

SICS/R-90/9004

**Relating Attribute Grammars and
a Constraints-Prolog
Programming Environment**
by
Philippe Mathieu and Torbjörn Keisu

Workshop on Programming Environments, Nässlingen, Aug. 1989.

RELATING ATTRIBUTE GRAMMARS AND A CONSTRAINTS-PROLOG PROGRAMMING ENVIRONMENT

PHILIPPE MATHIEU and TORBJÖRN KEISU

Swedish Institute of Computer Science,

P. O. Box 1263, S-164 28 Kista, Sweden

Internet: philippe@sics.se, keisu@sics.se

=DRAFT=

Abstract. *In this short paper, we show how to relate a constraint programming environment with some kind of functional attribute grammars.*

*Kan vi så mycket vara
att vi betyder allt.*

— H. Martinson, Gåtan

Contents

Introduction	2
Constraints in PROLOG	2
Attribute Grammars	4
The Link	7
Conclusion	11
References	12

Introduction

It is well-known that there is a strong isomorphism between pure well-formed functional attribute grammars and definite clause programs with dependency function. Interestingly enough, the functional attribute grammars that appears in this connection are of a special type: they do not involve extra conditions on the input positions (C_p conditions in this paper). Keeping track of this part of the definition of attribute grammars seems to be the key of an isomorphism — extending the previous one — between some kind of attribute grammars, indeed *those that we really want to use in practice*, and definite clause programs (with constraints). The purpose of this paper is just to give some indications about such a correspondence. In the first section we give the definition of a program with constraints.

The second section is devoted to a very short survey of what is needed from the theory of attribute grammars. We suppose that the reader already have some knowledge about this theory. The third section explains the correspondence mentioned above, and is the core of the paper.

1. Constraints in Prolog

In this paper, we adopt the following definition.

1. DEFINITION. A program (with constraints) is a set of definite clauses with constraints.

2. DEFINITION. A definite clause with constraints is an expression of the form

$$(*) \quad A :- C_0, B_1, \dots, B_n, C_n$$

where, A and the B_i are atomic formulae, i. e., as usual, expressions of the form $p(t_1, \dots, t_m)$ where p is a predicate, and the t_i are terms, and the C_i are sets of constraints.

A constraint is defined as a predicate typed by a calculable ring (Here, a ring is a commutative, unitary and Noetherian ring:)

3. DEFINITION. Given a calculable ring A , a constraint is an atomic formula

$$p(t_1, \dots, t_{n_p}) :: A$$

where the predicate p and the terms t_i are expressions in the language of A .

The definition of a calculable ring is a little bit intricately; and it is given here for the seek of completeness.

4. DEFINITION. A ring \mathbb{A} is calculable if the following conditions are satisfied.

- (1) There is a coding for \mathbb{A} . By this we mean that there is a morphism $\text{cod}_{\mathbb{A}}$ from a subset $I_{\mathbb{A}}$ of a monoid $\Sigma_{\mathbb{A}}^*$ constructed on a finite alphabet $\Sigma_{\mathbb{A}}$ onto \mathbb{A} that codes \mathbb{A} as a ring;
- (2) There is an algorithm that solves generically the problem of explicitly giving with respect to the above coding the set of solutions of a (not necessarily homogeneous) linear system of equations in \mathbb{A} .

We drop the index \mathbb{A} when no confusion is possible from the context.

In extend, the first part of the previous definition means that there are algorithms to test the following properties:

- (1) If $\sigma \in \Sigma$, test if $\sigma \in I$;
- (2) If $\sigma_1 \in I$ and $\sigma_2 \in I$, test if $\text{cod}(\sigma_1) = \text{cod}(\sigma_2)$;

The second part of the definition means that there are algorithms to solve the following problems:

- (1) Find the code in I of a system of generators of the \mathbb{A} -module of solutions of the equation system

$$\sum_{j=1}^n a_{ij}x_j = 0 \quad (i = 1, \dots, m)$$

where the elements a_{ij} are of the form $\text{cod}(\sigma)$;

- (2) Test if the system

$$\sum_{j=1}^n a_{ij}x_j = b_i \quad (i = 1, \dots, m)$$

has a solution and in case of a positive answer, explicitly find some solution. Here too, the elements a_{ij} and b_i are of the form $\text{cod}(\sigma)$.

For examples and a more comprehensive exposition see [MK1] and [MK2].

A. Proof Trees. Without entering here into details, let us mention that a substitution is defined, as usual, relatively to a so-called Herbrand universum but have to respect the typing by calculable rings.

5. DEFINITION. A proof tree is an ordered and labeled tree whose labels are atomic formulae or are empty, each branch of the tree being associated to a set of constraints. The set of proof trees for a given program \mathfrak{P} is defined as follows:

- (1) If $A :- C$ is an instance of a clause c via a substitution θ then the tree consisting a two vertices whose root is labeled A or, sometimes (A, C) and

whose only leaf has the empty label (denoted \square) is a proof tree if \mathcal{C} is satisfied in the corresponding calculable rings;

- (2) If $\mathfrak{x}_1, \dots, \mathfrak{x}_n$ (for some n) are proof trees with roots labeled $(B_1, \mathcal{C}_1), \dots, (B_n, \mathcal{C}_n)$ and if $A :- C_0, B_1, \dots, B_n, C_n$ is an instance of a clause c via a substitution θ , then the tree consisting of the root labeled with (A, \mathcal{C}_0) and the subtrees $\mathfrak{x}_1, \dots, \mathfrak{x}_n$ is a proof tree if \mathcal{C}_0 is satisfied in the corresponding calculable rings.

6. DEFINITION. Let *goal* be a null-ary predicate of the language which does not occur in the clauses of a given program \mathfrak{P} . A computation for a program \mathfrak{P} given a clause c of the form

$$\text{goal} :- C_0, B_1, \dots, B_n, C_n$$

often written as

$$?- C_0, B_1, \dots, B_n, C_n$$

is a proof tree of the augmented program $(\mathfrak{P} \cup \{c\})$ with root labeled *goal*.

7. Remark. In a similar way, we define *partial proof trees*, and *partial computations* of a program: these computations are also referred to in the literature as *resolution trees*.

It should be clear that in a given program a constraint clause of the form (*) above is essentially equivalent to a clause of the form:

$$A :- C, B_1, \dots, B_n$$

From now on we will only consider constraint clause of the above type.

2. Attribute Grammars

We give here a very short summary of the definition and some properties of attribute grammars. All this material is well-known and is to be found in the existing literature (See for instance [DM2], a paper that we will often quote without explicitly referring to it.)

In this paper, an attribute grammar is defined as follows.

- 1. DEFINITION.** An attribute grammar is a triple

$$G = (\mathcal{G}, \mathcal{A}, \mathcal{R})$$

where

- (1) \mathcal{G} is a context-free grammar: (N, P) with N as the set of non-terminal symbols and P as the set of production rules;

- (2) \mathcal{A} is a family $(Attr(X))_{X \in N}$ of attributes of the non-terminal symbols (a finite set), such that, for each $X \in N$ we have a splitting $Attr(X) = Inh(X) \amalg Syn(X)$ by specifying disjoint sets: $Inh(X)$, the inherited attributes of X , and $Syn(X)$, the synthesized attributes of X ;
- (3) \mathcal{R} is a family $(R_p)_{p \in P}$ of functional semantic definitions.

The functional property of the R_p is defined below. But, first some more material.

2. DEFINITION. Let p be a production in P of the form

$$p : X_0 \rightarrow X_1, \dots, X_n$$

For each $i = 0, \dots, n$ and each a in $Attr(X_i)$, we introduce a new symbol $a(i)$, called the occurrence of the attribute a in p . The set of all such occurrences is denoted $Pos(p)$ and is called the set of positions of p .

3. LEMMA. A splitting of the positions of a production p is given by

$$Pos(p) = Input(p) \amalg Output(p)$$

where

$$Input(p) = \{a(i) \mid a \in Inh(X_0) \text{ or } a \in Syn(X_i) \text{ for } i > 0\}$$

and

$$Output(p) = \{a(i) \mid a \in Syn(X_0) \text{ or } a \in Inh(X_i) \text{ for } i > 0\}$$

Now we may define the functional property of the $(R_p)_{p \in P}$.

4. DEFINITION. The functional property of the (R_p) of the production p is the fact that every R_p is of the form:

$$\bigwedge_{x \in Output(p)} (x = t_x) \wedge C_p$$

where t_x is a term (in some predefined sorted logical language) with variables in $Pos(p) - \{x\}$ and C_p is a term with variables in $Input(p)$ only.

Possibly, the language permit us to consider some of the expressions C_p above as typed by a calculable ring \mathbb{A} . In this case, we write $C_p :: \mathbb{A}$

5. DEFINITION. In the case where some of the C_p are typed this way by calculable rings, we will speak about constraint attribute grammars.

From the very definition of an attribute grammar it is easy to define a dependency relation (cf. [CF]:)

6. DEFINITION. Let $p \in P$ be a production of an attribute grammar G . For $x \in \text{Output}(p)$ and $y \in \text{Pos}(p)$, we say that x depends on y if y appears as a variable in some term t such that $x = t$ is a conjunct in the first part, $\bigwedge_{x \in \text{Output}(p)} (x = t_x)$, of R_p . We write this relation as $x D_p y$.

The family of all the binary relations $(D_p)_{p \in P}$ is called the attribute dependency scheme of the grammar G .

A. Derivation Trees. Derivations trees are constructed by pasting together instances of production rules of P . For a given tree τ , we shall assume that an enumeration of the node is given.

7. DEFINITION. By a position in such a tree τ , we mean any pair $a(k)$ such that $a \in \text{Attr}(X)$ for some $X \in N$ and k is the number of the node of τ labeled by X .

The set of all positions of τ is denoted by $\text{Pos}(\tau)$.

8. DEFINITION. By an occurrence of a production rule p in a tree τ , we mean any subtree t of τ originating from p . We denote by P_τ the set of occurrences of the production rules from which τ is composed.

Now, we extend the dependency relation to the positions of a derivation tree.

9. DEFINITION. Define the dependency relation D_τ on the positions of τ as the union

$$D_\tau = \bigcup_{t \in P_\tau} D_t$$

where D_t is the relation on the positions of the subtree t induced by the relation D_p on the production rule $p \in P$ which t is an occurrence from.

10. DEFINITION. An attribute grammar $G = (\mathcal{G}, \mathcal{A}, \mathcal{R})$ is said to be well-formed if for every derivation tree τ of the grammar \mathcal{G} , the transitive closure of the relation D_τ is a partial order.

B. Decorated Trees. We outline here a syntactic attribute evaluation for well-formed attribute grammars; the idea consists in representing attribute values by terms constructed from the function symbols of the semantic definitions of the attribute grammar.

Let p be a production rule that occurs in a derivation tree τ . Each input position of the corresponding occurrence of p is either an output position a of an occurrence of a production rule p' , or it is a minimal element of the dependency relation D_τ of τ . In the former case, the value of a is determined by a term whose variables are input positions of p' , and this term can be substituted for a . Since

the attribute grammar is well-formed the value of each attribute position of τ will finally be represented by a term whose only variables are minimal positions with respect to the dependency relation D_τ

11. DEFINITION. *A derivation tree τ together with the construction described above is called a decorated tree if the described syntactic attribute evaluation validate the relations $(R_p)_{p \in P}$.*

3. The Link

Let \mathfrak{P} be a program (set of clauses). Let c be a clause of \mathfrak{P} of the form

$$A :- C, B_1, \dots, B_n$$

where

$$A = p(t_{00}, \dots, t_{0m_0})$$

$$B_i = q_i(t_{i0}, \dots, t_{im_i})$$

and

$$C = C_0 :: A_0, \dots, C_r :: A_r$$

From now on we restrict ourselves to the case where each calculable ring is a polynomial ring:

$$A = C[T_1, \dots, T_q], \text{ for some } q \geq 0$$

over some calculable ring C that is a subset of the constants of the language.

In that case:

$$A_i = C_i[T_{1i}, \dots, T_{q_i}]$$

A. Transforming constraint clause programs into constraint attribute grammars. Given \mathfrak{P} a program, we define N as the set of the predicate letters of \mathfrak{P} .

We keep the notations above for the clause c . To this clause, we associate a production rule:

$$p_c : X_0 \rightarrow X_1, \dots, X_n$$

where $X_0 \in N$ corresponds to p , and $X_i \in N$ to q_i .

The positions of X_0 in c are the set:

$$\{p1(0), \dots, pj(0), \dots, C_jk(0), \dots\}$$

where $pj(0)$ is the 0-position of an attribute pj corresponding to the term t_{0j} of the predicate p and $C_jk(0)$ is the 0-position of an attribute corresponding to the

polynomial variable T_{k_j} of the ring \mathbb{A}_j of the constraint C_j of the set \mathcal{C} . Let n_0 be the number of all such positions.

The positions of X_i in c are similarly the set:

$$\{\dots, q_{ij}(i), \dots\}$$

where $q_{ij}(i)$ is the i -position of an attribute q_{ij} corresponding to the term t_{ij} of the predicate q_i . Let n_i be the number of such positions.

To get further, we must introduce a concept related to constraint clause programs.

1. DEFINITION. A dependency function for the constraint clause program \mathfrak{P} is for each clause c of \mathfrak{P} a function d_c defined as follows.

- (1) If l is a pure predicate (i. e., an untyped predicate) of c then d_c is a function from the set $\{1, \dots, l(\text{arity of } l)\}$ into the set $\{\uparrow, \downarrow\}^{(\text{arity of } l)}$.
- (2) If $C :: \mathbb{A}$ is an element in \mathcal{C} then d_c is the function that applies each element in the set of variables of the polynomial ring \mathbb{A} onto the element \downarrow of the set $\{\uparrow, \downarrow\}^{(\text{number of variables in } \mathbb{A})}$. The dependency function for the program is then just the union of these functions.

2. DEFINITION. We call an argument assigned to \downarrow (resp. \uparrow) inherited (resp. synthesized).

With this definition, it makes sense to extend to clauses the terminology of the theory of attribute grammars as regards the definition of *Output*, ...

3. DEFINITION. If for each output position a of a clause, each variable of a occurs in some input position we say that the dependency function that determines the splitting of the position of the clause is safe.

From now on, we make the hypothesis that the dependency functions are safe.

Now, let us define the semantic rule R_p , as the conjunction of the following rules:

- (1) For each output position a of c we construct the following semantic definition. Let t_a be the term at the position a of c . For a variable x in t_a let b be an input position including x . Put

$$S_{xb} = \{\text{the set of all composed selectors } \pi \text{ such that } \pi(t_b) = x\}$$

The semantic definition for a is of the form

$$a = (t_a)^\theta$$

where θ is the substitution assigning to each variable x in t_a the term $\pi(b)$ for some π in S_{xb} .

- (2) For each pair of different occurrences of a variable x at input positions b_1 and b_2 of the clause c we construct the condition

$$\pi_1(b_1) = \pi_2(b_2)$$

where π_1 and π_2 are the selectors corresponding to the considered occurrences of x in the terms at the positions b_1 and b_2 .

- (3) For each input position b , if the term t_b is not a variable we construct the condition

$$\text{instance}(b, t_b)$$

- (4) For each constraint in \mathcal{C} , we add the corresponding semantic condition.

B. Transforming constraint attribute grammars into constraint clause programs. We start with a well-formed constraint attribute grammar G . To define a program \mathfrak{P} , we construct for each production rule

$$p : X_0 \rightarrow X_1, \dots, X_n$$

a corresponding clause c_p .

Let $Pos(p) = \{a_1, \dots, a_k\}$, where k is the sum of the number n_{a_i} of attribute of the several non-terminal symbols that occur in the production p . Assume that $a_i \preceq a_j$ for $i \leq j$. For $i = 1, \dots, k$ denote by x_i a variable corresponding to a_i for a_i an input position, and the term t_{a_i} from the semantic rule R_p for a_i an output position.

Now, to each non-terminal symbol we associate a predicate symbol with an arity equal to the number of attribute of this non-terminal symbol. For instance we associate to the above production, the predicate symbols p, q_1, \dots, q_n . To the conjuncts of the expression C_p we associate a set of constraints \mathcal{C} .

The clause corresponding to the above production rule is then:

$$p(t_{00}, \dots, t_{0n_{X_0}}) :- \mathcal{C}, \dots, q_i(t_{i0}, \dots, t_{in_{X_i}}), \dots$$

where t_{ij} is the former variable or term of the corresponding position of p (resp q_i).

C. The main result. We express it in the following theorem.

4. THEOREM. Under the two correspondences described above, the proof trees and the decorated trees are in a one-one correspondence.

Proof. It follows immediately from the detailed constructions that build the correspondence. ■

5. EXAMPLE. We just give here a raw example.

Let be the production rule:

$$X(x, y, z) \rightarrow \epsilon$$

with the semantic rules:

$$\begin{aligned} x &= f(y) \\ y &= g(z) \\ z^2 &= 1 :: \mathbf{rat} \end{aligned}$$

The corresponding clause ϵ should be of the form:

$$p(f(g(z)), g(z), z) :- z^2 = 1 :: \mathbf{rat}$$

which in turn gives the following production rule:

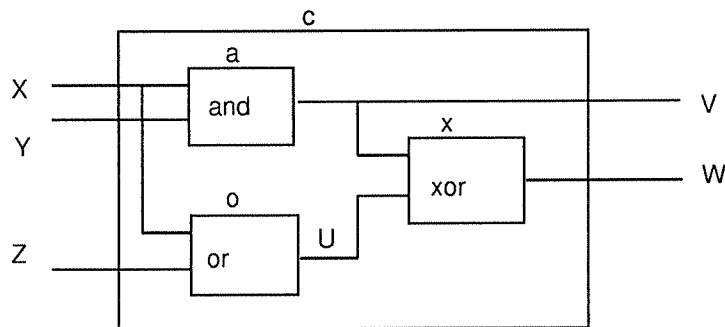
$$X_\epsilon(p1(0), p2(0), p3(0), C1(1)) \rightarrow \epsilon$$

with the semantic rules:

$$\begin{aligned} p3(0) &= (\pi1 - (f \circ g))p1(0) \\ (\pi1 - g)p2(0) &= (\pi1 - (f \circ g))p1(0) \\ C1(1) &= p3(0) \\ (C1(1))^2 &= 1 :: \mathbf{rat} \end{aligned}$$

Where **rat** are well-chosen polynomial rings over the rational field \mathbb{Q} .

6. EXAMPLE.



Such a circuit is described by a production rule of an attribute grammar

$$p : c \rightarrow a \circ x$$

with the attributes:

$$\text{Attr}(c) = \{X, Y, Z, V, W\}$$

$$\text{Attr}(a) = \{X, Y, V\}$$

$$\text{Attr}(o) = \{X, Z, U\}$$

$$\text{Attr}(x) = \{U, V, W\}$$

We have a natural splitting:

$$\text{Output}(p) = \{c.V, c.W, a.X, a.Y, o.X, o.Z, x.U, x.V\}$$

(The notation should be clear.)

The semantics rules are (\mathbb{B}_2 is the Boolean ring with two elements:)

$$c.V = a.V$$

$$c.W = x.W$$

$$x.U = o.U$$

$$a.X = c.X$$

$$o.X = c.X$$

$$a.Y = c.Y$$

$$o.Z = c.Z$$

$$x.V = a.V$$

$$a.V = c.X \cdot c.Y :: \mathbb{B}_2$$

$$x.W = a.V + o.U :: \mathbb{B}_2$$

$$o.U = c.X + c.Z + c.X \cdot c.Z :: \mathbb{B}_2$$

and the corresponding constraint clause is:

$$c(X, Y, Z, V, W) :- \mathcal{C}, a(X, Y, V), o(X, Z, U), x(U, V, W)$$

(with the evident dependency function.) Here, \mathcal{C} is the list:

$$V = X.Y :: \mathbb{B}_2, W = V + U :: \mathbb{B}_2, U = X + Z + X.Z :: \mathbb{B}_2$$

Conclusion

In this short paper, we have stated the correspondence between constraint attribute grammars and constraint clause programs. This correspondence have been illustrated by two “toy” examples. In a forthcoming article, we will explain how to use this correspondence in *e. g.*, protocol verification, geometric design, ...

The PROLOG language (with constraints) mentioned in the core of the paper is currently in course of implementation at the SICS.

References

- [CF] B. Courcelle and P. Franchi-Zannettacci, *Attribute Grammars and Recursive Program Schemes*, TCS **17** (1982).
- [DM1] P. Deransrart and J. Małuszyński, *Modelling Data Dependencies in Logic Programs by Attribute Grammars*, Research Report **323** (1984), INRIA.
- [DM2] P. Deransrart and J. Małuszyński, *Relating Logic Programs and Attribute Grammars*, Research Report (1985), Linköping University.
- [K] T. Keisu, "A Constraint-Solver Prototype," Thesis, The Royal Institute of Technology, Stockholm, 1989.
- [M1] P. Mathieu, *Attribute Grammars and AI Programming Languages for CAD/CAM*, Proceedings of the NordData Conference, Trondheim (1987).
- [M2] P. Mathieu, *Attribute Grammars and CAD/CAM*, Intern Report (1987), Commission of the European Communities.
- [MK1] P. Mathieu and T. Keisu, *On Constraint Programming*, Workshop on Constraint Programming, Pisa (1989).
- [MK2] P. Mathieu and T. Keisu, *On some Methods of Computer Algebra in Constraint Programming*, To appear.