Bilingual Conversation Interpreter:
A Prototype Interactive Message Translator
Final Report

by
Hiyan Alshawi, Carl Brown, David Carter,
Björn Gambäck, Steve Pulman and Manny Rayner
Bilingual Conversation Interpreter: A Prototype Interactive Message Translator Final Report

Hiyan Alshawi, Carl Brown, David Carter, Björn Gambäck, Steve Pulman and Manny Rayner

June 1991*

SICS research report R91:11
SRI Cambridge Report CCSRC-018
ISSN 0283-3638

Swedish Institute of Computer Science
Box 1263
S-164 28 KISTA
Stockholm, Sweden

SRI International
Cambridge Computer Science Research Centre
23 Millers Yard, Cambridge CB2 1RQ
Cambridge, England

© 1991 Swedish Institute of Computer Science, SRI International

*This is a joint report by SICS and SRI, which is published by both institutes simultaneously.
Executive summary

This document is the final report for a research project aimed at producing a prototype system for on-line translation of typed dialogues between speakers of different natural languages. The work was carried out jointly by SICS and SRI Cambridge. The resulting prototype system (called Bilingual Conversation Interpreter, or BCI) translates between English and Swedish in both directions.

The major components of the BCI are two copies of the SRI Core Language Engine, equipped with English and Swedish grammars respectively. These are linked by the transfer and disambiguation components. Translation takes place by analyzing the source-language sentence into Quasi Logical Form (QLF), a linguistically motivated logical representation, transferring this into a target-language QLF, and generating a target-language sentence. When ambiguities occur that cannot be resolved automatically, they are clarified by querying the appropriate user. The clarification dialogue presupposes no knowledge of either linguistics or the other language. The prototype system has a broad grammatical coverage, a initial vocabulary of about 1000 words together with vocabulary expansion tools, and a set of English-Swedish transfer rules. The formalisms developed for coding this linguistic information make it relatively easy to extend the system.

We believe that the project was successful in demonstrating the feasibility of using these techniques for interactive translation applications, and provides a sound basis for development of a large scale message translator system with potential for commercial exploitation.

The main sections of the report are the following:

- A non-technical introduction, summarizing the BCI's design, and containing a sample session.
- An overview of the Swedish version of the CLE.
- A detailed discussion of the theory and practice of QLF transfer.
- A description of the interactive disambiguation method.
- Suggestions for possible follow-on projects aimed in the direction of practically usable commercial systems.
Acknowledgments

The Bilingual Conversation Interpreter project was funded by the Swedish Institute of Computer Science\(^1\) and the work was carried out jointly by SICS and SRI. The greater part of it was completed while Manny Rayner was employed at SICS.

During the course of the project we received useful feedback and comments from many people, among whom we should particularly mention Ulli Block, Rudolf Hunze, Anna Lövgren, Christer Samuelsson and Stefanie Schachtl. Invaluable technical help on the many networking problems encountered was provided by Doug Moran, Mabry Tyson and Tommy Wallo. Last, but absolutely not least, we want to thank Rune Gustavsson, Janusz Launberg and Siwert Sundström for their help and support.

Hiyan Alshawi, David Carter, Steve Pulman and Manny Rayner
SRI International, Cambridge Research Centre

Carl Brown and Björn Gambäck
Swedish Institute of Computer Science

\(^1\)SICS is sponsored by the Swedish Board for Technical Development (STU) and the members of the SICS industrial consortium (Asea Brown Boveri, Bofors Elektronikindustrier AB, Telefon AB LM Ericsson, Försvarste Materielverk, IBM Svenska AB, and Swedish Telecom).
Contents

Executive summary .............................................................. i
Acknowledgments ............................................................... iii
Table of contents .............................................................. v
List of tables ................................................................. ix
List of figures ................................................................. xi

1 Introduction ................................................................. 1
  1.1 Project aims and summary ........................................... 1
  1.2 Translation by Quasi Logical Form transfer ......................... 2
  1.3 Current status of the BCI project ................................... 3
  1.4 Structure of the report ............................................... 7

2 Design philosophy of the system ....................................... 9

3 The Swedish Core Language Engine .................................... 13
  3.1 An overview of Swedish .............................................. 13
    3.1.1 Morphology ...................................................... 14
    3.1.2 Syntax .......................................................... 14
  3.2 Adapting the CLE to Swedish ....................................... 15
    3.2.1 Morphology ...................................................... 16
    3.2.2 The lexicon ..................................................... 17
    3.2.3 Syntax and semantics ......................................... 18

4 Quasi Logical Form transfer ............................................. 23
  4.1 The transfer formalism .............................................. 23
  4.2 Types of transfer rule .............................................. 25
    4.2.1 Identity rules .................................................. 26
    4.2.2 Atomic lexical rules .......................................... 26
    4.2.3 Non-atomic lexical rules ..................................... 27
    4.2.4 Structural rules ............................................... 28
    4.2.5 Discard rules .................................................. 29
  4.3 Effectiveness of QLF Transfer ..................................... 30
    4.3.1 Expressiveness .................................................. 30
    4.3.2 Simplicity and reversibility .................................. 31
    4.3.3 Monotonicity ................................................... 33
    4.3.4 Compositionality .............................................. 33
5 Disambiguation and interaction 39
  5.1 Recognizing a choice as spurious 39
  5.2 Selecting one option automatically 40
  5.3 Querying one of the users 41
  5.4 Recovery 43

6 Distributed development 45

7 Summary and further directions 47
  7.1 Summary 47
  7.2 A second-stage BCI 48
  7.3 Interactive translation of electronic mail 48
  7.4 Speech-to-speech translation 49
  7.5 The “multi-lingual workstation” 50
  7.6 Interactive multilingual composition of texts 50
  7.7 Conclusion 51

8 References 53

A BNF definition of the QLF notation 55

B BCI user manual 57
  B.1 Running the BCI 57
  B.2 Description of CLE commands 59
    B.2.1 Command types 59
    B.2.2 Basic commands 59
    B.2.3 Help commands 60
    B.2.4 Phase selection and control 61
    B.2.5 Printing mode 63
    B.2.6 Printing 64
    B.2.7 The stepper 66
    B.2.8 Parameter setting 67
    B.2.9 Loading and re-compilation of rules and lexicons 69
    B.2.10 Commands for dealing with unknown words 71
    B.2.11 Testing and debugging 72

C Using VEX, the CLE lexicon tool 73
  C.1 Getting started 73
  C.2 Morphology and syntax 74
  C.3 Dealing with inconsistencies 79
  C.4 Sortal restrictions 81
  C.5 Relational nouns 85
  C.6 Lexicon file processing 87
  C.7 A QVEX dialogue 90
  C.8 Backtracking 90
  C.9 Setting up VEX for your application 92

D Examples 93
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>E  The English and Swedish lexicons</strong></td>
<td></td>
</tr>
<tr>
<td>English lexicon</td>
<td>98</td>
</tr>
<tr>
<td>Swedish lexicon</td>
<td>118</td>
</tr>
</tbody>
</table>
List of Tables

1.1 A typical BCI dialogue ........................................... 6
3.1 Breakdown of work performed in Swedish adaptation of the CLE .... 15
4.1 Types of transfer rules ........................................... 26
4.2 Reasons for complex verb-transfer rules .......................... 27
4.3 Directionality of transfer rules ................................... 32
4.4 Examples of complex transfer types ............................... 34
4.5 Examples of transfer contexts .................................... 34
4.6 A complete compositionality table ............................... 35
List of Figures

1.1 System architecture for the BCI ........................................... 4
1.2 Example of QLFs ............................................................. 5
5.1 Example of BCI interaction .................................................. 42
Chapter 1

Introduction

1.1 Project aims and summary

For the past year, SRI International and the Swedish Institute of Computer Science have pursued a joint research project intended to investigate the feasibility of constructing a sophisticated semi-automatic Machine Translation system by configuring and adapting existing state of the art components centered around the SRI Core Language Engine. At the end of the project's first year, this has resulted in a prototype system, the "Bilingual Conversation Interpreter" (BCI), which is capable of interactive translation between English and Swedish, using a vocabulary of about 1000 words and a broad range of possible grammatical constructions. In this first chapter, we will give an overview of the project's design philosophy and the current state of implementation of the prototype BCI; the discussion will be kept at a fairly non-technical level. Subsequent chapters will delve into the system in more detail.

We will begin by describing the BCI's position within the field of Machine Translation as a whole. It is at the moment generally accepted among researchers in the field that fully automatic high-quality Machine Translation is not feasible as a short-term prospect, except within extremely limited sub-domains.\(^1\) Disregarding these, realistic projects must normally compromise, either by accepting low-quality output (which may subsequently be post-edited), or by allowing human interaction during the translation process to supply knowledge not directly available to the system.

We have chosen the second alternative, for the following reasons: firstly, there is a large class of applications where two monolingual humans can achieve a goal by carrying out a dialogue in real time; for the sake of concreteness, we have during the project focussed on a hypothetical application, where the BCI is being used by a Swedish car-hire firm in order to communicate with an English customer.\(^2\) Secondly, such an architecture allows practical systems to be built at the level of the current state of the art, while providing a smooth development path for future improvements. As the basis of Natural Language Processing (NLP) technology improves, less human interaction is required.

---

\(^1\)The example *par excellence* is weather forecasts, as exemplified in the well-known TAUM-METEO system (Thouin 1982).

\(^2\)A similar, though less sophisticated, system for translation between Japanese and English is reported in (Miike et al, 1988).
1.2 Translation by Quasi Logical Form transfer

The central technical idea in the BCI is the concept of Quasi Logical Form transfer. Here, we have attempted to create an intelligent compromise between the opposing paradigms of "syntactic (or semi-syntactic) transfer" and "knowledge-based interlingua", which we will first briefly summarize. In the syntactic transfer approach, which is in practice by far the more common one, translation is carried out in three stages: the source-language text is transformed into a syntactic representation (most commonly some kind of tree-like structure), which is then transferred into a target-language counterpart. Finally, the target-language text is generated from its syntactic representation. Knowledge-based interlingua-based systems, in contrast, perform translation in two stages: the source text is reduced to a language-independent intermediate representation, and the target text is generated from this representation directly. Very few systems are of course completely pure examples of either approach; in particular, many architectures based on syntactic transfer also employ some interlingual semantic ideas, of which the most important is usually a version of case-grammar. This does not, however, substantially affect the following discussion.

On the positive side, transfer is the easier alternative to implement, since the techniques of syntactic analysis (and to a lesser extent generation) are well-understood and relatively straightforward. However, the fact that different languages use widely different syntactic forms places a great burden on the transfer component, which becomes correspondingly more complex and harder to understand. To take an example from the English-Swedish language-pair: although the structures of *He hired a car* and the corresponding *Han hyrde en bil* are identical, transforming the sentence into a question already creates non-trivial problems. The Swedish inverts the word-order (*Hyrde han en bil?*), while the English introduces an auxiliary (*Did he hire a car?*), necessitating an extra rule. The problem is that the representation is too shallow to "factor out" each language's own way of forming questions. There tend to be many phenomena of this kind, which interact to form a rapidly growing set of complex transfer rules. In a multi-lingual transfer-based system another problem surfaces: a separate transfer-component is required for each language pair, implying that the number of transfer components is proportional to the square of the number of languages included.

On the other hand, pure interlingua systems do not suffer from these problems, since the intermediate representation is not tied to any particular language. The difficulty is rather that too little is as yet known about formal knowledge-representation techniques to make it feasible to specify a robust interlingua for more than a small subset of natural language; moreover, even if the theoretical apparatus were present, transformation to a language-independent form in general requires access to vast quantities of implicit "common-sense" knowledge, the formalization of which is a Herculean task. Although interesting experimental systems have been developed (for example, at Carnegie-Mellon University's World Center for Machine Translation), it seems unlikely that they can be turned into robust products in the short- or medium-term.

Our architecture is half-way between the two positions outlined above. The source text is analyzed into a representation ("Quasi Logical Form", or QLF), which has been carefully designed so as to represent exactly the aspects of linguistic meaning which do not involve context or "common-sense" knowledge. The source QLF representation is transferred into a target counterpart, from which target-language generation is used to produce the target text. In other applications, such as NL query interfaces to databases, the QLF
representation would be subjected to a further phase of processing which adds contextually determined factors, such as the referents of pronouns (Alshawi 1990). However, our hypothesis has been that a useful translation can be obtained by performing transfer directly on QLFs, when necessary dealing with problems of contextual interpretation by querying the user: for example, to translate the Swedish sentence Vi har en blå Volvo ("We have a/one blue Volvo"), the system would ask a question to determine whether en would more appropriately be translated as a or one. These questions are phrased in such a way as to assume no knowledge on the source-user's part of either linguistics or the target language.

Our judgement, based on the experience gained during the first year of the project, is that QLF-based transfer successfully circumvents many of the difficulties that arise using pure transfer or interlingua methods; it manages to factor out the problems caused by linguistics, which are reasonably tractable, and leaves those caused by knowledge, which are not. The result is a robust and modular architecture, which can be debugged and expanded with a relatively low expenditure of effort. In the next section, we describe the BCI's architecture in more detail, and show a sample session.

1.3 Current status of the BCI project

The main components of the BCI are two copies of the SRI Core Language Engine (CLE), a state of the art general-purpose tool for natural-language analysis and generation, equipped with English and Swedish grammars respectively. The basic system software and the English grammar and lexicon were written at SRI Cambridge Research Centre between 1986 and 1989, with an expenditure of about fourteen man-years of work. Adaptation of the English-language components to Swedish was done at SICS during 1990-91, and took about 16 man-months. The two copies of the CLE are linked by the transfer and interaction components, which are comparatively small pieces of software; the transfer component consists of an interpreter and a set of declarative transfer rules which can be extended in a modular way. The system is normally run on a pair of SPARC workstations under either SICStus or Quintus Prolog. The overall architecture of the system is shown in Figure 1.1 on page 4.

The CLE is capable of running both in analysis and generation modes, using a single grammar which is compiled in different ways for the two tasks; generation is performed using the Semantic Head-Driven algorithm (Shieber et al 1990). Analysis turns sentences into QLF representations, while generation works in the opposite direction. Intermediate stages include processing of morphology and syntax (grammar).

The English and Swedish grammars are both fairly large and cover most of the common constructions in their respective languages, including questions (YN- and WH-), topicalized clauses, imperatives, passives, relative clauses, negation, cleft constructions, conjunction, noun-phrase and verb-phrase modification by preposition-phrases, adjectives and adverbs, various kinds of complex determiners, proper names, dates and times, possessive constructions and about fifty different kinds of complements to verbs and adjectives. There is a good treatment of inflectional morphology, which for Swedish covers all main inflectional classes of nouns, verbs and adjectives.

The function-word lexicon for each language contains about 400 words, including most pronouns, conjunctions, prepositions, determiners, particles and "special" verbs. In addition, there is a "core" content-word lexicon with common nouns, verbs and adjectives: the English one contains about 900 more entries, while the Swedish version, which is still
Figure 1.1: System architecture for the BCI
under development, has about 350. New lexicon entries can be added by users using a
tool developed for the purpose.3

The QLF notation is a conservatively extended version of first-order logic, and is
perhaps best described here by illustration.4 Continuing the example from the first section,
the QLFs for *He hired a car* and *Did he hire a car?* are as shown in Figure 1.2.

[past,
 [hire_3p,
  qterm(ex,sing,A,[event,A]),
  a_term(ref(pro,he,sing,1( [] )) ,B,
        [and,[male,B],[personal,B]]),
  qterm(a,sing,C,[car1,C]])] ; in the past
 ; the 3-place relation "hire"
 ; obtained between an event,
 ; a male person referred to
 ; by the singular pronoun "he",
 ; and a car

QLF for *He hired a car.*

[ynq,
 [past,
  [hire_3p,
   qterm(ex,sing,A,[event,A]),
   a_term(ref(pro,he,sing,1( [] )) ,B,
        [and,[male,B],[personal,B]]),
   qterm(a,sing,C,[car1,C]])]
 ; yes-no question: is it true that
 ; in the past
 ; the 3-place relation "hire"
 ; obtained between an event,
 ; a male person referred to
 ; by the singular pronoun "he",
 ; and a car

QLF for *Did he hire a car?*

Figure 1.2: Example of QLFs

The main point to notice is that the representation of the second sentence differs
from that of the first only by having the operator ynq ("yes-no question") wrapped round
it. This principle of cleanly separating out distinct aspects of the sentence in its QLF
representation is adhered to consistently, and as explained above greatly simplifies the
transfer process.

The BCI has been debugged using a test-set of about 400 sentences (a selection of
which are shown in Appendix D), mainly taken from the hypothetical car-hire domain.
The current level of robustness of the prototype is that of a fairly good "demo" system;
median total processing times (analysis, transfer and generation) for short sentences in
the car-hire domain are around 10 seconds, which puts the BCI among the fastest of the
systems reported in the literature.

A short sample dialogue is shown on page 6: "Client" refers to the English-language
user, while "Agent" is his Swedish-language counterpart. Glosses for the Swedish text are
italics, and some of the less interesting disambiguation questions asked by the system have
been omitted for the sake of brevity.

---

3 This Vocabulary Expander tool, VEX, is described in Appendix C.
4 The QLF notation is fully defined in Appendix A.
<table>
<thead>
<tr>
<th></th>
<th>Client:</th>
<th>System to Agent:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>I want to hire a car.</td>
<td>&quot;Jag vill hyra en bil&quot;</td>
</tr>
<tr>
<td></td>
<td>(I want hire a car)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Agent: Skulle du vilja ha en Volvo?</td>
<td>What does &quot;en&quot; mean?</td>
</tr>
<tr>
<td></td>
<td>(Would you want have a Volvo?)</td>
<td>1 någon (some)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 precis en (one)</td>
</tr>
<tr>
<td></td>
<td>Enter a number:</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>System to Client:</td>
<td>&quot;Would you like to have a volvo&quot;</td>
</tr>
<tr>
<td>3</td>
<td>I don't like Volvos.</td>
<td>&quot;Jag tycker inte om Volvoar&quot;</td>
</tr>
<tr>
<td></td>
<td>(I think not of Volvos)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>System to Client:</td>
<td>&quot;Vi har en Saab och en liten Audi.&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;We have a saab and a small audi&quot;</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Can I pick up a Saab at the airport at 20:45?</td>
<td>Consider the following bracketings...</td>
</tr>
<tr>
<td></td>
<td>System to Client:</td>
<td>1: can i {pick up a saab} at the airport at 20:45</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: can i pick up {a saab at the airport} at 20:45</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enter the number of one you like, or minus the number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of one you don't like:</td>
</tr>
<tr>
<td></td>
<td>System to Agent:</td>
<td>&quot;Kan jag hämta en saab vid flygplatsen 20:45?&quot;</td>
</tr>
<tr>
<td></td>
<td>(Can I collect a Saab at the-airport 20:45?)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Agent: Jag tror att kontoret där stängner klockan 19:00.</td>
<td>&quot;I think the office there closes o'clock 19:00&quot;</td>
</tr>
<tr>
<td></td>
<td>(I think that the-office there closes o'clock 19:00)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>System to Client:</td>
<td>&quot;I think the office there is closing at 19:00&quot;</td>
</tr>
<tr>
<td>8</td>
<td>Client: OK, where is it?</td>
<td>Agent: Det går att hämta bilen från kontoret i staden</td>
</tr>
<tr>
<td></td>
<td>System to Agent:</td>
<td>(It goes to collect the-car from the-office in the-city.)</td>
</tr>
<tr>
<td></td>
<td>&quot;Okay var är den/det&quot;</td>
<td>&quot;It is possible to pick the car up from the office</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in the city&quot;</td>
</tr>
<tr>
<td></td>
<td>(OK where is it (common)/it (neuter)?)</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>System to Client:</td>
<td>&quot;Det är vid Slussen.&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;That is at Slussen&quot;</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Client: Thankyou!</td>
<td>System to Agent:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;Tack&quot;</td>
</tr>
</tbody>
</table>

A horizontal line indicates that processing of the previous message is complete.
It should be apparent that the BCI's output, while perhaps not perfectly idiomatic, is nevertheless of a good standard and is readily comprehensible. Note also that non-trivial translation is possible, as evidenced for example in the seventh box, where the expression *Det går att* (literally: “It goes to”) is rendered as *It is possible to*.

1.4 Structure of the report

The rest of the report is laid out as follows. In Section 2, we go into the choices underlying the BCI's design philosophy in more detail, in particular with regard to the choice of QLF as the transfer level. Following this, Section 3 describes the Swedish version of the CLE, and Section 4 the actual process of QLF transfer. In this part, we have concentrated on attempting to provide an objective evaluation of the effectiveness of the transfer formalism in terms of the practical experience gathered during the system's development and testing.

The following two sections deal with the less purely linguistic sides of the project. Section 5 describes the interactive disambiguation process, and Section 6 then discusses briefly our solution to the practical problems arising from being forced to divide development work between two groups in different countries. The final section sums up and gives pointers to possible further research. The appendices contain a BNF definition of the QLF formalism as used in the system, user manuals for the BCI and the VEX lexicon acquisition tool, examples of translations, and listings of the English and Swedish lexicons.
Chapter 2

Design philosophy of the system

In this chapter, we will describe in more detail the reasoning behind the design philosophy of the system. The basic idea is to perform translation by connecting together two copies of the CLE, one for each language; it follows that the fundamental design decision is to determine the level of representation at which the two systems will communicate. We have already indicated in an informal way why we chose this to be Quasi Logical Form (QLF). Here, we will expand further on this theme.

Basically, QLFs were selected as the appropriate level for transfer because they are far enough removed from surface linguistic form to provide the flexibility required by cross-linguistic differences. On the other hand, the linguistic, unification-based processing involved in creating them can be carried out efficiently and without the need to reason about the domain or context: the QLF language has constructs for explicit representation of contextually sensitive aspects of interpretation. Another aspect is the need to provide a clean interface to user disambiguation; when it is necessary, for correct translation, to resolve an ambiguity present at transfer level, the system interacts with the source language user to make the necessary decision, asking for a choice between word sense paraphrases or between alternative partial bracketings of the sentence. There is thus a strong connection between our choice of a representation sensitive to context and the use of interaction to resolve context dependent ambiguities.

We arrived at these general criteria partly by considering previous attempts to create general linguistically-motivated frameworks for transfer. We begin by noting that the representational structures on which transfer operates must contain information corresponding to several linguistic levels, including syntactic and semantic information. For transfer to be general, it must operate recursively on input representations. We call the level of representation on which this recursion operates the "organizing" level; semantic structure is the natural choice, since the basic requirement of translation is that it preserves meaning.

Experience has shown that syntactic phrase structure transfer, or deep-syntax transfer, results in complex transfer rules (e.g. Thurmaur 1990, Nagao and Tsujii 1986). McCord's (1988, 1989) organizing level appears to be that of surface syntax, with additional deep syntactic and semantic content attached to nodes. As we have argued, this level is not optimal, which may be related to the fact that McCord's system is explicitly not symmetrical: different grammars are used for the analysis and synthesis of the same language, which are viewed as quite different tasks. Isabelle and Macklovitch (1986) argue against such asymmetry between analysis and synthesis on the grounds that, although it is tempting as a short-cut to building a structure sufficiently well-specified for synthesis to take
place, asymmetry means that the transfer component must contain a lot of knowledge about the target language, with dire consequences for the modularity of the system and the reusability of different parts of it. In the BCI, however, the transfer rules contain only cross-linguistic (contrastive) knowledge, allowing analysis and generation to make use of exactly the same data.

Kaplan et al (1989) allow multiple levels of representation to take part in the transfer relation. However, Sadler et al (1990) point out that the particular approach to realizing this taken by Kaplan et al has problems of its own and does not cleanly separate monolingual from contrastive knowledge.

In contrast to systems such as Rosetta (Landsbergen, 1986) which depends on stating rule by rule correspondences between source and target grammars, we wish to make the monolingual descriptions as independent as possible from the task of translating between two languages. Apart from its attractions from a theoretical point of view, this has practical advantages in allowing grammars to be reused for different language pairs and for applications other than translation.

Both form (e.g. the form of referring noun phrases) and content need to be considered in designing a representation for transfer, but syntactic phrase structure trees are inappropriate because they are too closely related to the surface form of a source language: the transformations required for mapping between differing syntax trees result in complex transfer rules, and a general loss of compositionality. In contrast, QLFs appear to have excellent properties from this point of view. In Section 4.3.4 below, we consider this argument in more detail. The predicate-argument structure required for the application of sortal restrictions is also absent from syntactic analyses. Sortal restrictions can be very significant to translation because of the importance of word sense disambiguation to performing this task.

The arguments advanced so far make it clear that phrase-structure is unlikely to be the best transfer level in the context of the CLE; apart from QLF, there are also the possibilities of using LF (Logical Form) or RQLF (Resolved QLF) levels.\footnote{Readers unfamiliar with the distinctions between the various levels of representation in the CLE are referred to e.g. (Alshawi 1990).}

At LF level, sortal restrictions can be applied but at this level the form of noun phrase descriptions used and also information on topicalization is no longer present. Vagueness present in specifier phrases will also have been removed by an explicit commitment to a particular quantifier. It is also well-known that producing completely resolved interpretations can require arbitrary knowledge of the domain of discourse, knowledge which is usually not available to an automatic translation system.

This leaves us with the QLF and RQLF levels. Both these levels are deep enough to allow the application of sortal restrictions for word sense disambiguation. Both representations also contain noun phrase descriptions and syntactic information in the categories of QLF constructs. However, not all the information appearing in the RQLF about how QLF constructs have been resolved is necessary for translation. For example, while pronoun resolution is sometimes required for translation between language pairs with differing pronoun systems (especially with regard to gender), definite descriptions are often best translated into target definite descriptions rather than referents, since otherwise the view of the referent in the source is lost during translation. Similarly, translation from resolved ellipsis can result in unwieldy target sentences. Scoping and collective/distributive distinctions do not normally manifest themselves in paraphrases of RQLF interpretations, so ambiguities corresponding to these distinctions are often preserved during translation.
It would thus appear that, for many constructions, there is little advantage to be gained for the purpose of translation from the process of interpreting unresolved QLF constructs. For practical systems, aspiring to unrestricted domain translation, there might even be something to be lost by doing so: the lack of contextual knowledge and appropriate means for applying it mean that the interpretation process would be error prone. Contextual knowledge is available to humans in the machine-aided translation setting, so we are concentrating at present on systems in which humans can provide contextual resolution for the cases where this is required. The BCI application is well suited to this approach.

In arguing for QLF-level transfer, we are asserting that predicate-argument relations of the type used in QLF are the appropriate organizing level for compositional transfer, while not denying the need for syntactic information to ensure that, for example, topichood or the given/new distinction is preserved.
Chapter 3

The Swedish Core Language Engine

This chapter will describe the Swedish version of the CLE (hereafter S-CLE); in practice, the S-CLE grammar was not so much written from scratch as adapted from the English version, exploiting the large overlap between the structures of the two languages. We will begin in Section 3.1 by giving a brief overview of Swedish, concentrating on those aspects which turn out to have important consequences for Swedish-English transfer; in the following section, we describe in fair detail how the actual adaptation process was carried out. The only parts of the system we will examine in depth are morphology, syntax and semantics, and the lexicon. Processing beyond the level of generation of QLFs was not included in the project, but we have observed that the scoping and reference resolution modules continue to function for Swedish with at most very minor changes. The general level of functionality of the Swedish version is comparable to that of the English one, except that the lexicon is currently somewhat smaller (approximately 1000 entries,\footnote{This figure refers to the sizes of all the lexica taken together (cmf page 3).} or half as many as in the English system). The total development effort was about 16 man-months over a period of a year.

3.1 An overview of Swedish

Swedish is a Germanic language, spoken by about eight million people in Sweden and a half-million in Finland, where it is the official second language. It is related very closely to Danish and Norwegian, and somewhat less so to Dutch, English and German; the similarity to Norwegian is sufficiently great that the two languages are mutually inter-comprehensible. Except phonetically, it is generally regarded as a fairly easy language for people whose native tongue is English. Here, we will describe some of the more significant divergences between Swedish and English, not necessarily because they all are relevant to the task of translation as it is done in the BCI (our translation model allows the grammars to be developed independently), but partly to give a rough indication of the relative level of difficulty of this particular language-pair.
3.1.1 Morphology

An immediate and obvious difference is inflectional morphology, which in Swedish is very much more complex. Swedish allows inflection of verbs, nouns and adjectives: verbs have nearly the same inflections as their English equivalents, with two main exceptions. Firstly, there is an extra form (the supine) which is used to form the perfect tense, and which is generally distinct from the past participle; secondly, most inflectional forms have a "passive" counterpart, formed by adding the suffix -s. This is explained in more detail below. There are three conjugations of weak verbs (the second being divided into two sub-classes), and one conjugation of strong verbs, divided into three sub-classes. Nouns are marked for both number and definiteness; thus bil is "car", bilar is "cars", bilen is "the car" and bilarna is "the cars". They also have grammatical gender (common or neuter). There are five different declensions of nouns, corresponding to the plural endings -or, -ar, -er, -en and "null" (i.e. the singular and plural are the same). Adjectives are marked for number, gender and definiteness, but these are conflated so that there are actually only three distinct forms.

3.1.2 Syntax

Moving on to syntax, the most important differences in word-order stem from the strongly verb-second nature of Swedish: formation of both YN and WH-questions is by simple inversion of the subject and verb, without the introduction of an auxiliary. This is illustrated in the following examples:

Han såg Maria.  "He saw Mary."
Såg han Maria?  "Did he see Mary?"
Vem såg han?    "Who did he see?"

The same process is used to form topicalized clauses, which are extremely common, especially in spoken or colloquial language:

Maria såg han.  "Mary, he saw."
Idag såg han Maria.  "Today, he saw Mary"

Another significant difference at clause-level concerns negation, which is expressed with the particle inte ("not"); inte is placed after the main verb in a main clause, but before it in a subordinate clause, thus:

Han såg inte Maria.  "He did not see Mary."
Han sade att han inte såg Maria.  "He said that he did not see Mary."
Kvinnan som han inte såg.  "The woman that he did not see."

Similar considerations also apply to a number of other common adverbials (so-called "mobile adverbs"), including ofta ("often"), alltid ("always") and troligen ("probably").

The third important divergence at this level is formation of passives, which as already mentioned can be expressed by addition of the suffix -s to the corresponding active form.\footnote{There is also a rich system of derivational morphology, especially for forming compound nouns. Since the S-CLE has as yet no capacity to deal with this, we will not discuss it here.}

\footnote{With the exception of the present passive, which is formed from the imperative.}
Den skrevs av Maria. “It was written by Mary.”
(lit. “It wrote-PASSIVE by Mary.”)

Den ska skrivas av Maria. “It will be written by Mary.”
(lit. “It will write-INF-PASSIVE by Mary.”)

With regard to the NP structure, most of the differences derive from the fact, already mentioned above, that Swedish marks for definiteness. This means that the definite article can often be omitted, although it is obligatory before an adjective: thus “the car” is bilen, but “the red car” is den röda bilen. Prenominal adjectives agree in number, gender and definiteness with their nouns. There are a number of other divergences concerned with prenominal modification, but these are of a somewhat esoteric nature.

As with any language, there are a number of syntactic phenomena which are hard to fit into a systematic classification. Of these, we single out for special attention the very common vad ... för construction (which is analogous to the German was für):

Vad såg du för bil?. “What kind of car did you see?”
(lit. “What did you see for car?”)

A thorough treatment of Swedish morphology and syntax can be found in for example (Thorell 1973).

3.2 Adapting the CLE to Swedish

We begin by indicating roughly how the work performed in the adaptation was divided between the various parts of the system: for each module, Table 3.1 shows in the first column an estimation of the number of person-months devoted, and in the second and third the overlap between the Swedish and English sets of rules and rule interpreters, respectively, expressed as percentages.4

<table>
<thead>
<tr>
<th>Module</th>
<th>Effort (months)</th>
<th>Overlap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rules</td>
<td>Interpreter</td>
</tr>
<tr>
<td>Segmentation</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Morphology</td>
<td>2</td>
<td>25</td>
</tr>
<tr>
<td>Lexicon</td>
<td>3</td>
<td>70</td>
</tr>
<tr>
<td>VEX</td>
<td>4</td>
<td>60</td>
</tr>
<tr>
<td>Syntax</td>
<td>3</td>
<td>80</td>
</tr>
<tr>
<td>Semantics</td>
<td>1</td>
<td>90</td>
</tr>
<tr>
<td>Generation</td>
<td>1</td>
<td>90</td>
</tr>
</tbody>
</table>

These figures do not include effort expended on repairing software problems judged to be unrelated to Swedish.

In some cases, the basic formalism was generalized to some degree, necessitating corresponding extensions to the rule interpreters; these are noted in the fourth column, together with other comments. In general, it is clear that the amount of work needed to adapt the various CLE modules to Swedish declined steadily as a function of their “distance” to

4For the purposes of the present discussion, we will use the term “interpreter” indiscriminately to refer both to interpreters and compilers.
surface structure. It seems likely that this would be the case with adaptation to other European languages.

In the remainder of the section, we will discuss some specific technical problems, and their solution within the S-CLE.

3.2.1 Morphology

In the English morphology rules, affixes are always literals whereas in Swedish a compact grammatical description requires giving affixes categories with feature specifications.

There are basically two tricks, which suitably applied can be used to capture all the morphological structure fairly compactly; we illustrate them first with one of the simpler examples, namely the rules that describe pluralization of nouns. Recall that Swedish nouns can be pluralized in several different ways, depending on their declension, but that singular and plural forms coincide for 5th declension nouns. The main rule takes care of the 1st, 2nd, 3rd and 4th declension nouns: simplified slightly, it is as follows:

\[
\text{morph(nbar_nbar_plural,} = \\
\text{ nbar:}[\text{agr=plur, synmorphc=M, def=n, subcat=S, ...} = \\
\text{ --}> = \\
\text{ [nbar:}[\text{agr=sing, synmorphc=M, def=n, subcat=S, ...}], = \\
\text{ 'PLURAL':[synmorphc=M]}]).
\]

The rule is supplemented by four lexical entries for the plural suffixes, one for each relevant declension; the number of the declension is encoded in the \text{synmorphc} ("syntactic morphological category") feature. The first two entries are thus

\[
\text{lex('or', 'PLURAL':[synmorphc=1]).} = \\
\text{lex('ar', 'PLURAL':[synmorphc=2])}. = \\
\]

An additional rule, however, is needed for the 5th declension nouns. Here we need an extra \text{NBAR} feature \text{nullmorphn}, which when set indicates that the plural affix is null. The null plural rule is consequently

\[
\text{morph(nbar_nbar_nullplural,} = \\
\text{ nbar:}[\text{agr=plur, synmorphc=M, nullmorphn=n,} = \\
\text{ def=n, subcat=S,...} = \\
\text{ --}> = \\
\text{ [nbar:}[\text{agr=sing, synmorphc=M, nullmorphn=y,} = \\
\text{ def=n, subcat=S,...]}]).
\]

The same basic mechanism is used to deal with verb and adjective morphology, although the details are considerably more complicated. There are two main problems. Firstly, to capture compactly the facts about conjugation of strong verbs it seems necessary to perform some derivations in two or more stages. Thus for example the verb \text{bryta} ("break") has supine \text{brutit}, common part participle \text{bruten}, neuter past participle \text{brutet} and plural/definite past participle \text{brutna}. In all four forms, the \text{y} in the stem has mutated into an \text{u}; it consequently makes sense first to derive a "supine stem" \text{brut}, from which each of the other forms is then produced. The grammar writer is still forced to specify explicitly that the supine stem is an irregular derivation, but the number of "irregular"
assertions is reduced from four to one, a major saving in view of the fact that Swedish has several hundred such verbs.\footnote{This could also have been done by explicitly including the "umlauting-series", which indicate which vowels can mutate into which; however, we did not choose to do so since these series are both very complicated and non-deterministic.}

The second problem is that null formations occur not just in a single inflected form as with nouns, but in at least three distinct ones: taking the imperative as the base form, we note that this can coincide with any of the infinitive (any first or third conjugation verb), the present (hyr, "hire"; kör, "drive"), or the past (kom, "came"; sov, "slept"). Thus the nullmorph feature (analogous, for verbs, to the nullmorph feature for nouns) needs to be able to take on the four distinct values inf, \((\text{fin}/\text{present})\), \((\text{fin}/\text{past})\) and n. Less obviously, there has to be a second feature (nonnullmorph), which is set appropriately to licence application of the normal affix-adding derivation rules, and which takes as value the set of all inflections except the one which is the value of nullmorph. Thus for example the entry for the present-tense verb affix -er is

\begin{verbatim}
lex('er',
   ['\text{\textbackslash V\text{-AFFIX}}':]
   [vform=(\text{fin}/\text{present}),]
   [invform=impera],
   [symmorph=(2\slash 4),]
   [nonnullmorph=<<(\text{fin}/\text{present}))]]).
\end{verbatim}

Although this solution leaves something to be desired in terms of elegance, it does not appear to be possible to improve on it substantially in a unification-grammar framework which lacks full feature-value negation.

3.2.2 The lexicon

We had anticipated a large overlap in the English and Swedish grammars. Perhaps somewhat surprisingly, it turned out that the amount of structure shared between between the English and Swedish lexicons was also very significant. This is most obviously apparent when considering function words, where the great majority of function words in one language have a clear counterpart in the other; in general, differences are often confined to variance in surface form and changes to the values of a small number of features. We illustrate this with examples of entries for a determiner (någon, the common-gender singular equivalent of some), a modal verb (måste, equivalent to must), and an interrogative pronoun (vem, equivalent to who).\footnote{In these and subsequent examples of rules from the S-CLE, the non-alphabetic characters \{, \} and | correspond to the Swedish letters "å", "ä" and "ö".}

\begin{verbatim}
lex(some,
   det:[type=norm, gaps=g(G,G,F,F), simple=y, lexical=y, of=_.])

lex('n)gon',
   det:[type=norm, gaps=g(G,G,F,F), simple=y, lexical=y, of=_,
   def=n, agr=(sing\slash common\slash 3)])
\end{verbatim}

\((\text{någon} \text{ is indefinite and common-gender})\).
lex(must,
    v: [mainv=n, subjform=S, mhdfl=n, gaps=Gaps
        vform=fin,
        subcat=[vp:[vform=inf, modifiable=_, mainv=_,
            gaps=Gaps, subjform=S, headfinal=_]]])

lex('m)ste',
    v: [mainv=n, subjform=S, mhdfl=n, gaps=Gaps
        vform=(fin/\present), subordinate=_, agr=Ag,
        subcat=[vp:[vform=inf, modifiable=_, mainv=_,
            gaps=Gaps, subjform=S, headfinal=_,
            agr=Ag, subordinate=y, lepxsivized=_]])

(Swedish modals need to pass the agr feature to their arguments).

lex(who,
    np: [simple=y, mass=n, determined=n, name=n, pron=y,
        wh=y, hascase=n, gaps=g(G, G, F, F),
        type=(q/\r), agr=3])

lex(vem,
    np: [simple=y, mass=n, determined=n, name=n, pron=y,
        wh=y, hascase=n, gaps=g(G, G, F, F),
        type=q, def=y, agr=(sing/\common/\3)])

(Vem is definite and common-gender. Unlike who, it must be singular, and cannot be used as a relative pronoun).

A count of the Swedish function word lexicon revealed that only 33 of the 401 entries had not been derived from a close English equivalent. Among these, the following deserve mention: the negation particle inte; the reflexive pronoun sig; the preposition hos (like French chez); the adjective sådan (like German solch); and the verbs finnas (existential to be), komma att (a future construction), lätta bili (refrain from), heta (like German heissen), det gör att (it is possible to), det gäller att (it is important to) and ligga and stå (existential verbs meaning roughly "to be somewhere").

With regard to content words, the main problem is that Swedish verbs, like French and German ones, can subcategorize for reflexive pronouns; for example, marry is gifta sig med (literally, "marry oneself with"), and decide is bestämma sig för ("decide oneself for"). This means that the number of verbal paradigms is approximately doubled, since most paradigms have a counterpart with an extra reflexive. The prototype entries for the reflexive variants are straightforward extensions of those for the normal verbs.

3.2.3 Syntax and semantics

We will now examine the syntactic phenomena described in Section 3.1 in more detail, in particular with reference to the structure of their representations at QLF level. Most of the differences that arise are in essence fairly trivial, and involve only minor extensions to the English grammar rules. To take a comparatively hard example: although the vad ... för construction mentioned at the end of Section 3.1.2 lacks any near English equivalent, it can
be dealt with by a straight-forward application of the constituent movement mechanism, letting an NP gap matching a fronted occurrence of vad be absorbed by a constituent consisting of the preposition för, followed by an NBAR. The syntactic rule and one of the semantic rules (there is a second one for plural NBARs) are as follows,

\[
\text{syn('np_f|r_nbar',}
\text{np:[agf=A, def=n, type=q, ...},
\text{gaps=g([npgap:[lexform=vad,gaptype=wh]|Gn],}
\text{Go,Fn,[g|Fo]])}
\text{-->
[p:[lexform='f|r'],}
\text{nbar:[agf=A, def=n, ...},
\text{gaps=g(Gn, Go, Fn, Fo)])].}
\]

\[
\text{sem('np_f|r_nbar', singular,}
\text{(qterm(<t=quant, n=sing, l=wh_type>,V,Nbar),}
\text{np:[quantform=quant, handle=V, ...},
\text{semGaps=((qterm(_._._),)|Sn,So}))}
\text{-->
['f|r',p:[pptype=subcategorised]],}
\text{(Nbar,nbar:[agf=sing, arg=V, ...},
\text{semGaps=(Sn,So)])].}
\]

Thus the QLF for Vad såg han för bil? ("What kind of car did he see?") is

\[
[\text{whq,}
\text{past,}
\text{[se_3p,}
\text{(qterm(<t=quant,n=sing,l=ex>,A,[entity,A]),}
\text{a_term(<t=ref,n=sing,l=han>,B,[and,[male,B],[personal,B]]),}
\text{qterm(<t=quant,n=sing,l=wh_type>,D,[bili,D])])].}
\]

It would be tedious to list explicitly all the small adjustments that have been made in the grammar, so we will restrict ourselves to a brief summary of their nature, before looking at the small number of interesting cases where the English and Swedish analyses exhibit non-trivial structural differences at QLF level.

Of the minor changes, the most obvious are agreement and word-order. In many rules, it is necessary to propagate the agf feature to ensure agreement in number and gender, and in the NP and NBAR rules the feature def must also be added to force agreement in definiteness. There are also a number of rules (for example, those dealing with dates and times), where the order of constituents is permuted, or particles added or omitted; in Swedish, correct phrasing for these expressions corresponds to the word-orders “2 January” and “o’clock two”. The grammar for definite noun-phrases, partitives and conditionals is also sufficiently different as to require new rules: definite noun-phrases can omit the article (mannen, “the man”), partitives require no particle (en liter bensin, “a litre of petrol”), and conditionals can be expressed by a word-order only present in archaic dialects of English (Går han, så går jag, literally “Goes he, so go I”). We hope that the reader will be happy to take our word that none of these give rise to more than superficial problems: we will consequently proceed to the more interesting questions raised by verb-second word-order and negation, which we discuss separately.
Verb-second word-order

We first consider the word-order problems deriving from the verb-second rule: essentially, the modifications to the English rules are the following.

- Any verb (not just auxiliaries) can be inverted.

- Any NP, PP, ADJP or ADVP (not just those whose wh feature is set) can be fronted by WH-movement.

- The rules for topicalized clauses are removed, since they are now subsumed by the movement rules used in English for WH-question and relative clause formation.

- Consequently, the top-level "sigma" rule for topicalized clauses expands to an S with the features for inversion and WH-movement set, thus:

\[
\text{syn} (\text{sigma\_topic, core,} \\
\text{sigma} \\
\text{-->} \\
[s: [\text{type=norm}, \text{inv=y}, \text{whmoved=y}, \text{subjcase=subj}, \\
\text{vform=fin}]])
\]

Although this analysis adequately describes the syntactic facts and produces plausible QLFs, a closer examination unfortunately reveals that things are not quite as good as one might hope; the QLFs produced by the Swedish and English grammars still contain some information originating in their divergent syntactic realizations. The problem can be exhibited in its simplest form in a YN question with a VP modifier, for example Did he sleep today?. In English, the past tense operator applies to the auxiliary did, and the tensed auxiliary applies to the VP sleep today. The QLF is thus schematically

\[
[y\text{nq,} \\
\text{past,} \\
\text{and,} \\
[sleep1, \langle \text{event\rangle, \langle he\rangle}, \\
\langle today\rangle]]
\]

In the corresponding Swedish sentence, Sov han idag?, the past tense is applied directly to the main verb, sova, to which the VP operator idag subsequently applies. This gives the QLF

\[
[y\text{nq,} \\
\text{and,} \\
\text{past,} \\
[sova1, \langle \text{event\rangle, \langle han\rangle}], \\
\langle idag\rangle]]
\]

In other words: the tense operator has wide scope over the VP operator in English, but narrow scope in Swedish.
Negation

Similar, but more complex, considerations apply to the treatment of Swedish negation. In view of the data presented in Section 3.1 above, and for other reasons, it seems fairly clear that the negation particle inte should be regarded as forming a constituent with its verb; to ensure that the word-order constraints are enforced, there is a feature subordinate, which has the value n for main clauses and y for others. This feature is percolated down to the V, where it is available to rules like the following one:

\[
\text{syn}(\text{v
ot
ot_v, core,}
\text{v: [subordinate=y, fronted=n, agr=Ag, vform=Fm, mainv=M, modifiable=N, gaps=G, subcat=Su, ...]}
\text{--->}
\text{[neg: []},
\text{v: [subordinate=y, fronted=n, agr=Ag, vform=Fm, mainv=M, modifiable=N, gaps=G, subcat=Su, ...]]}).
\]

Since negation in English is viewed as applying to a VP rather than a V, the relative scopes of operators representing negation, tense and VP modification are correspondingly affected. For example, in the English sentence *He did not sleep*, not is considered to modify the VP sleep, producing a VP which subsequently appears as an argument to the tensed auxiliary did. The QLF is accordingly

\[
\text{[past,}
\text{[not,}
\text{[sleep1, <event>, <he>]]]
\]

The Swedish counterpart, *Han sov inte* has the tensed main verb sov modified by the negation inte, giving a QLF with the scopes of the tense and negation operators reversed:

\[
\text{[not,}
\text{[past,}
\text{[sova_2p, <event>, <han>]]]
\]

In the Sections 4.2.4 and 4.3.4, we will describe more exactly how the problems just discussed affect the process of QLF transfer.
Chapter 4

Quasi Logical Form transfer

In this chapter, we will take a detailed look at the process of Quasi Logical Form transfer. In the first section, we will describe the formalism used and its intended semantics. The second section describes the appearance of the transfer rules in the current version of the BCI, and the third section concentrates on evaluating the adequacy of the transfer formalism in terms of the experience gathered during the course of the project.

4.1 The transfer formalism

QLF transfer involves taking a QLF analysis of a source sentence, say $QLF_s$, and deriving from it another expression, $QLF_t$, from which it is possible to generate a sentence in the target language. Leaving aside unresolved referential expressions, the main difference between $QLF_s$ and $QLF_t$ is that they will contain constants, particularly predicate constants, that originate in word sense entries from the lexicons of the respective languages. If more than one candidate source language QLF exists, the appropriate one is selected by presenting the user with choices of word sense paraphrases and of bracketings relating to differences in the syntactic analyses from which the QLFs were derived. This interaction is described in Section 5. Performing transfer at a level as deep as QLF highlights the fact that many, perhaps most, difficulties in translation do not arise from differences between the constructions used in two languages under consideration but from differences between the mapping of concepts to such constructions; it is perfectly possible for non-trivial problems to occur when translating between two dialects with exactly the same set of syntax rules, or indeed when paraphrasing within the same language. For example, translating I want him to go as Jag vill att han ska åka (literally, I want that he will go) is at QLF-level essentially isomorphic to paraphrasing I expect him to go as I expect that he will go.

We now descend to concrete implementation details. A transfer rule specifies a pair of QLF patterns. The left hand side matches QLF expressions for one language and the right hand side matches those for the other:

\[
\text{rule}(<QLF \text{ pattern 1}>) \\
\quad <\text{Operator}> \\
\quad <QLF \text{ pattern 2}>).
\]

The QLF patterns in these rules can be QLF (sub)expressions, or such expressions with ‘transfer variable’s showing the correspondence between the two sides, as explained later on. If the operator is $\leftrightarrow$ this states that the rule is bidirectional, otherwise a single direction of applicability is indicated by use of one of the operators $\Rightarrow$ or $\Leftarrow$. 

23
As will be described below (in Section 4.3.2), most transfer rules are simple bidirectional relations between two atoms, like the following English-Swedish examples:

\[
\text{trule(car1} \leftrightarrow \text{bili1).}
\]
\[
\text{trule(pai1} \leftrightarrow \text{betalai1).}
\]

Transfer rules often correspond directly to interlingual meaning postulates: when the expressions in a transfer rule are formulae, the symbols \(\leftrightarrow\), \(\Rightarrow\), and \(\Leftarrow\) can be read as the logical operators \(\rightarrow\), \(\rightarrow\), and \(\leftarrow\) respectively. A more complicated rule like

\[
\text{trule}\left([\text{and}, [\text{badi1}, X], [\text{luck1}, X]] \leftrightarrow [\text{otur1}, X]\right)
\]

translating between the English \textit{bad luck} and the Swedish \textit{otur}, can thus, as well as the examples above, be expressed as postulates in a logical language for which the set of predicate constants is the union of the set of such constants in QLF expressions for both languages:

\[
\forall X(cari(X) \leftrightarrow bili(X))
\]
\[
\forall ESYSO(pai1(E,S,0) \leftrightarrow betalai1(E,S,0))
\]
\[
\forall X(badi1(X) \& luck1(X) \leftrightarrow otur1(X)).
\]

Transfer rules are applied recursively, this process following the recursive structure of the expression tree for the source QLF. In order to allow transfer between structurally different QLFs, rules with transfer variables need to be used. These variables, which take the form \(\text{tr}(\text{atom})\), show how subexpressions in the source QLF correspond to subexpressions translating them in the target QLF. Transfer variables may appear more than once on either side of a transfer rule. For example, the following rule expresses an equivalence between the English \textit{to be called} (\textit{I am called John}), and the Swedish \textit{heta} (\textit{Jag heter John}):

\[
\text{trule}\left([\text{call_name}, \begin{array}{l}
\text{tr(ev),} \\
\text{qterm}(xt=\text{quant}, n=\text{sing}, l=\text{ex}, A, [\text{entity}, A]),} \\
\text{tr(ag),} \\
\text{tr(name)]} \\
\end{array} \leftrightarrow \begin{array}{l}
\text{heta1, tr(ev), tr(ag), tr(name)].} \\
\end{array}\right).
\]

Transfer rules are not restricted to the logical elements of QLF, but are also used to indicate the correspondence between unresolved QLF expressions containing QLF categories. For example, the \textit{a_term} for a pronoun may be transferred into an \textit{a_term} with a category having linguistic features that are specific to the target language. Transfer between \textit{qterms} for descriptive noun phrases also requires translation of the categories and the restrictions:

\[
\text{trule}\left(qterm(tr(cat), X, tr(rest)) \begin{array}{l}
\leftrightarrow \\
qterm(tr(cat), X, tr(rest)).
\end{array}\right).
\]

Other transfer rules for categories would then indicate the correspondence between definiteness, number and gender in the two languages.

Formally, we can regard a \textit{qterm} for a referential definite description, as a function from (linguistic and non-linguistic) contexts to referents. The linguistic featural information in
these terms is an important part of the specification of such a function. A bidirectional transfer rule between two referential terms can thus be regarded as a meaning postulate stating equality of two functions, \( f_1 \) and \( f_2 \):

\[
\forall C \forall X (f_1(C)=X \leftrightarrow f_2(C)=X)
\]

where \( C \) ranges over contexts and \( X \) over referents. Similarly, a unidirectional transfer rule, corresponds to an implication in one direction:

\[
\forall C \forall X (f_1(C)=X \rightarrow f_2(C)=X)
\]

so, in this case, it is not possible to infer from the fact that the unresolved expression corresponding to \( f_2 \) refers to \( X \) in a given context, that the one corresponding to \( f_1 \) will also refer to \( X \) in that context.

At present, the transfer formalism is not completely monotonic: there is a 'cut' in the transfer rule interpreter which disallows compositional transfer of a complex expression matched by the relevant side of a transfer rule. This was done partly for efficiency reasons, and partly on the intuitive assumption that compositional transfer might anyway be inappropriate in these circumstances. However, our experience has indicated that this was probably an unwise decision, and it is likely that a removal of the cut, and a consequent transformation of the framework to a purely monotonic one, would be desirable. (See also Section 4.3.3). We have not done this mainly for lack of available time to carry out the necessary testing and debugging.

### 4.2 Types of transfer rule

In Section 4.1 above, we described the transfer framework at an abstract level. In this section, we will give a more detailed picture of what the rules actually look like in practice.

At the moment, the transfer component contains 711 rules. It is helpful to divide these up into groups: on examination, the most relevant property to characterize by seems to be the extent to which the rule is *lexical* in nature. We adopt the following classification:

- **Identity**: Rules whose left- and right-hand sides are identical atomic expressions. An identity rule in effect declares an atom to be language-independent.

- **Atomic lexical**: Rules whose left- and right-hand sides are distinct atomic expressions. An atomic lexical rule maps a logical constant in one language into a different logical constant in the other.

- **Non-atomic lexical**: Rules whose left- and right-hand sides are distinct non-atomic expressions related to specific lexical items. A non-atomic lexical rule maps a piece of structure associated with a lexical item in one language to a corresponding piece of structure in the other.

- **Structural**: Rules whose left- and right-hand sides are distinct non-atomic expressions, neither of which is related to a distinct lexical item. A structural rule describes a general transformation of the QLF.

- **Discard**: Rules which remove structure from one side that cannot effectively be turned into anything meaningful on the other side. Discard rules are basically stopgaps which attempt to recover from common situations that cannot be handled correctly.
The number of rules of each type are shown in Table 4.1. The higher up a rule is in this table, the better; one measure of the success of the system is the low proportion of rules that fall into the last two classes. We now discuss each class individually.

<table>
<thead>
<tr>
<th>Rule type</th>
<th>Number</th>
<th>Percent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity</td>
<td>51</td>
<td>7.2</td>
<td>Identical atomic expressions</td>
</tr>
<tr>
<td>Atomic lexical</td>
<td>526</td>
<td>74.0</td>
<td>Both right- and left-hand sides atomic</td>
</tr>
<tr>
<td>Simple</td>
<td>452</td>
<td>63.6</td>
<td>Single word sense to single word sense</td>
</tr>
<tr>
<td>Multi</td>
<td>74</td>
<td>10.4</td>
<td>Atomic constants representing complex constructions</td>
</tr>
<tr>
<td>Non-atomic lexical</td>
<td>100</td>
<td>10.9</td>
<td>Non-atomic but still lexical in character</td>
</tr>
<tr>
<td>Structural</td>
<td>25</td>
<td>3.5</td>
<td>General structural QLF-transformations</td>
</tr>
<tr>
<td>Discard</td>
<td>9</td>
<td>1.3</td>
<td>Discarding 9 Swedish interjections</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>711</strong></td>
<td><strong>100.0</strong></td>
<td>Number of rules in translation component</td>
</tr>
</tbody>
</table>

### 4.2.1 Identity rules

Identity rules describe the extent to which the basic framework of the QLF-representation language is shared between the two grammars; syntactically, the form of a rule of this type is

\[
\text{true(<Atom> <=> <Atom>).}
\]

As should have been apparent from the discussion in Section 3, the overlap is in fact very substantial: as a result, the identity rules cover nearly all the basic constructs used in QLFs.

The following atoms are declared as being common: the “structural” functors . and \(\Box\) (list-constructors), qterm and term_coord (quantified terms), and, and1, impl, not, apply, sub and ~ (logical constants) and fact, a_index and island (miscellaneous); the sentential operators ymq and whq; the quantifiers ex, exs, wh and all; the numerical operators used in forming determiners, eq, lt and gt; the atoms pro, ref, sing, plur, bare, refl, name and time, used in referential terms; and the basic predicates event, state, entity, personal, impersonal, quantity, male and female.

### 4.2.2 Atomic lexical rules

These rules, which have the syntactic form

\[
\text{true(<Atom1> <=> <Atom2>).}
\]

in fact constitute the vast majority of those in the transfer rule set. Of the 526 atomic lexical rules, 452 transfer senses of single words to senses of single words. These words can be proper names (e.g. Gothenburg <=> Gteborg), nouns (question_noun <=> frgal), adjectives (hard_not_soft <=> hrd1), verbs (hire_3p <=> hyra_3p) or interjections (hello <=> hej). At risk of repeating ourselves unnecessarily, we point out again that transfer is of word-senses, not of words.
In the remaining 74 rules in this category, at least one side expresses the sense of a multi-word phrase, most commonly a verb taking a complex complement involving particles or reflexives. Thus for example the rule manage_to_do <=> lyckas_gåra defines an equivalence between manage to ("I managed to find a car") and lyckas ("Jag lyckades hitta en bil"). Atomic lexical rules of this second kind are interesting, since it is by no means clear that they would have such a simple form in other transfer formalisms; since they are also quite common (making up more than 10% of the current set), it is reasonable to suppose that they represent a very considerable saving in effort on the part of the transfer-rule writer.

4.2.3 Non-atomic lexical rules

Moving progressively further in the direction of proper structural transformations, there is the class of rules where at least one side consists of a non-atomic structure, but where the rule is still clearly tied to a specific lexical item. (In practice, it actually seems very rare for one side to be atomic and the other non-atomic; the current rule set contains no examples of such rules). Rules of this kind can arise for two main reasons, and we discuss each sub-class separately. In the current version of the BCI, the two sub-classes appear to be about equally large.

Firstly, there are what one might call the "genuine" non-atomic rules, which reflect cases where two languages simply express the same concept in widely differing ways. The conceptually simplest case is that of a noun in one language which must be translated into a composition of noun and adjective in the other. For instance, the Swedish word mormor cannot be rendered more compactly in English than as the phrase maternal grandmother. The rule expressing this is

\[
\text{true}([\text{and}, [\text{grandmother\_person}, X], [\text{maternal1}, X]] \iff [\text{mormor1}, X]).
\]

As pointed out in Section 4.1, rules like this one also have a natural interpretation as meaning postulates. However, our experience is that the common cases (making up about three-quarters of the "genuinely" complex rules) derive from verbs, which either map to different verbs, or to combinations of be and a predicative adjectival phrase. There seem to be a large number of possible reasons for needing a complex rule to translate a verb; the ones in Table 4.2 can all be found in rules taken from the current set.

<table>
<thead>
<tr>
<th>Complex transfer type</th>
<th>English-Swedish example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Different particles</td>
<td>John likes Mary</td>
</tr>
<tr>
<td>Passive to active</td>
<td>Insurance is included</td>
</tr>
<tr>
<td>Verb to adjective</td>
<td>John owes Mary $20</td>
</tr>
<tr>
<td>Support verb to normal verb</td>
<td>John had an accident</td>
</tr>
<tr>
<td>Single verb to phrase</td>
<td>John wants a car</td>
</tr>
<tr>
<td>Idiomatic use of PP</td>
<td>John is in a hurry</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
As is apparent from our choice of name, we feel that "genuinely" complex transfer rules are inescapable given the basic nature of the system architecture, and simply reflect contrastive facts about the language-pair. However, there is a second class of complex lexical rule, where the motivation for introducing complex expressions is rather to compensate for shortcomings in the grammars by taking the transfer context into account; rules of this kind thus say in effect that a constant should be transferred in a particular way only if it occurs in the given context. If such a rule is correct, it is so because the meaning of the constant is dependent on its context, although this is not recorded explicitly in the QLF.

At present, there are about 30 rules of this type in the transfer component. Despite this, examination reveals no compelling reasons for believing that most of these are necessary, and we think it likely that the majority reflect nothing more interesting than sloppy coding on the part of the writers of transfer rules. It would clearly be very interesting to determine experimentally whether this was the case by attempting to recode the offending cases, and we hope to be able to present the results of such experiments at a later date.

4.2.4 Structural rules

As previously indicated, the final two categories of rules are the undesirable ones. "Structural" rules are rules that effect general transformations of the QLF structure, and are not tied to any specific lexical item. The current set contains 25 structural rules, which divide into three sub-classes.

The first sub-class contains two rules whose purpose is to remove irrelevant structure from the source-language QLF; there is one such rule for the "anaphora-list" present in a_terms, which holds a set of potential anaphoric antecedents, and one for the part of the QLF structure in the representation of PP's which holds the syntactic form of the preposition. The actual appearance of the rules is the following:

\[
\text{true}(l(.)) \leftrightarrow l(\_)
\]

\[
\text{true}(p(.)) \leftrightarrow p(\_)
\]

The point is to declare that the piece of structure contained in the l or p operator will match any corresponding piece of structure in the other language. The need for such rules could be removed if the grammars were rewritten so as to produce QLFs without this information, which is not needed for the translation task; however, since it can be useful for other applications, it was deemed simpler to introduce the above rules, which are no more than a minor irritation. It is likely that this type of solution would be the normal one in any translation system built, like the BCI, by reconfiguring general-purpose NLP software.

The second and third sub-classes, unfortunately, are not so harmless, and reveal concrete shortcomings in the system; they derive respectively from the problems with word-order and negation described in Section 3 above. As described there, the peculiarities of English auxiliaries and Swedish negation can give rise to QLFs with differing scope-orders of operators; consequently, two sets of "structural" transfer rules must be introduced to re-adjust scopes during the transfer process. There are at present 8 rules which together handle the word-order/auxiliary problem, and 15 for the negation one; there is one basic rule of each kind, with variants for different tense-operators and numbers of modifiers.\(^1\)

\(^1\)At present, the rules handle no more than two verbal modifiers.
A typical example of a word-order rule is the following, which deals with the case of past tense and one modifier:

\[
\text{rule}([\text{pres}, [\text{and}, \text{tr(body)}, \text{tr(mod)}]]) \leftrightarrow \\
[\text{and}, [\text{present}, \text{tr(body)}], \text{tr(mod)}]).
\]

This is for example the rule that would be used when translating *Did you go there?* to *Åkte du dit?* (literally: "Went you there?"). The effect of the rule, as can be seen, is to enforce the constraint that the tense operator has wide scope in the English, and narrow scope in the Swedish, and the other rules in this class perform similar functions.

The rule in the “negation” sub-class invert scope-order as well as just attaching modifiers higher. Thus for example the corresponding rule for the same case (past tense, one modifier) is

\[
\text{rule}([\text{past}, [\text{not}, [\text{and}, \text{tr(body)}, \text{tr(mod)}]]]) \leftrightarrow \\
[\text{and}, [\text{not}, [\text{past}, \text{tr(body)}]], \text{tr(mod)}]).
\]

Here, an appropriate sentence would be *I didn't go there*, which becomes *Jag åkte inte dit* (literally: "I went not there"). There are more negation rules than word-order/auxiliary ones because the negation-inversion phenomenon occurs with all of the simple past, present and progressive tenses, and also with “to be” and a number of modals.

### 4.2.5 Discard rules

Finally, we mention briefly the small class of “discard” rules. These rules constitute a temporary solution to the problem of translating certain common Swedish particles: as the name indicates, the information conveyed by them is at present simply thrown away. The most important words in this class are *nog*, *väl* and *ju*, which are syntactically adverbials, but function primarily as “hedges” or “intensifiers”. Thus for example *Han är nog här* (“He is nog here”) means something like “I guess he is here” or “He is probably here”, but *Du får nog åka då* (“You must nog travel then) has more the flavour of “Well, you’ll just have to travel then”. The basic information conveyed by *nog* is approximately that the speaker wishes not to take too much responsibility for what he is saying;\(^2\) *ju* and *väl* are similarly vague, and are normally used to say that the speaker regards something as “clear” or “obvious”, despite the fact that the listener may not accept it to be so. Thus for example *Det sa du ju nyss* (“That, you said ju just now”) means something like “That's what you just said, wasn't it?”.

Words of this kind have no truth-conditional semantics that fits simply into the framework of QLFs, and are consequently hard to represent. Since they unfortunately also occur with high frequency in colloquial Swedish, it seems unwise simply to disallow them; finding a better solution than discarding them is an important, though challenging, topic for future development of the system.

\(^2\)This is similar to e.g. the Japanese *ne*. 
4.3 Effectiveness of QLF Transfer

The reader will already have gained some impression of the strengths and weaknesses of the QLF transfer method from the discussion in the previous section. Here we try to make a more systematic evaluation. We will pay particular attention to the following criteria:

- **Expressiveness**: Intuitively valid generalizations about transfer should be expressible in the transfer formalism.

- **Compositionality**: The fewer special cases arising from idiosyncratic interactions between rules, the better.

- **Simplicity**: The simpler the rules are, the better. Of course, simplicity is to some extent subjective; however, we think it fairly clear that the ideal rule is simply an equivalence between two terminal symbols, and that as many rules as possible should be of this form.

- **Reversibility**: Reversible rules are better than unidirectional ones.

- **Monotonicity**: Adding new rules should not invalidate old ones.

Most of these are not easy to measure objectively, if they are not absolute properties of the formalism. The big exception seems to be compositionality; in Section 4.3.4 we will describe a method for measuring compositionality, and present fairly detailed test results. First, however, we present what facts we can on the remaining criteria.

4.3.1 Expressiveness

Since we are intentionally limiting ourselves by disallowing access to full syntactic information (but only that placed in QLF categories) in the transfer phase, it is legitimate to wonder whether the formalism can really be sufficiently expressive. Here, we will attempt to answer this criticism; we begin by noting that shortcomings in this area can be of several distinct kinds. Sometimes, a formalism can appear to make it necessary to write many rules, where one feels intuitively that one should be enough; we treat this kind of problem under the heading of compositionality (see Section 4.3.4). In other cases, the difficulty is rather that there does not appear to be any way of expressing the rule at all in terms of the given formalism. In our case, a fair proportion of problems that at first seem to fall into this category can be eliminated by having adequate monolingual grammars and using the target grammar as a filter; the idea is to allow the transfer component to produce unacceptable QLFs which are filtered out, during generation, by fully constrained target grammars.

A good example of the use of this technique is the English definite article, which in Swedish can be translated as an article (*den* or *det* depending on gender), but preferably is omitted; however, an article is obligatory before an adjective. Solving this problem at transfer level is not possible, since the transfer component has no way of knowing that a piece of logical form will be realized as an adjective; there are many cases where an adjective-noun combination in English is best translated as a compound noun in Swedish. Exploiting the fact that the relevant constraint is present in the Swedish grammar, however, the “transfer-and-filter” method reduces the problem to two rules defining equivalences of qterm categories, namely
true(<t=def,n=sing,l=the> <=> <t=def,n=sing,l=bare>)
true(<t=def,n=sing,l=the> <=> <t=def,n=sing,l=den>)

The intrinsic limitation of the "transfer-and-filter" method is that there are still a certain number of cases where the requisite information is present neither in the source QLF nor in the target grammar: this can be caused by the presence of unresolved relations (Alshawi 1990) in the source QLF, or simply by deficiencies in the formalism which make it impossible to express certain distinctions. We present an example of each type of problem.

To illustrate the problem with unresolved relations we consider the rule which defines an equivalence between the English verb have and its Swedish counterpart ha. This is generally correct; however, in some exceptional cases like to have an accident, a different Swedish verb must be used. Although it is possible to ensure that the correct translation is always produced first, by suitably ordering the rules, there is no way to block the default rule altogether; since ha (like have) is associated at QLF level with an unresolved relation, sortal information cannot be used to filter out constructions where it takes an inappropriate object. Similar problems can occur with other unresolved relations, like those arising from noun-noun and genitive constructions, although they seem in the case of the present language-pair to be less frequent.

An example of the second case, where the problem arises due to limitations in the QLF formalism's representational adequacy, is the Swedish present tense: as in many other languages, this can have either a specific or a generic/habitual reading, the first corresponding to the English present progressive, and the second to the plain present. Thus for example Det regnar i Stockholm can be used to mean either It is raining in Stockholm (the specific reading) or It rains in Stockholm (the generic). Since both of these get the same QLF, it is not possible to do more than produce both translations and querying the target-language user as to which one is more appropriate (see Section 5).

4.3.2 Simplicity and reversibility

The most obvious way to evaluate simplicity is to give a count of the various categories of rule, and provide evidence that there is a substantial proportion of rules which are simple in our framework but would not necessarily be so in others.

Looking at the figures for breakdown of transfer rules by types given in Table 4.1 on page 26, the most telling figure would seem to be the 10.4% that fall into the category "atomic lexical", but which translate senses of multi-word phrases. Since these account for about a third of the rules that do not translate a sense of a single word to a sense of a single word, they are clearly an important class; moreover, they would in most systems based on syntactic transfer correspond to some sort of complex transfer rule. It thus seems reasonable to claim that there is at least one large class of rules where QLF transfer wins substantially on this criterion.

On the score of reversibility, we will once again count cases, as summarized in Table 4.3; here we find that over 90% of the rules are reversible, 2.4% work only in the English-Swedish direction, and 6.2% only in the Swedish-English direction. These also seem to be fairly good figures.

There is one class of irreversible rule which should be mentioned explicitly, as it currently constitutes a weakness in the transfer notation; it occurs in connection with the phenomenon of object raising, as for example when I want him to see it is translated as Jag vill att han ska se det, literally "I want that he will see it". The QLF for the English
Table 4.3: Directionality of transfer rules

<table>
<thead>
<tr>
<th>Directionality</th>
<th>Number</th>
<th>Percent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eng ↔ Swe</td>
<td>650</td>
<td>91.4</td>
<td>Fully reversible</td>
</tr>
<tr>
<td>Eng ⇒ Swe</td>
<td>17</td>
<td>2.4</td>
<td>English-Swedish direction only</td>
</tr>
<tr>
<td>Eng ⇐ Swe</td>
<td>44</td>
<td>6.2</td>
<td>Swedish-English direction only</td>
</tr>
<tr>
<td>Total</td>
<td>711</td>
<td>100.0</td>
<td>Number of rules in translation component</td>
</tr>
</tbody>
</table>

The sentence is schematically

\[
[pres,
 \ [want\_to\_do,
     \ <event>,
     \ "I",
     \ [apply,
       \ C\^-[see\_physically,<event>,C,"it"],
       \ "he"\]]]
\]

while its Swedish counterpart is

\[
[present,
 \ [vilja\_att,
     \ <event>,
     \ "Jag",
     \ [fact,
       \ [fut,
         \ [se\_3p,<event>,"han","det"]]]]]
\]

The transfer rule must substitute the representation of the object (here “he”) into the representation of the inner VP (here “to see it”), and has the following appearance:

\[
\text{trule}([\text{want\_to\_do}, \text{tr(evi)}, \text{tr(subj)},
    \ [\text{apply}, \text{tr(obj)}^{-} \text{tr(body)}, \text{tr(obj)}]] =
    [\text{vilja\_att}, \text{tr(evi)}, \text{tr(subj)},
    \ [\text{fact}, [\text{fut}, \text{tr(body)}]]]).
\]

Since it is impossible to reverse simply the process of variable substitution, the rule consequently works only in the English-to-Swedish direction.

It appears that a principled solution to this problem is actually not as hard as may at first appear, though we have not had time to implement it in the project. The transfer-rule compiler would have to treat function application specially; in the “application” direction, it would work in the same way as the rule above. In the opposite direction, it would non-deterministically construct a function-application from a form, by finding a sub-form and replacing it with a λ-bound variable; if a condition is placed on the abstracted element (in practice, that it is of a form that could be the QLF of an NP), the non-determinism involved need not be prohibitive. This would remove the only major gap in the current transfer formalism’s expressive power.
4.3.3 Monotonicity

As explained in Section 4.1, the transfer formalism fails to be monotonic in only one respect, namely that compositional transfer of a complex expression is disallowed when that expression is found to match a transfer rule. This can sometimes have unfortunate interactions with the “transfer-and-filter” method, since it may be the case that a piece of QLF happens to match a transfer rule that produces an inappropriate target-language counterpart, while compositional transfer would have succeeded. For example, the rule mentioned in Section 4.2.4 above,

\[
\text{true}([\text{pres}, \text{[and, tr(body), tr(mod)]}] \leftrightarrow \text{[and, [present, tr(body)], tr(mod)]}).
\]

is appropriate in the Swedish to English direction when applied to questions and other sentences with inverted word-order; when applied to a declarative sentence, on the other hand, compositional transfer is correct since here the English QLF’s structure is the same as the Swedish one’s. At the moment, this must be dealt with by adding a second rule,

\[
\text{true}([\text{and, [pres, tr(body)], tr(mod)]} \leftrightarrow \text{[and, [present, tr(body)], tr(mod)]}).
\]

to cover the declarative case by explicitly allowing compositional transfer.

The presence of such rules is obviously not desirable (even though they are not particularly numerous), and it is our opinion that a removal of the relevant “cut” from the transfer interpreter, with consequent reversion to a completely monotonic framework, would probably improve the formalism. This should not be done, however, without first ascertaining that the resulting loss of efficiency is not serious.

4.3.4 Compositionality

Perhaps the most important factor in keeping transfer simple is the degree to which the transfer relation is a homomorphism, i.e. the degree to which transfer rules are compositional. For compositionality to be a meaningful notion in the first place, it must be possible for transfer rules to apply to partial structures. These structures can consequently occur in different contexts; other transfer rules will apply to the contexts as such. The question is the extent to which particular combinations of rules and contexts give rise to special problems. In a perfectly compositional system, this will never happen, although it seems a safe bet that no such system exists today. What we want is a method which objectively measures how closely we approach the compositional ideal.

As far as we know, there is no accepted benchmark for testing compositionality of transfer. Our first step in this direction has been the construction of compositionality tables, in which a set of rules and a set of contexts are systematically combined in all possible meaningful combinations. The question is the extent to which the complex transfer rules continue to function in the different contexts. In the following three diagrams, we give an example of such a table for the current version of the BCI. Table 4.4 gives a set of rules, which exemplify six common types of complex transfer. Table 4.5 gives a set of twelve common types of context in which the constructions referred to by the rules can occur. Finally, Table 4.6 summarizes the results of testing the various possible combinations.

To test transfer compositionality properly, it is not sufficient simply to note which rule/context combinations are handled correctly; after all, it is always possible to create a
### Table 4.4: Examples of complex transfer types

<table>
<thead>
<tr>
<th>Complex transfer type</th>
<th>English-Swedish example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Different particles</td>
<td>John likes Mary</td>
</tr>
<tr>
<td>Passive to active</td>
<td>Insurance is included</td>
</tr>
<tr>
<td>Verb to adjective</td>
<td>John owes Mary $20</td>
</tr>
<tr>
<td>Support verb to normal verb</td>
<td>John had an accident</td>
</tr>
<tr>
<td>Single verb to phrase</td>
<td>John wants a car</td>
</tr>
<tr>
<td>Idiomatic use of PP</td>
<td>John is in a hurry</td>
</tr>
</tbody>
</table>

### Table 4.5: Examples of transfer contexts

<table>
<thead>
<tr>
<th>Transfer context</th>
<th>English-Swedish example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Present tense</td>
<td>John likes Mary</td>
</tr>
<tr>
<td>Perfect tense</td>
<td>John has liked Mary</td>
</tr>
<tr>
<td>Negated</td>
<td>John doesn't like Mary</td>
</tr>
<tr>
<td>YN-question</td>
<td>Does John like Mary?</td>
</tr>
<tr>
<td>WH-question</td>
<td>Who does John like?</td>
</tr>
<tr>
<td>Passive</td>
<td>Mary was liked by John</td>
</tr>
<tr>
<td>Relative clause</td>
<td>The woman that John likes</td>
</tr>
<tr>
<td>Sentential complement</td>
<td>I know that John likes Mary</td>
</tr>
<tr>
<td>Embedded question</td>
<td>I know who John likes</td>
</tr>
<tr>
<td>VP modifier</td>
<td>John likes Mary today</td>
</tr>
<tr>
<td>Object raising</td>
<td>I want John to like Mary</td>
</tr>
<tr>
<td>Change of aspect</td>
<td>John stopped liking Mary</td>
</tr>
</tbody>
</table>

(lit.: “wants to have”)

(lit.: “has hurry”)

(lit.: “I want that John . . .”)

(lit.: “John slutade tycka om Mary”)

(lit.: “John stopped like-INF Mary”)

(lit.: “John tycker om Mary”)

(lit.: “John har tyckt om Mary”)

(lit.: “John tycker inte om Mary”)

(lit.: “Tycker John om Mary?”)

(lit.: “Vem tycker John om?”)

(lit.: “Mary blev omtyckt av John”)

(lit.: “Kvinnan som John tycker om”)

(lit.: “Jag vet att John tycker om Mary”)

(lit.: “Jag vet vem John tycker om”)

(lit.: “Jag tycker om Mary idag”)

(lit.: “Jag vill att John ska tycka om Mary”)

(lit.: “John slutade tycka om Mary”)

(lit.: “John stopped like-INF Mary”)

(lit.: “John stopped like-INF Mary”)
Table 4.6: A complete compositionality table

<table>
<thead>
<tr>
<th>Transfer context</th>
<th>Different particles</th>
<th>Active to passive</th>
<th>Verb to adjective</th>
<th>Support verb to normal verb</th>
<th>Single verb to phrase</th>
<th>Idiomatic use of PP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Present tense</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>Perfect tense</td>
<td>OK</td>
<td>generator</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>Negated</td>
<td>pres-not</td>
<td>pres-not</td>
<td>pres-not</td>
<td>past-not</td>
<td>pres-not</td>
<td>transfer</td>
</tr>
<tr>
<td></td>
<td>pres-not</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YN-question</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>WH-question</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>Passive</td>
<td>OK</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Relative clause</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>Sentential complement</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>Embedded question</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>VP modifier</td>
<td>OK</td>
<td>transfer</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>Change of aspect</td>
<td>OK</td>
<td>transfer</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>Object raising</td>
<td>transfer</td>
<td>transfer</td>
<td>transfer</td>
<td>transfer</td>
<td>transfer</td>
<td>transfer</td>
</tr>
</tbody>
</table>

Each square in Table 4.6 consists of two entries, the first for the Swedish to English direction, and the second for English to Swedish direction. The entries are to be interpreted as follows:

- • means that the combination is not applicable, i.e. that the construction referred to by the rule cannot occur in this context.
- • OK means that analysis, transfer and generation all functioned correctly, without extra rules being necessary to deal with the particular context.
- • generator means that the generator component was unable to produce the correct target language sentence.
- • transfer means that the transfer component was unable to produce a correct representation.
- • Other entries are names of rules needed to deal with special combinations of rule and context. For this table, only two extra rules were needed: pres-not, which reverses the relative scope of the operators for negation and the present tense and past-not, which performs a similar function for the past tense (see discussion in Section 3).
completely ad hoc solution by simply adding one transfer rule for each combination. The problem must rather be posed in the following terms: if there is a single rule for each complex transfer type, and a number of rules for each context, how many extra rules must be added to cover special combinations? It is this issue we will address.

The actual results of the tests were as follows. There were 136 meaningful combinations (some constructions could not be passivized); in 115 of these, transfer was perfectly compositional, and no extra rule was needed. For example, the English sentence for the combination “Verb to adjective + WH-question” is How much does John owe Mary, which has QLF

\[
[\text{whq},\n[\text{pres},\n[\text{owe\_have\_to\_pay},\n\text{qterm}(\langle t=\text{quant},n=\text{sing}\rangle,A,[\text{event},A]),\n\text{a\_term}(\langle t=\text{ref},p=\text{name}\rangle,B,[\text{name\_of},B,\text{john}]),\n\text{qterm}(\langle t=\text{quant},l=\text{wh}\rangle,C,[\text{quantity},C]),\n\text{a\_term}(\langle t=\text{ref},p=\text{name}\rangle,D,[\text{name\_of},D,\text{mary}])]]] \]

The corresponding Swedish sentence is Hur mycket är John skyldig Mary? ("How much is John indebted to Mary?") and its QLF is\(^3\)

\[
[\text{whq},\n[\text{present},\n[\text{vara},\n\text{qterm}(\langle t=\text{quant},n=\text{sing}\rangle,A,[\text{state},A]),\n[\text{skyldig\_ngn\_ngt},\n\text{a\_term}(\langle t=\text{ref},p=\text{name}\rangle,B,[\text{name\_of},B,\text{john}]),\n\text{a\_term}(\langle t=\text{ref},p=\text{name}\rangle,C,[\text{name\_of},C,\text{mary}]),\n\text{qterm}(\langle t=\text{quant},l=\text{wh}\rangle,D,[\text{quantity},D])]]] \]

It should be evident that the complex transfer rule defining the equivalence between owe and vara skyldig,

\[
\text{trule}([\text{owe\_have\_to\_pay},\n\text{qterm}(\langle t=\text{quant},n=\text{sing}\rangle,A,[\text{event},A]),\n\text{tr}(\text{ag}),\text{tr}(\text{sum}),\text{tr}(\text{obj})] \]
\[\Rightarrow \]

[\text{vara},\n\text{qterm}(\langle t=\text{quant},n=\text{sing}\rangle,A,[\text{state},A]),\n[\text{skyldig\_ngn\_ngt},\n\text{tr}(\text{ag}),\text{tr}(\text{obj}),\text{tr}(\text{sum})])].\]

is quite unaffected by being used in the context of a WH-question.

Of the remaining 21 rule/context/direction triples, seven failed for basically uninteresting reasons: the combination “Perfect tense + Passive-to-active” did not generate in English, and the six sentences with the object-raising rule all failed in the Swedish-English direction for the reasons described in Section 4.3.2. The final fourteen failures are significant from our point of view, and it is interesting to note that all of them resulted from mismatches in the scope of tense and negation operators.

\(^3\) är is the present tense of vara (“to be”).
The question now becomes that of ascertaining the generality of the extra rules that need to be added to solve these fourteen unwanted interactions. As explained in detail in Section 4.2.4, a small number of “structural” rules (two of which were relevant here), are sufficient to reorder the scopes of tense, negation and modifiers, and account for the scope differences between the English and Swedish QLFs arising from the general divergences in word-order and negation of main verbs. These solved ten of the outstanding cases. For example, the combination “Different particles + Negated” is *John doesn’t like Mary* in English and *John tycker inte om Mary* (“John thinks not about Mary”) in Swedish; the QLF-pair is:

\[
\text{[pres}, \\
\text{[not}, \\
\text{[like_sp}, \\
\text{qterm(<t=quant,n=sing>, A, [event, A]),} \\
\text{a_term(<t=ref,p=name>, B, [name_of, B, john]),} \\
\text{a_term(<t=ref,p=name>, B, [name_of, B, mary])]} \\
\text{]} \\
\text{[not}, \\
\text{[present}, \\
\text{[tycka_cm}, \\
\text{qterm(<t=quant,n=sing>, A, [event, A]),} \\
\text{a_term(<t=ref,p=name>, B, [name_of, B, john]),} \\
\text{a_term(<t=ref,p=name>, B, [name_of, B, mary])]} \\
\text{]}]
\]

The extra rule here is

\[
\text{rule([pres, [not, tr(body)]]) } \Rightarrow \\
\text{[not, [present, tr(body)]]}
\]

which reorders the scopes of the negation and present-tense operators.

The four bad interactions left all involved the English verb *to be*; these were the combinations “Passive to active + VP modifier” and “Idiomatic use of PP + negation”, which failed to transfer in either direction. Here, there is no general solution involving the addition of a small number of extra rules, since the problem is caused by an occurrence of *to be* on the English side that is not matched by an occurrence of the corresponding Swedish word on the other. The solution must rather be to add an extra rule for each complex transfer rule in the relevant class to cover the bad interaction. To solve the specific examples in the test set, two extra rules were thus required.

Summarizing the picture, the tests revealed that all bad interactions between the transfer rules and contexts shown here could be removed by adding four extra rules to cover the 124 possible interactions. In a general perspective, (viewing the rules as representatives of their respective classes), the rule-interaction problems exemplified by the concrete collisions were solved by adding

- 26 structural rules to cover the standard scope mismatches caused by verb-second word-order and negation.
- two extra rules (one for present, and one for past tense) for each complex transfer rule of either the “Idiomatic use of PP” or “Active to Passive” types, to cover idiosyncratic interactions of these with negation and VP-modification respectively.
We view these results as very promising: there were few bad interactions, and those that existed were of a regular nature that could be counteracted without fear of further unwelcome side-effects. This gives good grounds for hoping that the system could be scaled up to a practically useful size without suffering the usual fate of drowning in a sea of ad hoc fixes.

One may also note that all failures of compositionality occurring in Table 4.6, with the exception of the be-negation case, were caused by the same phenomenon, namely scope mismatches involving tense operators. It is thus of considerable significance that an alternate QLF representation has been developed where tense and aspect are represented, not as operators, but rather as aforms; we expect to use this representation in later versions of the BCI.
Chapter 5

Disambiguation and interaction

The linguistic information available to the BCI defines a mapping of source language sentences onto source language QLFs, another of source language QLFs onto target language ones, and a third of target language QLFs onto target language sentences. In general, a given input will map onto several values, and it will be necessary for the system to choose which value is appropriate at each stage.

\[ S\text{-}sentence \rightarrow S\text{-}QLF(s) \rightarrow \text{translation} \rightarrow T\text{-}QLF(s) \rightarrow \text{generation} \rightarrow T\text{-}sentence(s) \]

There are two observations to be made here. Firstly, there will be sentences for which the composition of the three mappings yields no values, i.e. which are untranslatable; so some means of recovery is desirable. This is discussed in Section 5.4 below. Secondly, a particular choice between values may be spurious in that each choice leads to the same, or a very similar, meaning being conveyed to the target language user. This occurs in different ways in all of analysis, transfer and generation.

Thus when faced with a choice at any stage, the system can take any of the following actions, which are discussed in turn.

- Choose one option arbitrarily, on the grounds that the choice makes little or no difference to the meaning of the end result.
- Decide that one of the options is intrinsically better, for example because it conforms more closely to sortal restrictions, or because (if a QLF) it contains more salient or more frequent word senses. Reasoning, if performed, also comes into this class of actions.
- Present one of the users with (paraphrases of) the choices, and perhaps ask for a decision.

5.1 Recognizing a choice as spurious

It turns out to be easier to recognize a choice as spurious or unnecessary the closer we are to generating the output.

The simplest case is generation. In generating from target language QLFs, spurious choices are the rule rather than the exception, because sentences corresponding to the same QLF will normally just be syntactic variants of one another. The BCI therefore presents the first sentence it generates from a QLF, only considering alternatives if the user asks for them.
In transfer, spurious choices can arise because QLFs are not canonical: different QLFs can have identical, or very similar, meanings. If the Swedish sentence *Jag äter gärna godis* is transferred to English QLFs for, say, *I like to eat sweets* and *I enjoy eating sweets*, the difference would not be important. Such choices can be much reduced by taking advantage of the directionality of transfer rules; when one term in language A corresponds to two alternatives in language B, but one of those alternatives will always in fact be acceptable, the transfer rule involving the other can be made unidirectional. Thus, for example, we have the following rules, where the Swedish *angående* and *beträffande* can both map onto the English *concerning*, but in the other direction *concerning* is always translated as *angående*:

\[
\text{true}(\text{concerning1} \leftrightarrow \text{angående1}).
\]
\[
\text{true}(\text{concerning1} \Leftarrow \text{beträffande1}).
\]

The same technique could be applied to structural alternatives, as well.

In analysis, a choice (this time between distinct meanings) can be spurious when source and target languages share an ambiguity; for example, the English *John drove the car without any insurance* and its Swedish translation *John köpte bilen utan försäkring* are both ambiguous with respect to whether the car itself, or John’s driving of it, is uninsured.

There are two ways in which one might recognize a spurious choice between analyses. One is by “look ahead”: for example, if two source language QLFs differ only in the sense selected for a given word, the difference is unimportant if the two senses transfer to senses of the same target language word. In the English CLE lexicon, the word *bank* is defined with senses corresponding to (among other things) banks as buildings and banks as organizations. This distinction can be important in other language processing applications, but the same Swedish word is used for both.

In general, one might attempt to establish whether any given choice of analyses is spurious by following each option through all the way to generating multiple target language sentences, and seeing whether any generated sentence occurred in both sets. However, this is likely to be too time consuming to be practically useful. It may also have other practical difficulties in an architecture where the BCI components for the two languages run on two machines connected over a wide-area network.

### 5.2 Selecting one option automatically

Non-spurious choices occur principally in analysis, but also to some extent in transfer, when the competing QLFs represent different meanings which cannot be expressed by the same sentence in the target language. In analysis, this is the familiar problem of linguistic ambiguity. In transfer, it will normally involve a lexical distinction made in the target language but not in the source language; the source language word cannot truly be called ambiguous, but nevertheless, in any given context, only one of the target words is correct. An example would be translating the English *grandfather* to the Swedish *farfar* (paternal grandfather) or *morfar* (maternal grandfather), there being no single Swedish translation of *grandfather*.

The fundamental choice of strategies here is between making the choice automatically and querying one of the users, although in practice a mixed approach has been adopted, the system doing what it can and consulting a user when it cannot decide on its own.

Automatic decision-making can be carried out:
• on context-independent, sentence-internal grounds, such as sortal restrictions on predicates;

• using general linguistic or domain-specific tendencies, such as frequencies of word senses or grammatical constructions;

• using reasoning about the relation between the analyses under consideration and the specific or general context, as in the CLE processing levels that (in other applications) follow on from QLF construction.

In the BCI, the first type of decision is currently made by the system during the analysis phase, and may optionally be applied to target language QLFs too. The second could in principle be made, while the third, in general, is dependent upon advances in the state of the art, or at least on the availability of domain-specific back-end systems.

The CLE’s procedure for checking sortal restrictions has been adapted in the BCI to work on a preferential basis: a QLF that disobeys sortal constraints is only rejected if there is another QLF that disobeys fewer. This is because, especially in a relatively unconstrained domain, unpredictable uses such as metaphor may be quite common. A literal translation is better than nothing, but should be received with caution. Thus if a translation is produced from a sortally imperfect QLF, the fact can be indicated by attaching a query to the output, for example:

User 1: The engine of this car has died.

System to user 2: ... possible translation violated sorts
Ignore sort violation? (y/n): y

"Den här bilens motor har dött."

It would be quite feasible to attach weights, possibly domain-specific, to particular word senses and grammar rules, so that in the absence of convincing sortal differences, the reading that resulted from the most frequent senses and rules would be preferred. The potential already exists to do this in the CLE, to some extent, by ordering rules and lexical entries. The CLE’s representational levels also allow the results of reasoning to be incorporated if suitable intelligent systems become available.

5.3 Querying one of the users

When the BCI is unable to decide by itself on a QLF in analysis or transfer, it needs either to query one of the users, or to proceed with all the options, producing alternative outputs. The strategy that has been adopted in the BCI project is as follows:

• The system interacts with the source user to decide on a single QLF to be transferred, i.e. to resolve any source language ambiguities. This is done by using bracketing to indicate possible structures of a sentence or presenting the user with paraphrases of the senses of an ambiguous word.

• One or more target language QLFs may be produced from the single approved source QLF. Where possible, senses are conflated as shown in the example below. A sentence is generated from the first QLF, and presented to the target user. If the user disapproves of the translation, other target QLFs may be tried.
• Sentences generated from the same QLF will be fairly close paraphrases of each other. For clarification, the target user may request a second paraphrase of any of the sentences presented to him.

• The target user may query the source user (via the BCI, of course) about any remaining ambiguities.

An exchange involving most of these stages might be as follows:

User 1: His poor grandfather is ill.

System to user 1: What does "poor" mean?
   1 penniless
   2 unfortunate
   3 unimpressive
Enter a number: 2

System to user 2: "Hans stackars farfar/morfar är sjuk."
   His unfortunate (father's father / mother's father) is ill.
   Accepteras översättningen? j
   Accept translation? y

User 2: Är det hans farfar?

System to user 1: "Is it his paternal grandfather?"

Figure 5.1: Example of BCI interaction

A choice between target language QLFs that is a matter of a single lexical item is finessed by defining what might be called 'pseudo-words' involving slashes. In this example, farfar/morfar is defined as a Swedish (pseudo-)lexical item whose predicate is associated with the predicate for the English grandfather. It is not necessary for the system to be told which of the options the target user prefers.

The reasons for the division of labour shown here between source user, target user and system are as follows.

It is more appropriate for the source user to choose between source language QLFs, firstly because only the source user can reliably be expected to know which meaning he intended, and secondly because users may feel happier if they know that the system will not generate anything for the other user that might derive from an unintended meaning.

Correspondingly, the target user is better placed to choose between target language QLFs (implicitly, by inspection of the resulting generated sentences) for the practical reason that to ask the source user to do this would mean having to transfer structures back into different source language QLFs, which would not always be possible, and in any case would be beset by almost all the linguistic problems facing the whole translation process. The strategy of presenting sentences generated from multiple QLFs should not swamp the target user with information because the transfer rules will have been written so as not to produce large numbers of results (as in the case of concerning on page 40 above).
This overall interaction strategy allows variants of the BCI's interactive use in which it can be used to encode speech or e-mail messages in QLF format, eliminating source language ambiguities, and decode on reaching the other user (see Sections 7.3 and 7.4).

If there are multiple source language ambiguities in a sentence, they are factored out, with the user's attention being focused on one distinction at a time. Structural ambiguity, corresponding to different syntactic bracketings of the sentence giving rise to different QLFs, is tackled first. For example, if the user inputs the sentence I telephoned the man from France, he will be asked to make the following choice:¹

Consider the following bracketings...

1: I {telephoned the man} from France
2: I telephoned {the man from France}

Enter the number of one that you like, or minus the number of one that you don't like:

If more than one such ambiguity is present, as in He drove the car to the garage without any insurance, several menu options, each still with only one pair of brackets, are presented. The user makes a decision, positively or negatively, about any one option, and the process repeats, with the numbers both of alternative QLFs and of menu options being reduced until only one possible full bracketing remains.²

The set of syntactic categories that give rise to bracketings that may be presented to users is configurable by the grammar writer. Currently, the categories s, np and vp are used on both the English and Swedish sides.

The full bracketing eventually selected may correspond to several QLFs, partly because of straightforward lexical ambiguity as with poor above, and partly because of structural differences at the QLF level that happen to give rise to the same bracketings of the syntactic categories deemed to be significant. The latter case is currently handled simply by making an arbitrary choice, but this could be improved on by defining rules that allowed competing QLFs to be paraphrased (using the system's generator) in ways that brought out the contrast between them.

5.4 Recovery

In the sections above, the problem was that there were a number of possible interpretations to choose from. But sometimes, the problem is the opposite: there are sentences for which the composition of the three mappings (analysis, translation and generation) yields no values. For these untranslatable sentences, some means of recovery is desirable.

In analysis the same means of recovery as in the CLE proper are available, i.e. spelling correction, proper name inference, and so on.

In transfer, we experimented with a pragmatic approach, in which, if no applicable rule could be found for a given construction, it would be translated as it stood. For non-lexical items, this was normally the correct choice, due to the similarities between English and Swedish — many constructions are simply the same in both languages. For lexical items,

¹For the case of English-Swedish, it is rare for a choice between source language bracketings to make a difference to the target language structure because the word order in the two languages is quite similar. However, decisions of this type are very important for other, less closely related language pairs.

²We have also experimented with a facility which allows the experienced user to specify the desired bracketing explicitly in the input sentence.
however, this approach was the correct one for proper names only; in general it led to a lot of undesirable target QLFs, for which generation was doomed to fail. We therefore removed this behaviour in favour of the more monotonic approach of trying to provide explicit transfer rules for all structures that should be translatable.

In generation, due to the adopted "transfer-and-filter" approach, the target language generator normally should fail to generate from some of the QLFs it is presented with. If it, however, fails to generate from all the QLFs given by the translation component, some graceful recovery should be available. At present, the BCI system only allows for recovery from sortal restriction violations, as described above (Section 5.2).
Chapter 6

Distributed development

The work described in this report was carried out over a one-year period by collaboration between two groups of two or three researchers, one group in Cambridge, England, and the other in Stockholm, Sweden. This kind of arrangement is very common in research projects, especially, by the nature of things, projects on Machine Translation. However, efficient exchange of both ideas and software between groups in different locations can be a problem in such situations. We therefore describe here the facilities and procedures which, we think, enabled us to work together nearly as effectively as if we had all been located in the same building, even though we visited each other’s laboratories only every few months.

A fast and, above all, reliable electronic mail link was an essential prerequisite. We estimate that we exchanged around two thousand e-mail messages during the course of the project. Most of the time, the delay was of the order of five minutes; this allowed questions to be asked and answered and software to be updated in not much less time than if we had been working in the same building. It was particularly useful to be able to splice system code and output directly into a message.

Consistency between the versions of the system at the two sites was maintained as follows. Each site was responsible for developing an agreed subset of the system’s code and data files; individuals at a site worked on files in their subset using operating system commands based on the $UNIX^{TM}$ “RCS” (Revision Control System)$^{1}$ utility to check files in and out of the development directory. A send command was written which, when invoked manually after development had reached a consistent state, would send to a process at the remote site all necessary updates from the development directory, also copying them to the export directory at the local site. The process at the remote site would put the files received into an import directory. A manually-invoked receive command would then copy new files from the import directory to a dual directory. Compilation of the system then involved accessing files in both the development directory and the dual directory.

Thus the contents of the export directory at one site were (barring communication or human errors) always identical to those of the import directory at the other, except when an update message was in transit. This allowed the send operation to send differential updates, not whole new copies, resulting in a considerable saving in communication costs and a probable increase in reliability. Each differential update was accompanied by an indication of the size, in bytes, of the result of the update; if the updated version of a file

---

$^{1}$RCS manages multiple revisions of text files and automates the storing, retrieval, logging, identification, and merging of revisions.
in the import directory did not have the required size, a message was sent to an operator at the import site. Such problems, which only occurred rarely, could always be sorted out by resending the whole file. The distinction between the import and dual directories prevented errors like these from affecting the compilation of the system, since the files in the dual directory would stay in a consistent and valid (though slightly outdated) state until a receive operation was (manually) initiated.²

²We also experimented with automatic updates of the files in the development directories, but did not find this reliable enough for our purposes.
Chapter 7

Summary and further directions

In the previous chapters, we have described the design philosophy and prototype implementation of the BCI. Here, we will briefly summarize the key ideas of the project, and then suggest possible continuations of our work aimed in the direction of construction of commercially interesting products.

7.1 Summary

The central theoretical idea in the project has been the use of transfer through Quasi Logical Form, which as argued at length provides an excellent compromise between the competing methodologies of transfer and interlingua-based Machine Translation. Another important novelty has been the "transfer-and-filter" method, which appears to provide a general way of simplifying the transfer component by making full use of the mono-lingual constraints present in the target-language grammar. On the software engineering side, we have successfully demonstrated the feasibility of two novel approaches: the construction of a large-scale Swedish grammar by adaptation of an existing English one, and the use of QLF to facilitate an efficient and conceptually well-founded interactive disambiguation method. We have also shown how "compositionality tables" can provide the basis for an objective evaluation of the success or otherwise of a transfer architecture.

We feel we have every right to claim that the project has been extremely successful, especially taking into account the limited time and resources available. We hope in particular that it is apparent that the detailed description of the shortcomings of the QLF transfer method presented in Section 4.2 is actually a strong advertisement for our approach; we are sufficiently confident of ourselves that we are able to list explicitly all the things that go wrong, and leave implicit the fact that everything else works. It is our opinion, based on a survey of the literature, that very few other projects of this kind would be able to subject their systems to the same kind of scrutiny as we have. In particular, we think that there is ample evidence to support our conclusion that this architecture should scale-up well to larger applications.

In the remaining sections of this chapter, we will describe five ways in which the existing BCI prototype could be extended into the basis for a commercial application. The first of these is simply a more advanced version of the existing BCI; the second is a variant on the basic idea, used to translate e-mail messages. The third is a speech-to-speech translation system built by equipping the BCI with a speech-analysis front-end and a speech-synthesis back-end. This is perhaps the most exciting of the possible ways in which the project can
be continued, and is by no means as impractical as one might immediately be inclined to suppose.

The last two ideas are more speculative: the fourth is a "multi-lingual workstation", which uses QLF transfer as a representation to facilitate information exchange between different sub-systems, including access in several languages to a knowledge-based application. Finally, we look at the possibility of a system for multi-lingual composition of business or technical texts.

### 7.2 A second-stage BCI

A more advanced version of the BCI could include either an improved coverage of the existing English-Swedish language pair, or the introduction of new languages; based on our experience in adapting the CLE to Swedish, we consider it likely that similar adaptations to most European languages could be carried out with comparable investments of effort.

The English-Swedish BCI could be improved in several ways. The prime determiner of performance is good monolingual coverage, since this is where most of the work gets done; the system design means that the transfer rules are easy to write, as long as the various grammars correctly encode the relationship between surface strings and their QLF representations. A tentative plan would probably include the following points:

- **Upgrading of grammar**: Tests made at the end of the project's first year indicate that certain parts of the grammar (in particular, the treatment of tense and aspect) could profitably be rewritten. Following this, the grammar would be improved by empirically determining shortcomings in its coverage and extending it appropriately.

- **Completion of Swedish core lexicon**: The core content-word lexicon should be expanded to give coverage equivalent to the English one. This involves adding something between 400 and 800 new entries.

- **Construction of domain lexicon**: If the BCI were to be used with a specific real application in mind, a large domain lexicon would need to be constructed. This would involve frequency analyses of real texts, and construction of 2000-5000 entries.

- **Interfacing to large machine-readable lexica**: Better fail-soft performance could be provided by giving the system facilities to allow it to access existing bilingual lexica when unknown words are encountered; experiments have already been carried out in which the English CLE has been interfaced to a monolingual dictionary, and a fair proportion of the software support needed for this task is consequently already available.

The result of such an effort would be a large-coverage general-purpose system which could easily be adapted to a variety of dialogue-based applications.

### 7.3 Interactive translation of electronic mail

A variant on the basic theme of the BCI, which may well be of greater immediate interest, would be to integrate it into an existing electronic mail system to allow interactive composition of messages intended for foreign-language addressees. Messages would include lists of source-language QLFs as well as the actual source-language text, these QLFs being
deciphered at the receiver end by target-language versions of the system. This configuration makes it possible to produce output in multiple languages transparently to the source-language user.

Our impression is that the quality of the output from the BCI is of a level well-suited to this kind of application. Although translation can be overly literal, or in other ways less than perfect, it is rare for it to be incomprehensible, or actually incorrect, in the sense of expressing something radically different from what the source-user intended. Consequently, both users can communicate through the system with a fair degree of confidence; this last point is of great practical importance, since it would normally be impossible either to check the translation or to post-edit it.

On the basis of our experience in the project so far we think it likely that an application of this kind, which was sufficiently robust to be of real use in practical contexts, could be created without doing more than systematically extending the system in the ways indicated in the previous section.

### 7.4 Speech-to-speech translation

Although automatic speech-to-speech translation was not long ago regarded as being well beyond the state of the art, it appears that progress in this area has been more rapid than was generally anticipated; the time-scales involved are certainly longer than for applications of the kinds described in the first two sections, but there is already talk about products appearing on the market by the beginning of the next decade. NEC recently announced a prototype system which could carry out automatic speaker-independent translation between English and Japanese with a 500 word vocabulary in a circumscribed domain, and it is well-known that ATR have made large research investments in similar systems. Reports of this work have sparked off general interest in the field, not least of course due to the enormous commercial opportunities it presents.

Our impression is that it would be quite feasible to consider converting the BCI to work in a speech-to-speech mode; there are in particular two factors that weigh heavily in its favour. Firstly, the high quality of the translation output becomes doubly important in this context, since post-editing is for obvious reasons impractical. Secondly, there is already a body of experience on using the the CLE for speech applications, which has shown that the CLE is shown is well-suited to the speech environment. Given that speech front-ends to the English-language version of the CLE and Swedish-language text-to-speech systems already exist, it is likely that a research prototype of an English to Swedish system could be built in a relatively short time.

If the system were to be used within a limited application domain, it would also be possible to consider incorporating machine-learning techniques recently developed at SICS, which use examples of typical sentences to "tune" a grammar to a specific application (Rayner and Samuelsson 1990; Samuelsson and Rayner 1991). This would have the effect of automatically cutting down the range of possibilities for the speech recognizer, increasing its accuracy for domain sentences, and also simultaneously speeding up the analysis phase by a substantial factor.
7.5 The "multi-lingual workstation"

In a wider perspective, it would be possible to use the idea of QLF transfer to create what might be called a "multi-lingual workstation". There is already a growing interest in the idea of using natural language as a standard for accessing knowledge-based applications; for example, (Bobrow et al 1990) describe a system in which English is used as a standard query-language to access a set of knowledge-based tools totaling over 800 different functionalities. The user is not obliged to state which tool or tools he wishes to use, the system deciding this automatically. Broadening the coverage of such interfaces to encompass two or more natural languages has obvious advantages in terms of making them available to a wider audience.

To do this efficiently, it is extremely important to choose an architecture which will allow the main kernel of the system to be shared between the different languages covered; although the peripheral parts, such as the grammar and lexicon, may be quite language-specific, the reverse will tend to be true of the components encoding the central functions of reasoning, database query construction and answer generation. QLF transfer appears to provide a natural solution to the problem. Instead of using transferred QLFs as input to the generator, they would be fed into the knowledge-based query application; similarly, output QLFs encoding the system's answers would be bypassed through transfer before being sent to the selected output language's generator. This would have the effect of transforming a single-language system to a multi-lingual one by doing little more than configuring existing components.

One experiment in this direction that could be undertaken in the near future is a synthesis of the BCI with CLARE, a second-generation natural language interface prototype currently being developed at SRI. CLARE, which includes a version of the CLE as one of its components, is capable of advanced types of linguistic processing, including the interpretation of anaphora, tense and ellipsis.

It is worth pointing out explicitly that the sort of application sketched here is only possible with an translation architecture of the kind used in the BCI, in which the output of semantic analysis can be input both to the transfer component and to the understanding and reasoning processes.

7.6 Interactive multilingual composition of texts

Looking ahead, it may be possible to improve the system to the point where it could be used for interactive multilingual composition of formal documents such as business letters and technical manuals, although this places rather heavier demands on the quality of the output text; existing systems for machine translation of this type of document all rely either on extensive post-editing, or on a highly constrained input language. To achieve the necessary fluency, it would probably be necessary to use a number of new techniques, including generation with reference to statistical collocation information.
7.7 Conclusion

In the above sections, we have outlined several ways in which the current BCI prototype could be improved and extended. We believe that there is a real possibility of using this work to form the basis of a machine-translation system which is both theoretically well-founded and practically useful, and that this would represent an important advance in the field. Hopefully, we will be able to report in due course whether these predictions can be realized in practice.
Chapter 8

References


Appendix A

BNF definition of the QLF notation

First order logic notation

\( \langle \text{formula} \rangle \rightarrow [(\langle \text{predicate} \rangle, \langle \text{argument} \rangle), \ldots, \langle \text{argument}_n \rangle] \)

\( \langle \text{predicate} \rangle \rightarrow \text{love1} | \text{donkey1} | \text{mule1} | \text{tall2} | \text{geq} \ldots \)

\( \langle \text{argument} \rangle \rightarrow \langle \text{term} \rangle \)

\( \langle \text{term} \rangle \rightarrow \langle \text{variable} \rangle | \langle \text{constant} \rangle \)

\( \langle \text{variable} \rangle \rightarrow X | Y \ldots \)

\( \langle \text{constant} \rangle \rightarrow \text{dobbina1} | \text{mary1} | \text{titanic1} \ldots \)

\( \langle \text{formula} \rangle \rightarrow [\text{not}, \langle \text{formula} \rangle] \)

\( \langle \text{formula} \rangle \rightarrow [\text{and}, \langle \text{formula} \rangle, \langle \text{formula} \rangle] \)

\( \langle \text{formula} \rangle \rightarrow [\text{or}, \langle \text{formula} \rangle, \langle \text{formula} \rangle] \)

\( \langle \text{formula} \rangle \rightarrow [\text{impl}, \langle \text{formula} \rangle, \langle \text{formula} \rangle] \)

\( \langle \text{formula} \rangle \rightarrow \text{quant}((\langle \text{quantifier} \rangle, \langle \text{variable} \rangle, \langle \text{formula} \rangle, \langle \text{formula} \rangle)) \)

\( \langle \text{quantifier} \rangle \rightarrow \text{forall} | \text{exists} \)

Higher order extensions

\( \langle \text{if \_form} \rangle \rightarrow [(\langle \text{mood\_op} \rangle, \langle \text{formula} \rangle)] \)

\( \langle \text{mood\_op} \rangle \rightarrow \text{dcl} | \text{whq} | \text{ynq} | \text{imp} \)

\( \langle \text{argument} \rangle \rightarrow \langle \text{formula} \rangle \)

\( \langle \text{argument} \rangle \rightarrow \langle \text{abstract} \rangle \)

\( \langle \text{predicate} \rangle \rightarrow \text{believe1} | \text{want1} | \text{past} | \text{fut} \ldots \)

\( \langle \text{quantifier} \rangle \rightarrow \text{most} | \text{several} | \text{wh} | \text{count} \ldots \)

\( \langle \text{quantifier} \rangle \rightarrow \langle \text{variable} \rangle \text{\(^\wedge\)} \langle \text{variable} \rangle \text{\(^\wedge\)} \langle \text{formula} \rangle \)

\( \langle \text{quantifier} \rangle \rightarrow \text{set}(\langle \text{quantifier} \rangle) \)

\( \langle \text{quantifier} \rangle \rightarrow \text{amount}(\langle \text{variable} \rangle \text{\(^\wedge\)} \langle \text{variable} \rangle \text{\(^\wedge\)} \langle \text{formula} \rangle, \langle \text{functor} \rangle) \)

\( \langle \text{abstract} \rangle \rightarrow \langle \text{variable} \rangle \text{\(^\wedge\)} \langle \text{lambda\_body} \rangle \)

55
\( \langle \text{lambda\_body} \rangle \rightarrow \langle \text{formula} \rangle \)

\( \langle \text{lambda\_body} \rangle \rightarrow \langle \text{abstract} \rangle \)

\( \langle \text{term} \rangle \rightarrow \text{kind}(\langle \text{variable} \rangle, \langle \text{formula} \rangle) \)

\( \langle \text{constant} \rangle \rightarrow 'U'(\langle \text{constant}_1 \rangle, \ldots, \langle \text{constant}_n \rangle) \)

\( \langle \text{constant} \rangle \rightarrow 'SF'(\langle \text{functor} \rangle, \langle \text{constant}_1 \rangle, \ldots, \langle \text{constant}_n \rangle) \)

**Context dependent constructs**

\( \langle \text{term} \rangle \rightarrow \text{qterm}(\langle \text{category} \rangle, \langle \text{variable} \rangle, \langle \text{formula} \rangle) \)

\( \langle \text{term} \rangle \rightarrow \text{a\_term}(\langle \text{category} \rangle, \langle \text{variable} \rangle, \langle \text{formula} \rangle) \)

\( \langle \text{term} \rangle \rightarrow \text{a\_index}(\langle \text{variable} \rangle) \)

\( \langle \text{term} \rangle \rightarrow \text{term\_coord}(\langle \text{category} \rangle, \langle \text{variable} \rangle, \langle \text{term} \rangle, \langle \text{term} \rangle) \)

\( \langle \text{formula} \rangle \rightarrow \text{a\_form}(\langle \text{category} \rangle, \langle \text{variable} \rangle, \langle \text{formula} \rangle) \)

\( \langle \text{functor} \rangle \rightarrow \langle \text{variable} \rangle \)

\( \langle \text{formula} \rangle \rightarrow [\text{island}, \langle \text{formula} \rangle] \)
Appendix B

BCI user manual

B.1 Running the BCI

The BCI is normally run on a pair of SUN SPARC workstations under SICStus or Quintus Prolog, with one machine running the English version of the CLE, and the other the Swedish one. Depending on the set-up of the system, communication between the two version is either carried out using UNIX sockets, or using ordinary text-files. The system is started by performing the following sequence of actions:

- Load the English CLE on the first machine from its saved-state.
- Load the Swedish CLE on the second machine from its saved-state.
- Start the interprocess communication link between the two machines.
- Set the appropriate switches on both copies of the CLE, to disable production of LFs and enable transfer.

The annotated session below shows in detail how this is done. Input from the user is underlined, and comments are in italics, prefixed with a semi-colon.

```
host1%: bci/engdev/Bci ;Load English CLE
yes

| ?- protalk_start. ;Start link (send)   
| ?- protalk_listen(host1). ;Start link (receive)\(^1\)

Connection for service bci/tcp from host2 established
yes

| ?- s. ;Start English CLE
```

\(^1\)Specifying the name of the other host ("host1") is not necessary if the system is using file communication (as opposed to socket communication).
(switch on transfer)

| ?- \texttt{s.} ;Start Swedish CLE

(switch off quantifier scoping)

(switch on transfer)

\textbf{Hello.} ;Sample sentence to translate
Segmentation: < 0.1 secs
Morphological analysis: < 0.1 secs
Syntactic parsing: < 0.1 secs
Semantic analysis: < 0.1 secs
Sorted analysis (no sortal constraint violations):
Sorted analysis:

\texttt{a.form(hello,A,A)}

Confirm this analysis? (y/n): y
... sending as message 1

\texttt{[CR]} ;Type [CR] to read message\textsuperscript{2}

'Hej' ;Translation

Accept translation? (y/n): y

\texttt{Tack!} ;Return message
Segmentation: < 0.1 secs
Morphological analysis: < 0.1 secs
Syntactic parsing: < 0.1 secs
Semantic analysis: < 0.1 secs
Sorted analysis (no sortal constraint violations):
Sorted analysis:

\texttt{a.form(tack,A,A)}

Confirm this analysis? (y/n): y
... sending as message 1

\texttt{[CR]} ;Type [CR] to read message
... translated message 1
(hello.) ;Notification that first message was processed

'Thankyou' ;Translation of return message

Accept translation? (y/n): y

\textsuperscript{2}Typing [CR] to read incoming messages is not necessary if socket communication is used.
B.2 Description of CLE commands

B.2.1 Command types

CLE commands fall into a number of categories which are described in turn below. These categories are:

- basic commands
- help commands
- phase selection and control
- printing mode
- printing
- the stepper
- CLE parameter setting
- loading and re-compilation
- dealing with unknown words
- testing and debugging.

Most of these commands are given from within the CLE loop. Such commands start with a period, to distinguish them from natural language sentences, and consist of one or more tokens separated by spaces. A period at the end of such a command is ignored. CLE loop commands will be presented following the CLE loop prompt “>>”, with argument shown in angle brackets.

Some commands, however, are executed at the level of the Prolog query loop. They must take the form of valid Prolog queries, and therefore have to terminate with a period. These commands will be presented following the Prolog prompt “| ?-”.

To define your own commands is fairly straightforward. See the comments in loop.pl for details.

B.2.2 Basic commands

| ?- cle.

Enter the CLE loop, which interprets both English/Swedish sentences and the CLE commands described below.

| ?- ... , cle(S).

Enter the CLE loop and take input from the stream S, which should have been opened for reading from a file containing CLE commands and sentences. Commands and sentences will be read and processed until end of file or a “.q” command is encountered. For example:

| ?- open(mycorpus, read, S), cle(S).

Alternatively, S can be a Prolog string, from which a single CLE command or English/Swedish sentence will be read and processed. If the string finishes without a punctuation character, “.” will be assumed. Any characters in the string after the end of a sentence will be ignored. For example:
| ?- cle("What are the sales figures?").

>> .=

Print the values of all the processing options or switches, and the commands that would be needed to reverse their values. Most switches have associated with them three commands for turning them on and off and for printing their value. The latter two are typically suffixed by "-" and "=" respectively. For example, the command .i turns on semantic interpretation, .i- turns it off, and .i= states whether it is on or off.

>> .q

Exit from the CLE loop and return to the Prolog query loop. All processing options (e.g., .i-) in force at the time this command is given will still be in force if the CLE loop is re-entered.

| ?- vex.
| ?- qvex.
>> .vex
>> .qvex

Enter the lexicon tool for defining new words to the system. vex defines words individually, while qvex can process groups of words of the same type as a collection. See C for further information.

B.2.3 Help commands

>> .h

List the CLE help commands described in this section.

>> .h (key)
>> .ha (key)

Print help information relevant to the string key, which should be the start of, or the whole of, a CLE or VEX command name or a word appearing in a one-line command description. Thus .ha sort will print out information on CLE and VEX commands relevant to sortal restrictions. (For a description of VEX commands, see Appendix C).

>> .hc (com)

Print help information on CLE or VEX command com.

Each of the remaining help commands gives a description of CLE commands of a particular type:

>> .hb

Print information on "basic" CLE commands (Section B.2.2 above).
Print information on commands that select or otherwise control CLE processing phases (Section B.2.4).

Print information on data structure printing commands (Sections B.2.5 and B.2.6).

Print information on lexicon and rule loading commands (Section B.2.9).

Print information on commands relevant to dealing with unknown words (Section B.2.10).

Print information on system development and debugging commands (Section B.2.11).

**B.2.4 Phase selection and control**

Retry the most recent sentence that successfully underwent segmentation and word correction, in its corrected form. (See Section B.2.10 for commands related to word correction, and Section B.2.11 for applying the CLE to a corpus of sentences).

Retry the most recent sentence in its “original” form, i.e. before any word correction.

Retry the most recent sentence from the interactive disambiguation phase onwards.

Retry the most recent sentence from the transfer phase onwards.

The rest of the commands in this section control the execution of various CLE processing phases. Each command with a "-" switches a phase off, and the corresponding command without the "-" switches it on. When any phase other than sortal restriction application is turned off, none of the subsequent phases take place either.

With a single numerical argument $N$, repeat input number $N$. This works when input numbers are included in CLE prompts, which is the case when the .hm command has been given:
Switch the history mechanism (inclusion of input numbers in the CLE prompt) on or off, respectively. Default is off.

```
>> .hm
>> .hm-
```

Switch the execution of the semantic interpretation phase on or off. Default is on.

```
>> .i
>> .i-
```

Switch sentence generation (of the input sentence) on and off. The default is off. If later processing phases other than transfer are being performed (scoping, reference resolution, etc.) then generation is from the results of resolved interpretations (RQLFs), otherwise paraphrase generation from unresolved semantic analyses (QLFs) takes place. Note that only a subset of the rules used for analysis are available for generation (as specified in declarations in the file that is the value of `setsys parameter gendec`), so it is not the case that every sentence that can be analysed can be generated.

```
>> .g
>> .g-
```

Switch transfer of QLFs on and off. The default is off.

```
>> .s
>> .s-
```

Switch the execution of the quantifier scoping phase on or off. Default is on (but scoping then only takes place if transfer is off).

```
>> .ms (n)
```

Set the total maximum number of scoped logical forms produced for a given sentence to \( (n) \) (default is 10).

```
>> .msa (n)
```

Set the maximum number of scopings produced for each alternative unscoped logical form to \( (n) \) (default is 2).

```
>> .r
>> .r-
```

Switch on and off the application of sortal restrictions, which normally takes place before quantifier scoping and during the plausibility phase. Switching off sortal restrictions is independent of other aspects and phases of sentence processing. (See also commands `.rt` and `.cst` in Section B.2.6 for debugging sortal problems). Default is on.
Switch on and off whether sortal restriction checks (when they occur) are applied preferentially. In preferential mode, the QLF(s) with the fewest sortal restriction violations survives the check. In non-preferential (absolute) mode, only those QLF with no violations at all survive.

```
>> .rp
>> .rp-
```

Switch the reference resolution phase on and off. Default is on.

```
>> .rcm
```

Reset the context model, i.e. clear any possible reference antecedents that may currently exist.

```
>> .e
>> .e-
```

Switch the execution of the logical form evaluation phase on and off, respectively. The default is off. Logical form evaluation involves a call to the interface procedure `process_chosen_lf_applic` (see Alshawi et al 1989).

```
>> .ap
>> .ap-
```

Switch application interface procedures on and off. The default is off. Evaluation is independently controllable with the .e- command.

In fact, .ap switches the flag `loop_option(application_on)` which (as distributed) controls the predicates `plausible_lf_applic` and `track_context_applic`. Resolution methods which are intended to be switched off by .ap- should check this flag, and .ap- can be made to force .e- by having `process_chosen_interpretation` do the same.

```
>> .uv
>> .uv-
```

Switch on and off whether the user is asked to verify candidate logical forms. The default is on.

### B.2.5 Printing mode

These commands affect what information is printed out during processing and the way that information, and the information displayed by the explicit printing commands, is presented.

```
>> .p
>> .p-
```

Switch the printing of intermediate structures on and off, respectively. Default is off.
Switch abbreviated printing mode on and off, respectively. In abbreviated mode, syntactic analysis nodes are printed as identifiers consisting of the name of a syntax rule and an integer that can be used to retrieve a full description of the node using the .pd command. Abbreviated printing mode is the default.

\[ \text{.pa (Mode)} \]

(Mode) affects the lists of alternatives in the display tree derived from the packed syntax records. If (Mode) is bag, the list is a bag of alternatives. If (Mode) is set, duplicates modulo variable renaming are removed. If (Mode) is gen, subsumed items are removed as well. Which constituents are shared can be seen by switching on record abbreviation (see the .pab command above), but it makes sense to use this in "bag" mode.

\[ \text{.pt} \]
\[ \text{.pt-} \]

Switch on and off, respectively, the printing of processing phase timings. Default is on.

\[ \text{.th} \]

Show the current values of the thresholds for printing packed structures. If the number of alternatives implicit in a packed structure is more than the relevant threshold, then before the structure is printed the user is asked to confirm that the printing is required.

\[ \text{.tp (n)} \]

Set the threshold for querying whether to print packed syntactic analysis structures to (n).

\[ \text{.ti (n)} \]

Set the threshold for querying whether to print unscooped logical forms to (n).

### B.2.6 Printing

Most of the commands listed in this section display the internal records generated by the various processing phases. The formats of these records are described in (Alshawi et al 1989).

\[ \text{.f (cat)} \]

Display the syntactic and semantic features associated with the category symbol (cat). The features are printed in the order in which they appear in the internal (compiled) representation of categories.
Print separate syntactic analysis structures (i.e. without packing alternative substructures). Syntactic features are instantiated as fully as possible in these structures. This command is affected by the abbreviation printing mode: if abbreviation is on (i.e., the .pab command has been given), syntactic analysis node numbers are printed and the local trees can be examined using the .pd command. Note that since the semantic interpretation phase deletes syntactic analysis records, this command can be used only if semantic interpretation is switched off.

Print the syntactic constituent records generated by the parsing phase.

Print all syntactic analysis ("daughter") records generated by the parsing phase. If the semantic interpretation phase has been carried out, records involved in complete analyses of the sentence will not be printed, since they will have been ‘consumed’ in the course of semantic interpretation.

Print the syntactic analysis record corresponding to the node numbered \( n \) in the abbreviated syntactic analysis structure.

Print the semantic constituent records generated by the semantic interpretation phase.

Print the unscoped interpretation records generated by the semantic interpretation phase.

Print the rules used by the transfer phase.

Print original entries (i.e. the entries read off disc) for the words or predicate names \( K1 \) \( K2 \ldots \)

Print out comments on sort instantiations (particularly mismatches) that occur in the course of producing logical forms from sentences. This is a diagnostic command, and is best used with sort restrictions turned off (.r-, Section B.2.4) to discover why sortal restrictions are blocking expected interpretations.

If no arguments are given, comments on the final (chosen) logical form are printed. If there are arguments, they should be integers referencing scoped logical forms (e.g. .rf 3 will cause the third scoped logical form to be printed). The argument 0 is taken to refer to the final, chosen, logical form.

Print out sort terms (sortal restrictions) relevant to \( key \), and check their consistency. \( key \) may be either a word or a predicate name.
B.2.7 The stepper

>> .st

Enter the "stepper" to inspect rules and lexical entries and to build syntactic and semantic analysis trees "by hand". The stepper maintains a list of items which may be unified with each other. The items are either categories (for use with syntax rules), or pairs of the form (LF,Category) (for use with semantic rules).

The stepper displays categories with explicit feature names after the application of defaults. Categories in semantic rules are used after unification with their appropriate counterparts in syntax rules.

The stepper uses its own unifier which has an occurs check. When unification fails, the stepper indicates the (first) reason for failure (including occurs check failures). This is useful for pinpointing features causing category unification failure.

Stepper commands are Prolog terms which should be terminated with a period. The stepper prompt is ">>>". The basic commands for building syntax trees and semantic analyses are:

>>> (word).

Include syntactic categories of word (word).

>>> *(word).

Include semantic pairs of word (word).

>>> (ruleid)<[n1,...].

Apply syntactic rule with id (ruleid) to items n1,.... If it succeeds, this results in a category (for the mother) being included as a stepper item.

>>> (ruleid)*(case)<[n1,...].

Apply semantic case (case) of semantic rule with id (ruleid) to items n1,.... If it succeeds, this results in a new semantic pair (for the mother) being included as a stepper item.

Commands for displaying rules and items:

>>> (ruleid).

Display syntactic rule with id (ruleid).

>>> (ruleid)*(case).

Display semantic case (case) of semantic rule with id (ruleid).

>>> p/n.

Display item n.

>>> ps.

Display summary of current items (often more useful than .p).
>>> p.
Display all current items.
Other commands:

>>> []
Throw away all items.

>>> [n1,...]
Throw away all except items n1,....

>>> -[n1,...]
Throw away items n1,....

>>> h.
Print help information about stepper commands.

>>> (m)+(n).
Display results of unifying items (m) and (n).

>>> q.
Exit from stepper loop.

B.2.8 Parameter setting

A number of the CLE loop commands assume that certain parameters have been set. These parameters can be initialized and altered by calls to the two-argument predicate setsys, which is called from the Prolog query loop. For example:

```prolog
| ?- setsys(grammar,'cle/grammari.pl').
| ?- setsys(pagewidth,80).
```

Default values for these parameters are obtained from the file defaultsys.pl. In cases where it makes sense for the parameter to have a set of values, the second argument of setsys may be either a single atom or a list:

```prolog
| ?- setsys(precomplexfile,[funcsorts,funcsenses,funcsyncats]).
```

Parameters may also be altered using the CLE loop command "ss", as follows:

```prolog
>> .ss (param) (value)
>> .ss (param)
```

The first form is suitable for use when the value is a Prolog atom. It should be typed without any quote characters to escape it, e.g.

```prolog
>> .ss grammar cle/grammari.pl
```
The second form should be used when the value is non-atomic. The value will be prompted for, and should be entered as a Prolog term terminated by a period.

Some of the parameters currently used by the system and the interpretations of their values are as described below. Where the default value of the parameter is not explicitly indicated, it is set in the file defaultsys.pl and can easily be consulted or changed there.

**morphology**  One or more files containing morpho-syntactic rules.

**segmentation**  One or more files containing morphological data (standard affixes).

**derivations**  One or more files containing morphological sense derivation rules.

**grammar**  One or more files containing syntax rules.

**semantics**  One or more files containing semantic interpretation rules.

**synfeats**  One or more files containing feature defaults to be applied to syntax rules and entries.

**semfeats**  One or more files containing feature defaults to be applied to semantic rules and entries.

**scoperules**  The file containing quantifier scoping rules.

**hierarchy**  The file defining the type hierarchy (actually lattice) for sort classes.

**test_file**  The file containing the test sentences to be processed by the .t command. (See Section B.2.9.)

**syn_rule_groups**  The value of this parameter is a list of the grammar rule groups that are to be compiled into the parsing table. Changing its value requires re-compiling the grammar as well as other things that depend on it (see the description of the .lg command on page 69). N.B. At present, the only way to change this parameter with any effect is to edit the file synrules.pl where it is reset to a default value at the beginning of the file. Refer to (Alshawi et al 1989), for a few more details on rule-groups.

**ppmode**  The default pretty-printer mode for most printing commands (including pp at Prolog level). This parameter controls the way lists (including argument lists) which do not fit on a single line are printed. In vertical mode they are printed on separate lines, and in block mode they are printed in an indented block. In sorted mode, sorts in logical forms are flattened prior to printing, but otherwise it is the same as block mode. The default for this parameter is block.

**ppdepth**  The maximum depth of embedded structure explicitly displayed by the pretty-printer. Structure below this depth is displayed as elliptical dots (...). Default is 15 levels.
ppwidth The number that is the width (in characters) of the display for use by the pretty-printer. Default is 70.

maxscopings See the .ms command.

max_scope_ambig See the .msa command.

testlexfile,paralexfile,normalexfile,precomplexfile Lexicon parameters. See Sections C.6 and B.2.9.

userimp VEX parameter. See Section C.6.

B.2.9 Loading and re-compilation of rules and lexicons

The formats of rules appearing in the files used by these commands are described in (Alshawi et al 1989). The name of such a file (or list of files) can be reset from those provided by the default system (declared in the file defaultsys.pl) by setting the appropriate parameter using setsys or .ss.

Many of the files can be re-loaded in either compiled or interpreted form. As usual, the compiled form is preferred for efficiency but the interpreted form loads much more quickly, and so is useful during development.

\[\gg .1a\]
\[\gg .1ai\]

Read and re-compile (.1a) or consult (.1ai) all linguistic data files, i.e. syntax rules, morphology rules, sortal restrictions, semantic rules, feature defaults, lexical entries, sense entries, and scope rules.

\[\gg .1m\]

Read and re-compile the (syntactic) segmentation and morphology rules.

\[\gg .1d\]

Read and re-compile the morphological sense derivation rules.

\[\gg .1g\]
\[\gg .1ig\]

Read and re-compile (.1g) or consult (.1ig) syntax rules and their feature defaults. Changes to the sets of syntactic features associated with category symbols are reported to the user. If the feature system has been changed, the .1g or .1ig command should be followed by the .1l, .1i, .1d, .1s, and .1m commands to ensure consistency of internal data structures. The default loading sequence uses .1ig because this saves system compilation time during the development cycle. With the current CLE grammar, .1g increases parsing time by about 20%.

\[\gg .1i\]
\[\gg .1ii\]

Read and re-compile (.1i) or consult (.1ii) semantic interpretation rules and their feature defaults. Changes to the sets of semantic features associated with category symbols are reported to the user. If the semantic feature system has changed, the .1i or .1ii command should be followed by the .1s and .1d commands to ensure consistency of internal data structures.
Read the transfer rules.

Read and re-compile the quantifier scoping rules.

Read and re-compile the sortal restriction hierarchy.

Make (re-compile) the generator tables.

The descriptions given here of lexicon loading commands are a summary of the information in Section C.6.

Make (read from disc) the paradigm lexicon contained in the files that are the values of the setsys parameters testlexfile and paralexfile. The implicit entries in the paralexfile(s) are expanded using those in the testlexfile(s), and compiled into internal format. Finally, all expansions of normal lexical entries are cleared, as they may have become inconsistent.

Make (read from disc) the whole lexicon according the values of the relevant setsys parameters. First the mpl command, described above, is executed. Then the lexicon file(s) specified by the setsys parameters normallexfile and precomplexfile are read and stored, making them available for use by the CLE. Finally, the command .pl (see below) is executed: entries in the precomplexfile(s) are expanded and compiled, to avoid the overhead of doing so during sentence processing.

Make (read from disc) the lexicon file referenced by items, as described in Section C.6.

Expand and compile entries in the file(s) specified by the setsys parameter precomplexfile, loading these files if they are not already loaded.

Expand and compile the lexicon file referenced by items, as described in Section C.6.

Abolish the lexicon file (file): i.e. retract all assertions (both original, uninternalised, and cached, internalised) in main memory that resulted from reading (file) from disc using the .ml or .pl command.
>> alk (key)

As for .alf, but abolish all assertions for the word or predicate (key), whatever files they originated in.

>> clf (file)

Clear the lexicon file (file): i.e. retract all the cached assertions of internalised lexical entries from file (file), but do not abolish original, uninternalised entries.

>> clk (key)

As for .clf, but clear assertions for the word or predicate (key), whatever files they originated in.

B.2.10 Commands for dealing with unknown words

The .vex and .qvex commands (see Sections C and B.2.2) allow the user to define new lexical items at any time. However, the CLE can also be made to try to deal with unknown words when they are encountered, with or without user intervention. The relevant commands are as follows.

>> ipn
>> ipn-

Switch on and off the option of inferring unknown proper names (both single-word and multiple-word) from capitalization and upper casing. Default is off.

>> el
>> el-

Switch on and off the option of looking for (the root form of) each unknown word in an external lexicon. If this option is turned on, the predicate use_external_lexicon/1 should be defined (see Alshawi et al 1989). Default is off.

>> sc
>> sc-

Switch automatic spelling correction on and off. Default is off.

>> ui
>> ui-

Switch on and off user intervention on unknown words. Default is on. Note that regardless of this switch setting, user intervention will not take place during processing of a test file (.t command, Section B.2.11).

>> ssc+
>> ssc
>> ssc-

When spelling correction (.sc) and/or user intervention (.ui) are in force, try to find alternative spellings in all cases (.ssc+), easy cases (.ssc, the default) or no cases (.ssc-).
Switch parsing for spelling correction on and off. Default is off. When it is switched on, spelling correction candidates are not immediately presented to the user. Instead, the system parses and interprets the word lattice defined by the correction possibilities for all unknown words in the sentence, and, if a sortally coherent interpretation is found, eliminates from consideration all candidates that are known not to take part in such an interpretation.

B.2.11 Testing and debugging

\[ \text{\texttt{.t}} \]

\[ \text{\texttt{.t \{file\}}} \]

Read sentences and commands from a designated test file to be processed by the CLE loop under the current execution options. The file must either be given in the command, or, if it is absent, be specified by setting the CLE parameter \texttt{test\_file}. It may contain CLE-loop commands for controlling printing and processing options, and must end with the \texttt{.q} command. It may also contain comments:

\[ \text{\texttt{.c \{word1word2...\}}} \]

Do nothing. This command may be included in test files to provide comments.
Appendix C

Using VEX, the CLE lexicon tool

This section tells you how to use the CLE lexicon tool, VEX (for Vocabulary EXpander), to define new lexical entries automatically. VEX simplifies considerably the task of extending the lexicon by taking care of those aspects of the task directly related to the CLE's linguistic representation. However, like any other aspect of natural language, the English lexicon is a complex object, and because of this, although VEX is designed to be user-friendly, it cannot reliably be used without an understanding of the issues and procedures discussed here.

All system output in this section is shown in **this font**; user input is shown underlined **like this**. The examples given are for the English version of the system; on the Swedish side, the dialogue takes place (almost) entirely in Swedish.¹

VEX should run without modification on UNIX systems. For other systems, two minor changes detailed in the CLE installation notes may be needed. For the purposes of experimentation, VEX will then be ready to use; but for serious, application-specific development work, it will need to be tailored to your domain. The adaptation process for application domains, is overviewed in Section C.9.

To learn how to use VEX, we suggest you first run it on the example dialogue described in sections C.1, C.2 and C.4, then read the other sections as well. For a discussion of the principles on which VEX is based, the reader is referred to the relevant chapter of (Alshawi ed. 1991).

C.1 Getting started

Invoke the CLE; VEX is compiled as part of it.

Starting VEX

There are two ways to use the VEX system. The first way in is to type one of the commands

```
| ?- vex.
|>> .vex
```

to Prolog ("? -") or the CLE command loop (">>"). This initiates a dialogue about the behaviours of words you want to define; VEX decides which of the **paradigm** word

¹The Swedish version of VEX is fully described in (Gambäck, Lövgren and Rayner 1991). In general, the menus in the Swedish version contain a lot more alternatives than do the English ones described here.
behaviours it knows about correspond to the behaviour of your word, and writes out a file containing instructions for building new entries using template entries for these paradigms as starting points. Another CLE command, described in Section C.6, will carry out these instructions.

Starting QVEX

The second mode of entry is to type one of the commands

```
|   ?- qvex.
>>   qvex
```

This is short for “quick VEX”; it assumes you already know that your new words or phrases behave either like one of the paradigms, or like a word you have defined earlier in the session using VEX or QVEX. QVEX is described in Section C.7.

You can switch freely between the CLE, VEX and QVEX by typing .cle, .vex or .qvex whenever the system is expecting input. (Strictly speaking .cle just causes VEX and QVEX to exit, so if you enter them from the Prolog command loop rather than from the CLE itself, .cle will put you back there.)

C.2 Morphology and syntax

The global comment

Enter the system from Prolog by typing

```
|   ?- vex.
```

You are told

```
******************************************************************************
*                        Entering VEX, the Vocabulary Expander.            *
*                        Type ? in response to any query you don’t understand. *
******************************************************************************

Please enter a GLOBAL comment which will be attached to ALL the entries that you create in this session. Finish the comment with a period on a line by itself.
%
```

This request is only made the first time you invoke VEX or QVEX in a Prolog session. You might respond with your initials and the date, or some other information that will apply to all the entries you are about to create, for example Test entry.

Words and word groups

VEX then tells you

```
The current VEX/QVEX switch settings are as follows:
* Entries will be placed in <phrase>.defs files.
* Common paradigms will be considered before uncommon ones.
* The global comment (which can be altered with ".gc") is:
```
% Test entry

Enter the word or word group to be defined:

The meanings of the switch settings referred to can be ignored for now. You should respond with the word(s) you want to define. Often there will only be one word; but you can also define phrasal verbs like “belt up” and compound nouns like “carbon tetrachloride”. Note that:

- input should be in lower case, except that initial letters of names should be capitals;
- full stops at the ends of input lines will be ignored, except when you are inputting comments;
- VEX only allows you to define one phrase at a time;
- you should always enter the root forms of words, e.g. “use”, not “uses” or “using”;
- you can obtain advice at any time by entering ?.

So if you wanted to define all the words in the sentence “Chemists use carbon tetrachloride up”, you would supply “chemist” in answer to the query on one occasion; “use up” on another; and “carbon tetrachloride” on another. (“Use” and “up” form a phrase despite being separated in the sentence. Suitable candidates for phrases are multiple-word nouns and names like “telephone directory” and “John Smith”, and verbs and adjectives followed by specific, compulsory prepositions or particles like “look at”, “get out of” and “appropriate to”).

Let’s assume you want to define “use up” first (although it is, in fact, already defined in the core lexicon). You answer the query:

Enter the word or word group to be defined: use up
terminating your input with a return.

(If, at this stage or any other, you make a mistake and hit “return” before you realise it, you can type . to tell the system to backtrack to some earlier point in the dialogue. See Section C.8 below for details.)

Parts of speech

You are then asked

Is this an adjective, noun or verb in your domain?
   Enter one or more of a,n,v:

The system assumes you know what these three parts of speech are, but don’t worry, you need to know very little else about syntax! If in doubt, consult a dictionary.

Categories such as prepositions and determiners are not definable through VEX because they are “closed classes” of “function” words: English has a fixed, usually small number of them, and if the CLE can deal with them they should already be well represented in the core lexicon provided with the system. This is also true of those adverbs that are not derived from adjectives by adding “-ly”.

Back to our dialogue: “use up” is a verb, so you respond:
Is this an adjective, noun or verb in your domain?
Enter one or more of a,n,v: v

Some words will be of more than one category: for example, “leap” is both a noun and a verb, so (if both uses were relevant to your application) you would respond **nv** here when defining “leap”. Also, if you know you want to define distinct senses of a word where there may be some syntactic similarities (e.g. the noun “bank” as both a financial institution and the side of a river), then you can repeat a category symbol, and go through the dialogue once for each sense.

**Morphological inflections**

VEX responds

I am now defining "use up" as a verb.

Enter the forms of the verb "use" (hit return if a form does not occur):
(Present 3rd person singular must be "uses".)
(Present participle must be "using".)
Simple past, e.g. yesterday John used [used]:

VEX is able to deduce the present tense, third person singular form of the verb, as well as the progressive form. For other forms, to save keystrokes, it guesses what the answer will be, and displays it inside square brackets; if its guess is correct, as it is here, just hit return. (At any point in a VEX dialogue where a default is offered in square brackets, hitting return will select that default).

In each case, VEX’s guess is correct, because “use” behaves regularly, so just hit return each time. If you are sure that your word has regular inflections, you can save time by typing **.ok** in response to one of these queries, and the rest will not be asked. If you make a mistake and want to backtrack to the previous query, type **.bt**.

**Grammaticality judgments**

Then, using the information that we are defining a two-word verb, VEX selects a subset of the patterns it knows about and asks you whether they are grammatical:

Here are some possible sentences for assessment:

1 The thingummy used up the whatsit.
2 The thingummy used the boojum up the whatsit.
3 The boojum was used up the whatsit by the thingummy.

Please enter the numbers of the sentences that are grammatical in your domain for the word/phrase being defined, or "?" for help:

The information you are about to provide enables VEX to do the tricky part of its job, so be careful! The first thing you will notice is the use of words like “thingummy”, “boojum” and “whatsit”. These are to help you bear in mind that you are being asked whether the sentences are **grammatical**, not whether they are **meaningful**. A sentence can be grammatical (“correct English”) whether or not it is meaningful, for example:
Hairy old gorillas snore loudly. (Grammatical and meaningful.)

Colourless green ideas sleep furiously. (Grammatical, but not obviously meaningful.)

Gorillas old snore hairy loudly. (Not grammatical, though some meaning may be discerned.)

**Word substitution**

To help you decide whether a sentence is grammatical, you may like to try to think of nouns to substitute for “thingummy”, “boojum” and “whatsit” that make it sound better. This kind of substitution is legitimate as long as you replace like with like. For example, a noun such as “thingummy” should not be replaced by an adjective or a verb; nor should it be replaced by a noun like “fact” or “intention” which expects a following clause (“the fact that...”, “the intention to...”). You may also substitute a verb for “exist” (which appears in some sentence patterns presented later), but if you do, it must be one that, like “exist”, doesn’t take an object. Examples of such verbs are “disappear”, “sneeze”, and “stroll”, but not “strike” or “say”. You can tell VEX to use substitute words in the sentences it presents by typing .sub and entering the substitutes as requested. Substitutes are reset when a new word or category is defined. To save time, VEX assumes regular morphology for any substitute verb entered, so that if you enter “sleep” for “exist”, then “existed” will be replaced by “slepted”.

**More on grammaticality**

You should also note that VEX is asking you what would be grammatical *in your domain*; that is, what constructions could occur in your domain. To some extent, the syntactic part of your lexicon, as well as the semantic part, will be domain-specific.

Further, note that you are being asked which sentences are grammatical *for the word or phrase being defined*. In other words, VEX is asking which sentence patterns you want the CLE to recognize as occurrences of what you are defining. Thus you would not enter a 2 here, because although a sentence like “John used the ice axe up the mountain” is grammatical, it is not an occurrence of “use up” in the (intended) sense of “consume” but of “use” with a location modifier.

When you are defining a verb, example sentences with verbal complements are separated from those without. A verbal complement is a phrase like “that the thingummy exists”, “to exist”, or “whether the boojum exists”. The separation is made because there are a large number of possible patterns for single-word verbs and adjectives with verbal complements; if you know your word never takes a verbal complement, you can ignore these.

Enter your answers as numbers separated by spaces; the correct response here is:

```
Please enter the numbers of the sentences that are grammatical in your domain for the word/phrase being defined, or "?" for help: 1
```

That is, sentence 1 is grammatical (in the sense just discussed), while sentences 2 and 3 are not.

**Less common paradigms**

You are then told:
If the definition can be expressed in terms only of common paradigms, then the following sentences will all be ungrammatical, or only grammatical through a transformation from a sentence you have already approved.

Simpler sentences (without verbal complements):

1 The thingummy used up.
2 The thingummy used the whatsit up.

More complex ones (with verbal complements):

3 The thingummy used up existing.
4 The thingummy used up the whatsit to exist.
5 The thingummy used up the whatsit that the boojum existed.

Please enter the numbers of the sentences that are grammatical in your domain for the word/phrase being defined, or "?" for help:

As discussed in Chapter 12 of Alshawi et al (1989), some paradigms are more common than others, and only the common ones were used to generate the first list of sentences. This second list involves less common paradigms, and if only common paradigms apply to the word you are defining, then nothing in the second list will be grammatical.

However, in our example, sentences 2 and 4 are in fact grammatical, so you enter the response 2 4.

Implication judgments

This prompts VEX to ask:

Does "the thingummy used up the whatsit to exist" necessarily imply
"the thingummy used up the whatsit IN ORDER TO exist"?
Type "y", "n" or "?":

The reason VEX asks this is that it is trying to work out whether sentence 4 is grammatical simply because “to exist” is an optional modifier. In this case, it is; in other words, “the thingummy used up the whatsit to exist” just means the same as “the thingummy used up the whatsit, and it did so in order to exist”. (For other verbs, the “to” phrase is essential: for example, “the thingummy wanted the whatsit to exist” doesn’t mean “the thingummy wanted the whatsit, and it wanted it in order to exist”. For yet others, it is ambiguous: “the thingummy needed the whatsit to exist” may or may not mean “the thingummy needed the whatsit, and it needed it in order to exist".) So you answer y to the query:

Type "y", "n" or "?": y

This is effectively the same as telling VEX to ignore the 4 that you gave it earlier. If you are unsure how to answer this question or one like it, type ?, and a few lines of explanation will be printed.
Note that VEX asked whether one sentence *necessarily* implied the other. If one or both sentences have several meanings in your domain, you should give a positive answer only if the implication holds for *all* readings of both sentences; if you were defining “need”, as discussed above, you would answer \( \Diamond \) here.

Assessing the paradigm choice

VEX now works out which of its paradigms your phrase behaves like. Here, there is only one: the paradigm named \( v_{\text{subj_obj_particle}} \), which covers transitive particle verbs. (The semantics of paradigm names is described in comments in the code file *vexpats.pl*). VEX therefore tells you:

The paradigm accepted is "\( v_{\text{subj_obj_particle}} \)."

The paradigm \( v_{\text{subj_obj_particle}} \) (Verb like "John THROWS Mary OUT") allows at least the following constructions:

1. The thingummy used up the whatsit.
2. The thingummy used the whatsit up.
3. The whatsit was used up by the thingummy.

Enter a name for this sense ("=" will expand to "use_up"):

In other words, any sentences of the forms 1 to 3 will, if all goes well, be analysed by the CLE as occurrences of a predicate whose name will be, by default, \( \text{use_up} \). You should verify that these three sentences (not all of which, incidentally, were originally presented to you) are all grammatical, and, when they are meaningful and mention the same things, can be synonymous; if they're not, then either you have made a mistake somewhere, or VEX has been unable to separate different senses of the word on syntactic grounds (see Section C.3 below), or the paradigm behaviours that VEX knows about do not cover the verb you are defining (or do not agree with your intuitions on it). So check your answers, and make sure you still agree with them all; if you change your mind, you can backtrack with a .\_\_gr, as explained in Section C.8.

Sense names

If you hit return in answer to the request for a sense name, the name \( \text{use_up} \) will be used. However, you may want to specify some other name. The character \( = \) will be expanded to \( \text{use_up} \) in whatever you type. Generally, unless you are sure that the phrase in question can only have one interpretation, it is best to provide some distinguishing mnemonic (such as \_\_converse):

Enter a name or extension for this sense: \[ \text{[use_up]}: =\_\text{converse} \]

C.3 Dealing with inconsistencies

Sometimes, VEX is unable to separate different uses of a word on syntactic grounds. This problem can manifest itself in several ways:

- VEX is able to find a consistent set of paradigms to base its new entries upon, but there is some overlap between them: that is, there is a construction that will be
recognised as an instance of more than one paradigm, and may lead to multiple parses. When this happens, VEX prints a warning.

- VEX gives no warning, but prints non-synonymous sentences under the same paradigm header. For example, the verb "remember" can occur in sentences like "John remembered to go home" and "John remembered going home"; however, these are not synonymous, as are the pair "John attempted to go home" and "John attempted going home".

- VEX finds several alternative paradigm sets of equal size, and has no way of choosing between them. When this occurs, a warning is printed.

If any of these situations arise, you should determine which of the grammatical sentence patterns are non-synonymous and therefore correspond to different senses, and enter them in separate VEX dialogues (e.g. by typing \texttt{vy} rather than \texttt{v} when asked for the list of syntactic categories). For this purpose you should ignore any sentences that VEX has stated (and you agree) are grammatical only by virtue of the addition of an optional prepositional phrase modifier or similar device.

**Paradigms and sentence patterns**

To separate out the different senses, you may wish to inspect directly the relationship that VEX assumes between paradigms and sentences. This can be done when you asked to make grammaticality judgments. If you include a \texttt{+} with your list of sentence numbers, VEX will tell you what paradigms would make those sentences grammatical, and what other sentences are implied by each of those paradigms.

**Negotiation**

Sometimes, however, VEX is unable to select any set of paradigms that together account for the behaviour of your word. It will then give you a chance to alter your answers. Suppose you were defining the word "smoke" as a verb, and had marked "the thingummy smoked" and "the whatsit was smoked by the thingummy" as grammatical, but not "the thingummy smoked the whatsit". VEX would then say

\begin{quote}
Sorry, I can't yet put together a consistent set of paradigm words.
\end{quote}

Either the following must be REJECTED:
\begin{itemize}
  \item a The whatsit was smoked by the thingummy.
\end{itemize}

or the following must be ACCEPTED:
\begin{itemize}
  \item b The thingummy smoked the whatsit.
\end{itemize}

Enter the letter of the sentence or sentence set you want to change your mind about:

This means that, according to the information VEX has, "the whatsit was smoked by the thingummy" cannot be grammatical unless "the thingummy smoked the whatsit" is also grammatical. You are therefore asked to reject the one as grammatical or accept the other. Either an \texttt{a} or a \texttt{b} response will satisfy VEX; the correct answer here is \texttt{b}.
In this example, you are asked to choose between a pair of single sentences; in general, there may be several options, and each option may consist of a set of sentences that must be accepted or rejected together. You cannot accept some members of such a set and reject others. However, you can respecify your original grammaticality judgments by inputting a .gr backtrack request.

Occasionally, VEX may ask you for more than one such correction; this does not mean that your first correction was unsatisfactory, merely that several adjustments are needed to make the set of grammatical sentences consistent. You may also find that, after a correction, you are asked a question about an implication relation between sentences that you have answered before. This is done for safety.

The existence of the negotiation facility should not lead you to adopt a strategy of selecting as grammatical just one or two of the sentences offered, and relying on VEX to "fill in the gaps". Such a strategy is unwise, because by not selecting a sentence, you are making a definite statement that it is not grammatical. VEX may, fortuitously, be able to assemble a consistent but incorrect set of paradigms on this basis; if it can do so, it will not ask you for any adjustments.

C.4 Sortal restrictions

After telling you which paradigms have been selected (either immediately or after adjustment), VEX asks you for information on the sortal restrictions for each sense. In our example, there is only one sense. VEX tells you:

The logical form for the sentence
"The thingummy used up the whatsit"
will contain the fragment
"[use_up_consume,<event>,<thingummy>,<whatst>]

Input one of the following:

(1) = sense1 to say that "use_up_consume" has the same sortal restrictions as the predicate "sense1";
(2) - to have no sortal restrictions;
(3) a (or return) to enter sortal restrictions on arguments:

In essence, VEX is asking you here what sorts of things can "use up" what sorts of other things. It shows you how the logical form of the "use up" construction is composed from those of its constituents. <event> will be the event variable; <thingummy> is the term representing the individual denoted by "the thingummy"; and <whatst> derives in the same way from "the whatsit".

The easy option here is just to type -; this would produce no sort rule for "use up", and so no semantic checks would be made on its arguments. In general, however, the sortal restrictions of the new word will need to be specified.

Choosing the strategy

Let's assume that, in your domain, "using up" is something done by a human to a liquid, and that the sort hierarchy released with the CLE is in place. (We make these assumptions
for illustrative purposes only; in reality, these particular semantics for "use up" are unlikely, and in any case a domain-specific hierarchy should be in use). You could tell VEX that use_up_consume has the same restrictions as, say, manufacture_3p by giving the response = manufacture_3p. However, it is also possible to enter the restrictions argument by argument. You do so by typing a or return. (The reply all has a similar effect to a, but prompts also for the sort of the result (range) of the predicate and of any argument positions that will be filled by propositions derived from clauses. The a response assumes default values for these, which should be correct for all normal purposes).

Equivalence and instancehood

For each of the arguments <event>, <thingummy> and <whatsit> in turn, VEX asks you for some places in the sort hierarchy that constrain the argument, or for a relation to another sort that is assumed to exist. The conventions are that:

= sense1 gives a structure matching the whole sort of sense1;

> sense1 gives one matching the "result sort" of sense1;

< sense1 gives one matching the sort of the first argument of sense1;

<< sense1 matches the second argument, and so on.

Thus if sense1 had sort [a,b,c]=\rightarrow d then the four structures just alluded to would match [a,b,c]=\rightarrow d, a and b, respectively.

The distinction between = and < is subtle but vitally important. When used to define the whole sort at once, as we could have done above as an alternative to the a command, = means that the item being defined is equivalent to another, or is a variety of it, as far as its sort is concerned, while < means it is an instance of it. So we might define the sort of general_moto rs1 by either = austin_rover1 or < manufacturer1. In fact, the < option will not normally be appropriate for defining a whole sort in one go, except for defining proper names (General Motors) as instances of concepts (manufacturers). It will, however, be appropriate for defining arguments, as these represent individuals. So some typical relations might be:

- fred1 < man_male_human (Fred is an instance of the class "man");
- fred1 = bill1 (Fred is the same kind of entity as Bill);
- programmer1 = person1 (A programmer is the same kind of entity as a person);
- <thingummy> < person1 (The individual denoted by "the thingummy" must be an instance of the class "person").

VEX does its best to check that the inputs you give it are consistent not only sortally (e.g. it complains if given the list male female) but also in terms of the types of predicates and arguments as explained here. Thus if you provide the input

Specify restrictions on sort for <event> [event]: = sleep_2p

VEX complains
The default restriction here on the event variable position for use_up_consume, event, is not quite specific enough; the default hierarchy distinguishes between “physical events”, those that take place in a specific location and can therefore be qualified by phrases like “in the laboratory”, and abstract events, which cannot. In addition, “use up” is not “propositional” in the sense that a verb like “tell” is; that is, one cannot say “John used up the chemical about the problem”. Thus an appropriate response is:

**Specify constraints on sort for <event> [event]:** physev nonprop

Then VEX asks

**Specify constraints on sort for <thingummy>:**

Remember, the <thingummy> is what’s doing the using up; and we’re assuming that using up is, in the domain in question, something done by humans to liquids, and that human is part of the hierarchy but liquid stuff is just a predicate in the lexicon. (If you forget how the sentence participants map onto logical form arguments, type .ds and the relationship will be redisplayed).

So we say that the <thingummy> is human and mobile:

**Specify constraints on sort for <thingummy>:** human mobile

Then VEX asks

**Specify constraints on sort for <whatsit>:**

We want to say that the <whatsit> must be a liquid. Because this isn’t in the sort hierarchy itself, we do it by saying that anything that can fill this role must also be able to fill the (only) argument of a liquid stuff predicate, or in other words, the <whatsit> must be an instance of a liquid; thus

**Specify constraints on sort for <whatsit>:** < liquid stuff

Sharing constraints between arguments

VEX allows you to specify that two arguments of a predicate must have “the same” sort in two different senses of “same”. If you answered the above query with «use up consume, the effect would be the same as having typed human mobile as you did for the second argument. However, specifying an operator on its own (e.g. just «) means that the two argument positions must have the same sort after they are filled. In that case, not only would the user-up and the entity used up both need to be human and mobile, but their sorts would also need to be compatible with one another. Sort sharing of this kind might be appropriate for verbs like “become” or “resemble”; one could place no restriction on the second (subject) argument, but say that the third argument would have to be the of the same sort as the second. This would allow “John became a guerrilla” but not “John became a gorilla”.
Inspecting other senses

To help you decide on appropriate sorts, the command .eg can be given when a sort is prompted for. This will print out some other senses that have been defined using this paradigm. For the "use up" example, with the core lexicon loaded, we would get

The instances of the current paradigm are:
carry_on_continue (from "carry on")
carry_out_perform (from "carry out")
give_up_object (from "give up")
note_down (from "note down")
set_up1 (from "set up")
think_out1 (from "think out")
throw_out1 (from "throw out")
use_up1 (from "use up")

(use_up1 occurs here because "use up" is, as we know, already defined in the core lexicon). Details of the sorts of one of these can then be obtained by, for example, .eg think_out1, giving

The sort of "think_out1" would be specified by "all" and then:
<event>: physev located prop
<thingummy>: animate
<whatsit>: < proposition1

Extra sort rules

The sort definition for use_up_consume is now complete. But sometimes you may want to specify two alternative sort rules for the same predicate; for example, you might want to say that the adjective "convincing" could apply either to a proposition or to a human. While it would normally be better to define two different senses of "convincing" in this case, so as to make the most use of the CLE's discriminatory power, VEX does allow you instead to define an alternative sort. It therefore asks

Enter an alternative (extra) sort rule? (y/n) [n]:

However, on this occasion, no extra rule is required, so we hit return.

The definitions file

VEX now has most of what it needs to know. It tells you

Now we've finished defining sortal restrictions for the sense "use_up_consume" of the phrase "use up".

I'm putting the entries for "use up" in the file use_up.defs...
Please enter comment for these entries, finishing with a period on a line by itself
%
**use_up.defs** is a text file containing the instructions for building entries for "use up" that VEX has constructed. You should enter the requested comment (e.g. an example sentence like "a person uses up a liquid"). Then, examine this file. If you’re not happy with it, you can edit it, or you can define "use up" with VEX again in a different way; the file use_up.defs will then, at your option, be overwritten.

If the .defs file to be written already exists, VEX will ask you whether you wish to overwrite it or extend it. Overwriting is straightforward; however, if you opt for extension, then VEX will first scan the existing file for numerically-suffixed senses of the word or phrase you are defining (e.g. for atoms such as use_up1 and use_up2). It will then alter the new sense numbers, if any, to avoid clashes with the old, warning you as it does so. In this way, it is possible to define different uses of a word on different occasions. However, only numerical suffixes are detected and altered in this way.

**Making entries available**

Once the .defs file has been written, VEX will ask:

```
Make entries immediately available to CLE? (y/n) [y]:
```

If you answer y (or just hit return), VEX will read the entries in the file into memory and assert them. It will also check to see whether the head word ("use" in this case) already has some definitions. If you are running with the main core lexicon file coreimp.pl loaded, VEX will ask:

```
Mask out entries for "use" in file coreimp (type "p" to see what they are)? [y]:
```

If you answer y (either immediately or after inspection with p) then VEX will add the assertion "ignore(use,coreimp)." to use_up.defs. The effect of this, both now and whenever use_up.defs is loaded in future, of causing all entries for "use" (including those for "use up") in file coreimp.pl to be ignored or masked out. This is the behaviour you want when you are defining replacement entries for a word, rather than additional ones.

Of course, ignore assertions can also be added to and removed from a .defs file by hand at any time.

**C.5 Relational nouns**

When you define a count noun sense, (i.e. a sense of a noun that can be preceded by "a" or "an") you are given the option of defining a relational predicate as well. Many nouns in the core lexicon are defined as relational in this way by assertions such as

```
relpred(friend1,friend1_of,[poss,genit]).
```

The interpretation of this assertion is that if friend1(Y) is true for some Y, then there must be a x such that friend1_of(x,Y) is true. In other words, if someone is a friend, then that person must be a friend of somebody or something. relpred assertions are picked up during reference resolution to deal with the vague predicators in phrases such as "John’s friend” and “the friend of John”. It is up to you whether or not you choose to define a relational sense; the decision will depend on the application. For example, although in many situations the noun “customer” might be viewed as relational (any customer is a
customer of something or someone), an interface to a database used in the sales office of a company might specify customerhood as non-relational if every customer is represented as a customer of the company in question and only of that company.

An example

Suppose that we do want to define “customer” as relational. When we have defined a sort for the count noun predicate customer1, VEX asks:

Do you wish to define a relational noun sense associated with customer1? (Answer y/n) [n]:

You answer y to this query, and are told:

New sense will be named "customer1_of", so that
[customer1,Y] => [customer1_of,X,Y] for some X.
(i.e. if Y is a customer1 then Y is the customer1 of some X).
Enter sortal constraint(s) on X:

It is necessary here to specify a sortal constraint, in the usual way, on the class of entities X that can have (not be) customers. For the default hierarchy, the class human_group might be appropriate.

Finally, VEX elicits information on the kinds of vague constructions that should give rise to instances of the relational predicate. To construct sentences exemplifying these, it asks for examples of things that can be, and have, customers:

Enter a noun phrase (e.g. John, the man, two metres) describing something that can BE a customer: the buyer

You should enter a complete noun phrase (e.g. “the buyer”) here rather than a count noun such as “buyer”; and similarly for the other participant in the relationship:

Enter another noun phrase, this time describing something that can HAVE a customer: the company

VEX then asks you to judge a list of sentences both for grammaticality and for similarity of meaning:

Here are some possible sentences involving the relational use.

1 The company’s customer is the buyer.
2 The customer of the company is the buyer.
3 The company has a customer of the buyer.
4 The company customer is the buyer.

Enter the numbers of the sentences that can occur and mean roughly the same:

The correct response here is 1 2. (Sentence 3 may look odd, but is appropriate for relational nouns such as “weight”, as in “John has a weight of 50 kilos).
C.6 Lexicon file processing

In this section, we first describe the relationship between the lexicon files, then present the CLE commands for reading lexicon files into memory and expanding them.

Lexical entry assertions

The following, with some indentation put in for readability, is what VEX puts into use_up.defs:

```
% Test entry - entry for "use up"
% A person uses up a liquid
subst_record(lex,v subj_obj particle,
    [v subj_obj particle, para particle],
    [use, up]).
subst_record(sense, v subj_obj particle,
    [v subj_obj particle1,
    v subj_obj particle,
    para particle],
    [use up consume, use, up]).
sor(use up consume, [[physev, nonprop],
    [human, mobile],
    arg(liquid stuff, 1)]
    => [prop]).
ignore(use, coreimp).
```

This says that:

- The syntactic ("lex") entry for "use up" is to be constructed by taking the lex template for v subj_obj particle and altering all occurrences of v subj_obj particle in it to use, and all occurrences of para particle to up.

- The semantic ("sense") entry is to be constructed in a similar way from the sense template for v subj_obj particle.

- The predicate use up consume maps triples of physical events, mobile human entities and liquids onto truth values.

- All entries for "use" in library file coreimp (or coreimp.pl) are to be ignored.

Sometimes you may wish to create new entries by editing old ones by hand rather than by using VEX or QVEX. If you do so, you will encounter assertions of the types just exemplified, plus assertions for irregular inflections, such as:

```
irreg(spoken, [speak, en], v_v_en).
```

The first argument is the inflected form, the second a list of regular components to which it is equivalent, and the third the morphology rule to be used to combine them.
Managing new definition files

You may, at any time, instruct VEX not to create a .defs file for each future word or phrase but instead to append all new entries to a single file. This is done with the VEX "backtracking" command .uuf (see Section C.8) which, despite its name, does not actually cause any work to be undone. The effect of .uuf can be reversed with .mdf ("make .defs files"). The advantage of appending to a single file is that you can conveniently collect all new entries in one place. The disadvantages are that such entries cannot be loaded immediately into the CLE, and neither can they be used as templates by QVEX (see Section C.7). A good strategy is the one adopted for core lexicon content words: to keep all entries in .defs files, but also to concatenate them together for ease of loading by the .ml command (see below).

The name of the file to which entries are appended when .uuf is in force is the value of the setsys parameter userimp. The initial value of userimp is set in defaultsys.pl to userimp.

Implicit entries

The "imp" in “userimp” stands for “implicit”. The advantage of creating implicit entries – entries defined in terms of how they differ from explicit paradigm ones – is that if the paradigm entries on which they are modelled are altered at some later date, no further manual alterations are needed; all that is necessary is to redo the expansion. Furthermore, implicit entries are, as we saw, relatively succinct, and are intended to be human-readable. The implicit/explicit distinction does not apply to sortal information, however; sor assertions are simply copied on expansion.

Making entries available

To become available to the CLE, a lexical entry must be read into memory and internalised, and the internalised result cached. It is read either by VEX, as we have already seen, or with the .ml or .pl CLE commands. An explicit entry is internalised directly, and an implicit one by expansion from an internalised paradigm entry. Internalisation is triggered either by the .pl command or by the CLE trying to access entries for a word for the first time.

The stages of lexicon processing

More globally, the stages of lexicon processing are as follows.

(1) When the system is created, or when the .mpl or .ml CLE commands (without arguments) are given:
  - the file(s) that are the value of the setsys parameter testlexfile (default testsyncats.pl and testsenses.pl) are read into main memory.
  - the file(s) given by parameter paralexfile (default paraimp.pl) are read in, and the implicit entries in them are expanded by substitution from testlexfile entries. All the paralexfile entries are then internalised and stored.²

²When a paradigm sense entry is internalised, the CLE looks for a syntactic category for the same paradigm that is consistent with the syntactic features specified in the sense entry. If there is exactly one such syntactic category, the sense entry is unified with that category before being stored; otherwise, a warning is given. This additional instantiation of sense entries should allow the correct syntactic entries
(2) When the system is created, or when the .ml command (without arguments) is given:

- the file(s) given by parameters normallexfile and precomplexfile are read in.
- Entries in the precomplexfile(s) only, and any other entries for those words in a normallexfile, are internalised directly (if explicit) or by expansion from the internalised paralexfile entries (if implicit). This action is also achieved by the CLE command .pl without arguments.

(3) Lexicon files that are not referenced by any setsys parameter can also be read in, either automatically during a VEX session or by using the .ml (or .pl, for immediate internalisation) command with one or more arguments. If more than one argument is given, the arguments are concatenated using the character "_"; thus .ml use up will load the file library(use_up.defs) (i.e. the file use_up.defs in some Prolog library directory). If one argument is given, it will be interpreted as a word, if the relevant file exists (e.g. .ml use will look for library(use.defs)) and failing that, directly as a relative file name (e.g. in .ml use.defs) or, as a last resort, an absolute file name (e.g. .ml /home/mydir/use.defs).

(4) When a word (or an inflected form of it) occurs in a CLE sentence, the attempt to retrieve the internalised entries for that word for the first time results in those entries being internalised and remembered. Internalisation is, again, either direct or by expansion from paradigm entries.

Clearing and abolishing lexicon data

You may undo the effects of .ml and .pl using the commands .alf, .akf, .clf, and .ckl, each of which takes one or more arguments. In these command names, "a" stands for "abolish", "c" for "clear", "f" for "file" and "k" for "key", where a key is a word or a predicate name. To clear a key means to retract all the cached (internalised) assertions for that key; this should not, on its own, be necessary during normal use of the system. To clear a file is to clear all the keys defined in that file. To abolish a key is to clear it and also to retract the (non-internalised) assertions for it read from the disc. To abolish a file is to clear all keys defined in that file and to retract all assertions read from that file (note, however, that if key K is defined in files F1 and F2, then abolishing file F1 will not cause all the assertions for K to be abolished, but only those that originated in F1).

You may ascertain the effect of one of these commands by telling the system to print the entries for a key that it read from disc with the command .poe, which expects one or more keys as arguments.

to be picked up during semantic interpretation, eliminating interference between entries originating from different paradigms. If the test lexicon is extended, care should be taken that, whenever two paradigms P1 and P2 could both be selected by VEX for the same word, each sense entry for the test lexicon word W1 corresponding to paradigm P1 is discriminating enough to distinguish the intended syntactic category from all others for both W1 and the word W2 corresponding to P2. For practical purposes, two paradigms normally have this property if and only if the sets of sentence patterns that they generate do not overlap.

³Note that Quintus Prolog, and hence the CLE, cannot derive relative file names from absolute ones, and hence cannot recognize the equivalence of an absolute and a relative reference to the same file. Absolute file name references are therefore to be discouraged, mainly because ignore assertions will not mask out entries in files loaded in this way.
C.7 A QVEX dialogue

QVEX, or Quick VEX, is used for defining sets of words that are known to behave in the same way. For example, let’s assume that you want to define a whole list of proper names (Tom, Dick, Harry etc). You do the first one by means of VEX, as above; this creates tom.defs for you. But now, rather than go through the whole dialogue again giving exactly the same answers, you can just supply the input qvex to VEX or to the CLE, and the system will respond

Entering QVEX, the CLE Quick Vocabulary EXPander

Enter word or phrase to copy from: either a paradigm one, or one defined in a .defs file in an accessible (library) directory (return gives paradigm menu):

to which you reply (on the same line)

Tom

You are then asked

Enter new items, separating phrases with commas:

and you reply (on the same line)

Dick, Harry, Bill, Bert

Remember the commas! They are used to separate phrases, which may consist of more than one word. A comma may, but need not, be followed by a space. Commas should only be used here; if used in answer to any other VEX or QVEX query, they will cause errors which may go undetected.

If the word or phrase being copied from is a paradigm one, you will be asked to specify sortal information again. However, if the original is a word or phrase in an accessible "defs" file, as "Tom" is here, then it will be assumed that the sorts of the new words are the same. The files dick.defs, harry.defs and so on will thus be created, and should be dealt with in the normal way (or, if .wur is in force, data will be appended to user imp).

If you hit return when the paradigm is requested, the system will print a menu of paradigm codes, each associated with a number to aid selection. Typically you would determine which code to choose by looking at the .defs file for an existing entry constructed by VEX. The Prolog predicate diagnose, described above its definition in module vex diag.pl, provides information on paradigm behaviours.

C.8 Backtracking

If you make a mistake in your input, it is normally possible to backtrack to the point where you made it and redo the dialogue from there. If, at any point, you realise you have gone wrong somewhere, then type a period as the answer to the question you are being asked. VEX (or QVEX) will then present you with a list of places you may backtrack to.

For example, suppose you are defining "use up", and realise when you are being asked for inflections that you mistyped use up as usw up. To the query

(1) Present 3rd pers sg: John ... s : [usws]:

you would type a period. You would then be told:

**Backtracking options at this point are the following:**

- `.ac` abort this category, start on next if any
- `.anc` ask for new categories (noun, verb etc.)
- `.bt` backtrack to preceding (or recent) input
- `.gc` ask for fresh global comment
- `.mdf` make `<phrase>.defs` file for each word/phrase defined from now
- `.ok` accept default answers for the rest of this list of items
- `.pro` enter Prolog break loop (type end of file to exit it)
- `.q` quit VEX/QVEX
- `.sw` scrap word or phrase currently being defined
- `.uap` use (consider) all paradigms together
- `.ucp` use (consider) common paradigms before uncommon ones
- `.uuf` append future entries directly to the `userimp` file

as well as some information on the normal CLE help commands, which are (like all CLE commands) also available here. This tells you that the the response `.sw` will cause VEX to backtrack to the point where it asks for a word or phrase to be defined.

Normally, a backtracking request will cause VEX to undo all the work it has done since you entered the erroneous input. This is necessary because the questions asked at any point in general depend on the answers to previous questions. However, inflections (plurals, tenses, etc.) can be corrected without undoing any work; see below for other such commands.

Because of the structure of the program and the actions carried out at different times, it is not possible to backtrack to any previous point. For example, if you are defining "use" as a noun having already defined it as a verb, you cannot backtrack into the sort dialogue for the verb use. However, the range of options available should be enough to allow convenient use.

For reference, the full set of backtracking options available at various points are as follows. An asterisk indicates that the option is performed without actually backtracking. As well as the backtracking commands, all CLE commands are also available (see Section B.2).

- `.ac` abort this category, start on next if any.
- `.anc` ask for new categories (noun, verb etc.).
- `.asr` abort sort rule currently being defined.
- `.bt` backtrack to preceding (or recent) input.
- `.dr` backtrack to deciding on having relational predicate.
- `.ds` display sentence and logical form fragment.*
- `.eg` list some existing examples of the current paradigm.*
- `.eg P` list the sort input for predicate P.*
- `.gc` ask for fresh global comment.*
- `.gr` ask which sentences are grammatical again.
C.9 Setting up VEX for your application

If you wish to adapt VEX to your application domain, because the sublanguage used in that domain is significantly different from standard written English, you may want to modify the set of sentence patterns that correspond to each paradigm entry, or to add new paradigm entries. This will require altering several predicates in vexpats.pl, following the guidelines in that file. Adding new paradigms will also require making additions to the test-and-development lexicon files testsyncats.pl and testsenses.pl and the implicit paradigm file paraimp.pl, and then remaking the paradigm lexicon files using .mpl. These operations require a thorough understanding of the structure of both the lexicon entries themselves and of the criteria used to select paradigm codes (see the comments in vexpats.pl for the latter).

The Prolog predicates diagnose, encountered earlier, and paradigm_instances are provided to help debugging in this process. They are described above their definitions in vexdiag.pl.

If you find a word behaviour that VEX does not cover, you should decide whether than behaviour is exhibited by a reasonably large or open-ended set of words, or whether it is confined to a small set which you can enumerate. In the former case, you may want to add a new paradigm, as above. In the latter case, it is better to define the relevant entries by hand in funsyncats.pl, funcsenses.pl and funcsorts.pl. Existing entries in the test lexicon files are usually a good starting point for doing this. Remember to take into account any existing entries for the words whose behaviour you are trying to capture.

Before using VEX to define words in an application domain, you will also need to adapt the sort hierarchy. This hierarchy will need to be entered in the file given by the setsys parameter hierarchy.
Appendix D

Examples

This appendix contains some examples of translations taken from the test suite. All examples shown are in the Swedish → English direction, so that non-Swedish readers can judge the quality of the output for themselves; the final sentences are chosen to illustrate the most frequent types of mistranslation produced by the system. Comments are in italics.

Kortnumret är "ABC 123".
→ The card number is "ABC 123".

Jag heter Manny Rayner.
→ I am called Manny Rayner.

Jag åker 91/2/15.
→ I travel on 15/2/91.

Jag tror att den kostar $20.
→ I think it costs $20.

Han flyger 15:30.
→ He is flying at 15:30.

Det finns 24 kontor i Sverige.
→ There are 24 offices in Sweden.

Vi har två röda Saabar.
→ We have two red Saabs.

Är bilen ny?
→ Is the car new?

Jag tror inte att det finns någon Volvo.
→ I do not think that there is a Volvo.

Han har åkt till Göteborg.
→ He has gone to Gothenburg.

Vi har ingen Audi.
→ We have no Audi.
Får jag rekommendera dig att ta Volvon?
→ May I advise you to take the Volvo?

Jag kan inte hitta någon annan bil.
→ I can not find another car.

Jag ska åka dit imorgon.
→ I will go there tomorrow.

Kommer du att betala med kreditkort?
→ Will you pay by credit card?

Jag tänker åka till Göteborg på onsdag.
→ I intend to go to Gothenburg on Wednesday.

Jag tror inte att det går att betala med kort.
→ I do not think it is possible to pay by card.

Det gäller att dyka upp då.
→ It is important to turn up then.

Heter du Hiyan Alshawi?
→ Are you called Hiyan Alshawi?

Vad sa du att du hette?
→ What did you say you were called?

Han ska ringa oss nu.
→ He should ring us now.

Har du ringt honom?
→ Have you rung him?

Den kostar $25.
→ It costs $25.

Jag har det inte här.
→ I do not have that here.

Jag tror att de stänger klockan 17:00.
→ I think that they are closing at 17:00.

Jag kan inte hitta dem.
→ I can not find them.

Regnar det i London?
→ Is it raining in London?

Han kommer klockan två.
→ He’s coming at 2:00.

Hans bil var en Saab.
→ His car was a Saab.

Vad hette killen som du talade med?
→ What was the man that you talked to called?
Vart ska du åka då?
→ Where will you go then?

Snöar det där i London?
→ Is it snowing there in London?

Jag reser dit på fredag.
→ I am travelling there on Friday.

Jag vill hyra en bil.
→ I want to hire a car.

Vi har ett kontor vid flygplatsen.
→ We have an office at the airport.

Kommer de idag?
→ Are they coming today?

Har du någon försäkring?
→ Do you have any insurance?

Alla bilarna är nya.
→ All the cars are new.

Vi har tyvärr ingen Volvo.
→ We have no Volvo.

"Tyvärr" ("unfortunately") is discarded.

Har du fått mina böcker?
→ Have you got my books?

"Received" would be better.

Han skulle åka på tisdag klockan tre.
→ He would go on Tuesday at 15:00.

"Was going to go" would probably be better.

Ska ni åka från Arlanda?
→ Will you go from Arlanda?

"Will you be coming" would be better.

Jag tror att han larvar sig.
→ I think he is silly.

"I think he is being silly" is preferable.

Jag tar den gula Volvon.
→ I take the yellow Volvo.

"I will take" is better, but to get this the system must find out that the Swedish present is being used as a future here.

Han gillar inte Saabbar.
→ Saabs are not liked.

The passive is incorrectly produced first. "He does not like Saabs" would be better.

Vilken bil ville du ha?
→ Which car was wanted by you?
Again, the passive is incorrectly generated first. This should be "Which car did you want".

Det nya kontoret ligger vid Arlanda.
→ There is the new office at Arlanda.
This should really be "the new office is at Arlanda", but both sentences currently receive the same QLF.

Jag har inte pratat med någon där.
→ I haven’t talked to a person there.
Instead of "a person", "anyone" would be preferable.

Har det hänt något?
→ Has a thing happened?
Similarly, "something" would be better than "a thing".
Appendix E

The English and Swedish lexicons

This final appendix contains the listings of the English and Swedish lexicons. Each entry is shown, together with its part of speech and a logical constant taken from its sense entry. For most content words, this is the name of the associated predicate. Note that adjectives always have multiple entries, each "normal" entry being paired with a similar one where the predicate name is suffixed with the string _degree. These correspond to the absolute and relative forms of the adjective, which the grammars treat as distinct. Note also that the Swedish lexicon includes all the affixes used in inflectional morphology.
asleep: adj
asleeml
asleeml_degree
association: nbar
association_humgr
assume: v
assume_that
at: p
at_location
at_rate
at_time
attack: nbar
attack1
attack1_of
attack: v
attack_2p
attack_3p
attempt: nbar
attempt
attempt1_of
attempt: v
attempt_something
attempt_to_do
audi: name
audi: nbar
audi_cartype
audience: nbar
audience_humgr
aug: month
aug: n
aug: month
aug: n
aug: name
aug: n
aug: month_noun
automatic: adj
automatic механически
automatic_mechanically_degree
e
automatic: nbar
automatic
autumn: nbar
autumn
available: adj
available1
available1_degree
average: adj
average_Commonplace
average_Commonplace_degree
avoid: v
avoid_doing
avoid_obstacle
away: p
axle: nbar
axle_carpark
ba: name
baby: nbar
baby1
back: nbar
back1
back1_of
back_bodypart
back: p
bad: adj
bad1
bad1_degree
bag: nbar
bag_object
ball: nbar
ball_object
bank: nbar
bank_building
bank_of_river
bank_organisation
base: v
base_on
basis: nbar
basis1
basis1_of
battle: nbar
battle1
bay: nbar
bay1
bci: nbar
bcii
be: v
bear: nbar
bear_animal
bear: v
bear_annoynce
bear_carry
bear_give_birth
bear_to_go
beast: nbar
beast1
beautiful: adj
beautiful
beautiful1
beautiful1_degree
because: p
reason_for
become: become+ed
become: v
become3p
bed: nbar
bed1
been: v
before: advp
before: p
before_proposition
before_time
begin: v
begin2p
begin3p
begin_to_do
beginning: nbar
beginning1
beginning1_of
behind: p
behind_location
being: v
being1
believe: nbar
believe1
believe: v
believe1ini
believe_prop
believe_someone
believe_someone_that
believe_something
bell: nbar
bell1
belong: v
belong_to
below: p
below_location
beneath: p
beneath_location
bentley: name
bentley: nbar
bentley_cartype
beside: p
beside_location
bet: bet+ed
bet: v
bet_that
between: p
between_location
between_time
beyond: p
beyond_location
big: adj
big
big1_degree
bill: name
bill: nbar
bill_of_money
bill_physical
biology: nbar
biology1
bird: nbar
bird1
birmingham: name
bishop: nbar
bishop
bit: nbar
bit1
bit1_of
bit_measure
black: adj
black_in_colour
black_in_colour_degree
blood: nbar
blood1
blue: p
blue
blow: nbar
blow_chump
blow: v
blow2p
blow3p
blue: adj
blue_in_colour
blue_in_colour_degree
board: nbar
board_humgr
boat: nbar
boat1
body: nbar
body_humgr
body_of_animate_being
bone: nbar
bone1
bonnet: nbar
bonnet_carpark
book: nbar
book1
booking: nbar
booking1
boot: nbar
boot_carpark
borrow: v
borrow_object
both: conj
and1
both: det
reduce: v
reduce_to1
refer: v
refer_to_2p
refer_to_3p
refer_to_4p
refer_to_do
regard: v
regard_as1
registration: nbar
registration1
relate: v
relate_to_connect
relation: nbar
relation1
relation1_of
relay: nbar
relay1
religion: nbar
religion1
religion1s
adj
gradedreligion1
religious1
rally: v
rally_on1
remain: v
remain_2p
remark: v
remark_that
remember: v
remember_3p
remember_doing
remember_prop
remember_to_do
remember_whether
remind: v
remind_someone_that
remind_to_do
rent: nbar
rent1
rent1_of
rent: v
rent_3p
rental: nbar
rental1
repeat: v
repeat_obj
repeat_prop
reply: v
reply_that
reply_to_person
reply_to_that
report: nbar
report1
report: v
report_that
report_to_3p
report_to_that
represent: v
represent_3p
representative: nbar
representative1
representative1_of
republic: nbar
republic_place
require: v
require1
research: nbar
research1
reservation: nbar
reservation1
reserve: v
reserve_3p
rest: nbar
rest1
rest1_of
restaurant: nbar
restaurant_building
result: nbar
result1
result: v
result_from1
result_int
return: nbar
return1
return1_of
return: v
return_3p
return_to_give_back
return_to_go_back
rich: adj
rich_in
rich_in_degree
rich_wealthy
rich_wealthy_degree
ride: v
ride_2p
ride_3p
right: right+ly
right: adj
right_not_left
right_person
right_person_degree
right_prop
right_prop_degree
ring: nbar
ring_for_finger
ring: v
ring_2p
ring_3p
rise: v
rise_2p
river: nbar
river1
road: nbar
road1
rock: nbar
rock_stone
roof: nbar
roof_carpart
room: nbar
room_place
round: p
round_location
route: nbar
route1
rule: nbar
rule1
rule1_of
rule_law
run: runved
run: v
run_3p_operate
run_dash
run_function
saab: name
saab: nbar
saab_cartype
safe: adj
safe_from1
safe_from1_degree
safe_to_use
safe_to_use_degree
safe_to_use_property
safe_to_use_proposition
sailor: nbar
sailor1
salary: nbar
salary1
salary1_of
sales: nbar
sales1
salary1
same: adj
sand: nbar
sandal
satisfactory: adj
satisfactory1
satisfactory1_degree
satisfactory1_property
satisfactory1_property_degree
satisfactory1_proposition
satisfactory1_proposition_degree
gree
satisfy: v
satisfy_3p
satisfy_3p_property
satisfy_3p_proposition
saturday: name
saturday: nbar
saturday_day_noun
save: v
save_2p
save_2p
save_board
save_rescue
say: v
say_show_3p
say_show_prop
say_show_what
say_speak_3p
say_speak_prop
say_speak_what
say_to_what
scene: nbar
scene1
scene1_of
scene_of_play
school: nbar
school_building
science: nbar
science1
scientific: adj
scientific1
scientific1_degree
scientist: nbar
scientist1
scissors: nbar
seal
sea: nbar
season1
season:
ora: possdet
ert: possdet
ett: det
1
en
ett: nbar
ett: number
1
eurohire: name
eurohire_name
faktura: nbar
fakturai
fall: nbar
fall1
fall: v
fallas_med
far: nbar
far1
farfar: nbar
farfar1
farfar/morfar: nbar
farmor: nbar
farmori
farmor/mormor: nbar
fattig: adj
fattigi
fattigl_degree
februari: name
februari_nam[p]_nmand
februari: nbar
februari_month_noun
februari: month
2 month
fem: det
5
fem: number
5
femte: ordinal
5 ordinal
fento: det
50
femto: number
50
femtonde: ordinal
50 ordinal
femton: det
15
femton: number
15
femtonde: ordinal
15 ordinal
ferrari: name
ferrari_name
ferrari: nbar
ferrari_cartype
fiat: name
fiat_name
fiat: nbar
fiat_cartype
fik: nbar
fik_st[1le]
finn: v
finnas1
fixa: v
det_fixar_sig
fjorton: det
14
fjorton: number
14
fjortonde: ordinal
14 ordinal
fjorde: ordinal
4 ordinal
fler: det
comp
flyg: v
flyga_2p
flygplan: nbar
flygplan
flygplatser: nbar
flygplats_st[ile]
folk: nbar
volkswagen_cartype
folkvagn: nbar
volkswagen_cartype
ford: name
ford_name
ford: nbar
ford_cartype
fordon: nbar
fordon1
fort: adj
fort1
forti_degree
forts(st: v
forts[sta_2p
forts[sta_att_gir/lra
fram[lr: p
framfl_rum
fransk: adj
franski
franski[degree
fransman: nbar
fransman
fredag: name
fredag_name[veckodal
fredag: nbar
fredag_day_noun
fredag: name
fredags_name_p[veckodal
frukost: nbar
frukost
fr[ga: nbar
frgai
fr[ga: v
fr[ga_neg_vad
fr[n: p
frn_riktning
frn_tid
fungera: v
fungera_2p
fyra: det
4
fyra: number
4
fyrtio: det
40
fyrtio: number
40
fyrtionde: ordinal
40 ordinal
f[rdig: adj
fr[di
fr[rdigi
fr[rdigi_degree
f[re: det
comp
filj: v
filja_after
filja_med
finster: nbar
finsteri
fri: p
fri_belning
fri_nng
fri_till[rande
frbi: p
frbi_rum
fri: p
fire_rum
fire_tid
f[rasl: v
f[rasl_att
f[rest[li: v
f[rest[li_att
f[ir: d
f[ir[ra[att
f[ir[ra[att
f[ir[rdig
f[rista: ordinal
1 ordinal
f[rist: v
f[rist[2p
f[rist_att
f[rist_ngt
f[rist[ad
f[ris[kra: v
f[ris[kra[att
f[ris[kra[att
f[ris[ring: nbar
f[ris[ring_penger
f[ris[k: v
f[ris[ka[gra
f[ris[vta: v
f[ris[vta[att
f: det
ratio
f: v
f_kan
f_mate
f_ngt
gambck: name
gambck_name
gammaxl: adj
gammaxl
gammaxl[degree
ganska: degree
gata: nbar
gata_st[ile
gativck: name
gativck_name
ge: v
g_e_4p
genom: p
genom_att
genom_instrument
genom_riktning
genom_tid
gift: v
gifta[med
gilla: v
gilla_3p
glm: v
glimma_bort_att
glimma_bort_gra
glimma_bort_ngt
glimma_bort_vad
godtag: interjection
gratis: adj
gratios
gratis1_degree
grund: nbar
grund1
gru: adj
gru
gru1_degree
gr: adj
gru1
gru1_degree
gui: adj
gui
gui1_degree
gui1: v
det_gullor_att
gulla_2p
gir: advp
gir: v
girra_sig_till
giteborg: name
giteborg_name
gj: v
det_gj_r_att
gjng: nbar
gng_tid
ha: v
hall: interjection
halv: halv
halva: det
han: n
hans: possdet
harkla: v
harkla_sig
hata: v
hata3p
heathrow: name
heathrow_name
hej: interjection
helg: nbar
helg_measure
helgdag: nbar
helgdag_tid
helst: advp
henne: np
hon
henner: possdet
hertz: name
hertz_name
heta: v
heta1
hit: pp
till_mj1
hitta: v
hitta3p
hiyan: name
hiyan_name
hjul: nbar
hjul1
hjlp: v
hjopa_till
hon: np
honom: np
han
hoppa: v
hoppas_att
hos: p
hos_n
huber: name
huber_name
hur: degree
dummy_if
mycket
hur: det
hur: pp
manner_of
huruvida: complementiser
complementiser
hva: nbar
hvis
hyr: v
hyra3p
h(cta)
hd3p
hd: v
det_hnder_att
det_hnder_ngt
hr: pp
i_umlaut
hrifnrn: pp
ifr3hilm
hrinne: pp
inom_run
hig: higt-umlaut
hig: adj
hig1
hig_degree
h: v
hira3p
hira_att
hira_hnta
hira_vad
hrd: adj
hrdi
hrd1_degree
i: multip
i_och_med
i: p
i_dtid
i_ritkning
i_run
i_tidsperiod
ibland: advp
gng_tid
icke: neg
idge: pp
pl_tid
ifr: p
ifr3n_ritkning
ifr1n_tid
igenom: p
igenom_ritkning
ig: pp
pl_tid
ih: p
imorgen: pp
pl_tid
inbilla: v
inbilla_sig_att
informera: v
rd: adj
rd1
d1_degree
rd: v
rd_3p
rkav: v
rkav_ut_flr
saab: name
saab_name
saab: nbar
saab_cartype
samma: adj
se: v
se_3p
se_att
se_hnda
se_ut
se_vad
sedan: p
sedan_proposition
sedan_tid
sen: advp
september: month
9 month
september: name
september_name_p_mnad
september: nbar
september_month_noun
sex: det
6
sex: number
6
sextio: det
60
sextio: number
60
sextiode: ordinal
60 ordinal
sextion: det
16
sextion: number
16
sextiode: ordinal
16 ordinal
sics: name
sics_name
siemens: name
siemens_name
siffra: nbar
siffrai
sig: np
reflexive
sin: possdet
sina: possdet
sist: adj
sisti
sisti_degree
sitt: possdet
sju: det
7
sju: number
7
sju: adj
sjuki
sjuki_degree
sjunde: ordinal
7 ordinal
sjutio: det
70
sjutioend: ordinal
70 ordinal
sjutio: det
17
sjutio: number
17
sjutioend: ordinal
17 ordinal
sj(tte): ordinal
6 ordinal
 ska: v
skal
skada: nbar
skadai
ske: v
skriva_att
skriva_ngt
skicka: v
skicka_3p
skicka_4p
skriv: v
skriv_2p
skriv_4p
skriv_ngt
skriv_prop
skulle: v
skulle1
skyldig: adj
skyldig_nmg_ngt
skyldig_nmg_ngt_degree
sk[e]: v
sk[mas_2p]
sk[mas_iver_att]
sk[mas_iver_vad]
sk[pbil]: nbar
sk[pbil]
slut: adj
slut_borta
slut_borta_degree
sluta: v
sluta_2p
sluta_gra
sluta_med_att_gra
sl[p: nbar
sl[pi]
sl]: v
sl[up]
smutsa: v
smutsa_med_3p
smutsig: adj
smutsig1
smutsig1_degree
snabb: adj
snabb1
snabb1_degree
snarka: v
snarka_2p
snart: advp
sn[a]: v
sn[a]_sal
som: comp_suffix
som: np
som: p
som_proposition
som_roll
sort: nbar
sort1
spela: v
spela_3p
spela_5p
spryk: nbar
spr[ki]
sri: name
sri_name
stackars: adj
stackars1
stackars1_degree
statt: nbar
statt_st1le
stark: adj
stark1
stark1_degree
steve: name
steve_name
stockholm: name
stockholm_name
stor: adj
stort
stort1_degree
stra[astare: nbar
stra[astare1
str[l]: v
str[l]_nmg_ngonstans
str[l]: nbar
str[l]1
str[l]2
str[n]: v
str[n]_2p
str[n]_3p
str[n][l]: v
str[n]_gonstans
str[n]: skrivet
svar: nbar
svar1
svara: v
svara_2p
svarat: adj
svarat1
svarat1_degree
svensk: adj
svenski
svenski_degree
svensk: nbar
svensk_medborgare
svenska: name
svenska_name
sverige: name
sverige_name
svr: adj
svr1
svr1_degree
svr1_property
svr1_proposition
svg: v
svg_nmg_att
svg_nmg_vad
svg_nmg_ngn
svg_att
svg_ngt
svg_vad
svkort: advp
svrklind: adv
svrklind1
svte: nbar
trols.proposition
tv: det
2
tv: number
2
tyck: v
tycka: adj
tycka_att
tycka_om

...
vart: pp
till_m)l
vauxhall: name
vauxhall_name
vauxhall: nbar
vauxhall_cartype
vecka: nbar
vecka_tid
veckoslut: nbar
veckoslut_tid
vem: np
vens: possdet
verka: v
verkal
vet: v
veta_att
veta_vad
vi: np
vid: p
vid_rum
vid_tid
vik: v
vika_ut_sig
viktig: adj
viktig
viktig_property
viktig_property_degree
viktig_proposition
viktig_proposition_degree
villja: villja=INF
villja: v
villja_att
villja_gira
vilk: det
vilk: np
vilen: det
viken: np
viket: det
viket: np
vindruta: nbar
vindrutal
vinn: v
vinn_3p
visa: v
visal
visa_ngn_vad
visse: adj
visse
vic: adj
vit
vit_degree
volkswagen: name
volkswagen_name
volkswagen: nbar
volkswagen_cartype
volvo: name
volvo_name
volvo: nbar
volvo_cartype
v{g: nbar
vg_stille
v{l: advp
v{nts: v
v{nts_2p
v{nts_p_att
v{nts_p_ngt
v{rd: adj
v{rd_ngt_degree
v{rd_ngt_degree
v{ster}s: name
v{ster}s_name
v{exl: nbar
v{exl}
v{exl}da: nbar
v{exl}da
v{r}: possdet
v{ra: possdet
v{rt: possdet
n: comp_suffix
(t: v
(t_3p
(ta_2p
(ta: v
ka: v
ka_2p
ka_med_ngt
lpen: adj
lpen1
lpen1_degree
lpena: v
lpena_2p
lpena_3p
lver: p
tvrs|ver_rum
ver_rum
vrig: adj
vrig
x: v
ka_2p
ka_ngt
(r: nbar
(r_tid
(t: t--umlaut
(t: p
(t_ngn
(t_riktning
tero(mna: v
tero(mna_3p
(tta: det
(tta: number
(tta: number
(tta: number
(tta: number
80
(tta: number
80
(ttiode: ordinal
80
(ttiode: ordinal
80
(ttiode: ordinal
80
(ttiode: ordinal
80