 237, 238, 277, 298, 299, 1567, 1570, 1868
ternary constraint, 330, 920, 971, 1016, 1598
Terno J., 276
Thiel A. J., 422, 868
Thiel S., iii, 5, 186, 434, 872, 898, 1034, 1406, 1414, 1428, 1466, 1622, 1630, 1634, 1772, 1844, 1854, 1918, 1958
Thorsteinsson E. S., 962, 966, 974
time window, 331
timetabling constraint, 331, 633, 641, 651, 657, 672, 690, 820, 849, 853, 877, 1021, 1027, 1079, 1090, 1180, 1185, 1323, 1575, 1583, 1585, 1669, 1683, 1709, 1800, 1804, 1812, 1855, 1859, 1861, 1865, 1880
Tollis I. G., 1898
Tong C., 224
topological constraint, 225
topological relation, 288
topological sort, 176, 375
Toth P., 594
touch, 331, 1559, 1903
tour, 61, 62, 110, 189, 237, 239, 250, 333, 663, 830, 1128, 1318, 1375, 1874
Tourbier Y., 99, 1034
track, 117, 214, 295, 330, 331, 694, 1468, 1878
transitive closure, 935, 936
tree, 332, 583, 587, 603, 1567, 1605, 1788, 1886, 1890, 1893
tree_precedence, 1210–1212

tree_resource, 89, 121, 187, 214, 237, 238, 295, 332, 345, 959, 1037, 1886, 1892
Trick M. A., 218, 1854
Truchet C., iii, 279, 317, 326, 1018, 1024, 1268, 1276, 1280
TRUE, 50
Trystram D., 305
tuple, 332, 1121, 1944
Turán P., 376
twin, 110, 195, 275, 283, 959, 1584

two-dimensional orthogonal packing, 332, 877, 1022
two_cycle, 1854
two_layer_edge_crossing, 134, 214, 235, 236, 252, 271, 287, 783, 1067, 1898
two_orth_are_in_contact, 122, 154, 170, 172, 174, 190, 236, 252, 269, 275, 291, 331, 1548, 1559, 1560, 1902, 1911
two_orth_column, 128, 236, 238, 252, 275, 282, 877, 883, 884, 888, 1548, 1906, 1915
two_orth_do_not_overlap, 52, 56, 90, 122, 154, 170, 172, 174, 178, 189, 236, 252, 266, 269, 275, 291, 877, 879, 1548, 1903, 1910
two_orth.include, 128, 236, 252, 275, 282, 877, 889, 1548, 1907, 1914

U

Ueda N., 253
Ullman J. D., 1784
Ullmann J. R., 1826
unary constraint, 333, 1107, 1110, 1118, 1454, 1840
unavailability period, 301
undirected graph, 333, 1605, 1874
used in graph description, 90
used in reformulation, 90
used by interval, 127, 148, 188, 195, 222, 244, 245, 314, 329, 1242–1244, 1645, 1754, 1755, 1919, 1924
used by modulo, 127, 148, 188, 195, 222, 244, 262, 314, 329, 1246–1248, 1649, 1758, 1759, 1919, 1928
used by partition, 129, 148, 188, 195, 222, 244, 277, 314, 329, 1124, 1250–1252, 1653, 1762, 1763, 1919, 1932
uses, 106, 119, 147, 148, 178, 188, 195, 222, 244, 266, 705, 1619, 1919, 1936
uses in its reformulation, 90

V

Valiant L. G., 434
Valiente G., 1826
valley, 114, 170, 171, 195, 298, 307, 862, 863, 1161, 1433, 1436, 1437, 1579, 1940
value constraint, 333, 426, 431, 439, 444, 447, 451, 454, 458, 463, 467, 479, 480, 486, 491, 495, 499, 514, 515, 518, 535, 546, 547, 561, 571, 575, 579, 621, 625, 629, 752, 763, 767, 868, 891, 899, 939, 1013, 1036, 1037, 1042, 1045, 1049, 1107, 1111, 1116, 1118, 1124, 1125, 1128, 1137, 1213, 1219, 1255, 1259, 1319, 1339, 1345, 1369, 1375, 1401, 1451, 1454, 1463, 1499, 1503, 1506, 1508, 1511, 1515, 1542, 1588, 1619, 1631, 1636, 1714, 1717, 1722, 1727, 1731, 1770, 1944
value partitioning constraint, 334, 539, 543, 553, 557, 1143, 1407, 1415, 1429, 1459, 1468, 1477, 1481, 1486, 1491
value precedence, 335, 1169, 1175, 1667
value symmetry, see indistinguishable values
value_precede, 1168
value_precede_chain, 1172
values, 1466, 1467
van Beck P., 5, 434, 1034, 1798, 1802
van der Veen J., 297
van Dongen M. R. C., 404
van Emden M. H., 1622, 1772
Van Hentenryck P., 78, 99, 100, 189, 210, 317, 326, 404, 786, 958, 1034
van Hoeve W.-J., iii, 5, 100, 273, 434, 502, 550, 890, 1034, 1498, 1510, 1514, 1622, 1726, 1750, 1884
van Lint J. H., 315
Van Roy P., 934
variable indexing, 335, 950, 955, 959, 963, 967, 979, 1157
variable subscript, 335, 950, 955, 959, 963, 967, 975, 1157
variable-based violation measure, 335, 426, 439, 1625, 1645, 1649, 1653, 1714, 1722, 1731, 1739, 1743, 1748, 1751, 1755, 1759, 1763, 1767, 1919, 1925, 1929, 1933
vec_eq_tuple, 106, 119, 154, 195, 332, 334, 1121, 1122, 1288, 1944
vector, 336, 423, 428, 475, 543, 633, 661, 723, 727, 731, 735, 739, 868, 1120, 1269, 1273, 1277, 1281, 1284, 1288, 1289, 1293, 1305, 1311, 1312, 1317, 1468, 1485, 1486, 1491, 1533, 1537, 1545, 1583, 1587
Vellino A., 202, 434
Vempaty N. R., ii
Vilím P., 280, 281, 786, 912
visible, iv, 135, 209, 236, 283, 324, 877, 1021, 1027, 1442, 1946
Voß H., 104
VOID, 56
Voss S., 278, 794
vpartition, 336, 1079

W

Wainwright R., 278
Wallace M. G., 99, 404, 406
Wallace R. J., iii, 722, 726, 730, 734, 738
Walsh T., iii, 78, 100, 181, 213, 376, 404, 434, 442, 474, 478, 502, 538, 552, 704, 1040, 1062, 1172, 1208, 1266, 1268, 1292, 1298, 1304, 1310, 1316, 1466, 1618, 1622, 1690, 1814, 1936
Wang C. C., 824
Weakley W., 216
Wei W., 1784
weighted assignment, 337, 1055, 1395, 1845, 1960
weighted_partial_alldiff, 133, 151, 159, 202, 236, 246, 293, 314, 323, 337, 435, 439, 450, 534, 1055, 1395, 1731, 1845, 1958
weighted_partial_alldifferent, 1958
weighted_partial_alldistinct, 1958
weightedSum, 1658
Weihe K., 594
Williams H. P., 434
Wilson N., 722, 726, 730, 734, 738
Wilson R. M., 315
Wolf A., 786
workload covering, 337, 820
Wormald N. C., 1784
INDEX

wpa, 1958
Wright E. M., 946
Würtz J., 189

X

xeqy, 1000
xexpyeqz, 1598
xgteqy, 1030
xgty, 1098
xleqy, 1262
xlty, 1326
XML schema, 404

Y

Yan H., 4, 434, 786
Yanasse H. H., 794
Yannakakis M., 175
Yannou B., 225
Yap R. H.C., 100

Z

Zampelli S., iii, 1018, 1276, 1280, 1826
Zanarini A., 100, 434
zebra puzzle, 337, 439, 950, 959, 1190
zero-duration task, 342, 694, 700, 789, 804, 814, 820, 914, 916, 918
Zhou J., 1772, 1778
Zhou N.-F., 742
Zimmermann W., 4