Using the Framework

The FRACAS Consortium

Robin Cooper, Dick Crouch, Jan van Eijck,
Chris Fox, Josef van Genabith, Jan Jaspars, Hans Kamp,
David Milward, Manfred Pinkal, Massimo Poesio, Steve Pulman

*With additional contributions from:*
Ted Briscoe, Holger Maier and Karsten Konrad

FraCaS

A Framework for Computational Semantics
LRE 62-051

Deliverable D16 — Final Draft

January 1996
A Framework for Computational Semantics

CWI Amsterdam
University of Edinburgh
Centre for Cognitive Science and Human Communication Research Centre

Universität des Saarlandes
Department of Computational Linguistics

Universität Stuttgart
Institut für Maschinelles Sprachverarbeitung

SRI Cambridge

For copies of reports, updates on project activities and other FRACAS-related information, contact:

The FRACAS Project Administrator
University of Edinburgh
Centre for Cognitive Science
2 Buccleuch Place
Edinburgh EH8 9LW, Scotland, UK
fracas@cogsci.ed.ac.uk

Copies of reports and other material can also be accessed via anonymous ftp from ftp.cogsci.ed.ac.uk, directory pub/FRACAS.
©1994, The Individual Authors

No part of this document may be reproduced or transmitted in any form, or by any means, electronic or mechanical, including photocopy, recording, or any information storage and retrieval system, without permission from the copyright owner.
Contents

1 The Relevance of Semantics for Commercial Applications 10

2 The Framework Tools 13

2.1 The CLEARS User Manual ........................................ 13

2.1.1 Introduction ..................................................... 13

2.1.2 Installation ..................................................... 14

2.1.3 Semantic Composition at your fingertips ..................... 19

2.1.4 Pull-Down Menus ................................................. 28

2.1.5 Pop-Up Menus .................................................... 34

2.1.6 Selected Topics .................................................. 36

2.1.7 Appendices ........................................................ 40

2.2 Extending Clic with Interaction and User Defined Graphics ........ 42

2.2.1 Interaction ....................................................... 44

2.2.2 Adding graphical objects ....................................... 45

2.3 EKntool User Manual ............................................... 54

2.3.1 Introduction ..................................................... 54

2.3.2 Interacting with ekntool ........................................ 54

2.3.3 The Main Grammar .............................................. 56
3 A Semantic Test Suite

3.1 Generalized quantifiers .................................................. 65
  3.1.1 Conservativity ....................................................... 65
  3.1.2 Monotonicity (upwards on second argument) ..................... 67
  3.1.3 Monotonicity (downwards on second argument) .................. 69
  3.1.4 Monotonicity (upwards on first argument) ....................... 71
  3.1.5 Monotonicity (downwards on first argument) ..................... 74

3.2 Plurals ................................................................. 76
  3.2.1 Conjoined Noun Phrases ............................................ 76
  3.2.2 Definite Plurals .................................................... 77
  3.2.3 Bare Plurals ....................................................... 79
  3.2.4 Dependent Plurals .................................................. 80
  3.2.5 Negated Plurals .................................................... 80
  3.2.6 Collective and Distributive Plurals ................................ 81

3.3 (Nominal) Anaphora ..................................................... 81
  3.3.1 Intra-Sentential .................................................... 82
  3.3.2 Inter-Sentential .................................................... 83
  3.3.3 Plural Anaphora .................................................... 83
  3.3.4 E-type and Donkey Pronouns ...................................... 85
3.3.5 Functional and Subsectional ........................................ 86
3.3.6 Simple Reflexives ................................................. 86
3.4 Ellipsis ................................................................. 87
  3.4.1 VP Ellipsis .......................................................... 87
  3.4.2 Gapping .............................................................. 88
  3.4.3 One Anaphora ...................................................... 89
  3.4.4 Sluicing ............................................................. 91
  3.4.5 Phrasal Ellipsis .................................................... 91
  3.4.6 Antecedent Contained Deletion ................................. 92
  3.4.7 Configurational Effects .......................................... 93
  3.4.8 Ellipsis and Anaphora .......................................... 94
  3.4.9 Ellipsis and Quantification .................................... 96
3.5 Adjectives ............................................................. 97
  3.5.1 Affirmative and Non-Affirmative ............................... 97
  3.5.2 No Comparison Class ............................................ 98
  3.5.3 Opposites ......................................................... 98
  3.5.4 Extensional Comparison Classes ................................. 99
  3.5.5 Extensional and Intensional Comparison Classes ............ 100
  3.5.6 Default Comparison Classes .................................... 101
3.6 Comparatives .......................................................... 101
  3.6.1 Phrasal Comparatives ............................................ 101
  3.6.2 Clausal Complement .............................................. 104
  3.6.3 Measure Phrases .................................................. 104
  3.6.4 Differential Comparatives ...................................... 104
3.6.5 Attributive Comparatives ........................................... 105
3.6.6 Comparatives and Quantifiers ....................................... 105
3.7 Temporal Reference .................................................. 106
  3.7.1 Standard Use of Tenses .......................................... 106
  3.7.2 Temporal Adverbials .............................................. 107
  3.7.3 Anaphoric Dimension ............................................. 114
  3.7.4 Adverbs of Quantification ....................................... 115
  3.7.5 Some more Complex Examples .................................. 115
3.8 Verbs ................................................................. 117
  3.8.1 Aspectual Classes ................................................ 117
  3.8.2 Distributive and Collective Predication ....................... 118
3.9 Attitudes ............................................................ 118
  3.9.1 Epistemic, Intentional and Reportive Attitudes .............. 118
  3.9.2 Preceptive Attitudes: “See” with Bare Infinitive Complements .............. 119

4 Computational Lexical Semantics in Review ........................................ 121
  4.1 Introduction .......................................................... 121
    4.1.1 Argument Structure .......................................... 122
    4.1.2 Event Structure ................................................. 123
    4.1.3 Qualia Structure ............................................... 124
    4.1.4 Inheritance Structure ......................................... 125
  4.2 Lexical Semantics in Acquilex .................................... 126
    4.2.1 Typed Default Feature Structures ........................ 126
    4.2.2 Systematic Polysemy ......................................... 127
4.2.3 Semantic/Pragmatic Interactions ........................................ 129

4.3 Conclusions ................................................................. 131

5 Evaluation of the Framework: further implementation and extension 132
Using the Framework

The FRACAS Consortium

Robin Cooper, Dick Crouch, Jan van Eijck,
Chris Fox, Josef van Genabith, Jan Jaspars, Hans Kamp,
David Milward, Manfred Pinkal, Massimo Poesio, Steve Pulman

With additional contributions from:
Ted Briscoe, Holger Maier and Karsten Konrad

Abstract

There are two major levels of processing that are significant in the use of a computational semantic framework: semantic composition for the construction of meanings; and inference either for the exploitation of those meanings, or to assist in determining contextually sensitive aspects of meaning. The first chapter of this deliverable outlines the semantic competences that are either required by — or would improve the performance of — a variety of different commercially relevant applications. Semantic composition and meaning construction is a core competence required by all applications, and inference is central to many. The second chapter takes the form of a manual describing the use of a program library and educational/research tool providing a concrete computational framework bringing together different types of semantic composition. The third chapter presents an inference test suite for evaluating the inferential competence of different NLP systems and semantic theories. Providing an implementation of the inference level was beyond the scope of FraCaS, but the test suite provides the basis of a useful and theory/system-independent semantic tool. The fourth chapter gives an overview of the state-of-the-art in computational lexical semantics. A considerable amount of time on the project was devoted to discussing lexical semantics, and revealed that the computational and formal semantic traditions are not as far apart as might first be assumed from the literature. Lexical semantics is a major semantic competence, and has an impact both on semantic composition and on inference. A deeper understanding of lexical issues is required even to separate out in a useful way those inferences in the test suite that hold for structural reasons and those that hold for lexical reasons. The final chapter discusses significant themes on computational semantics, and further directions for extending the frameworks developed during FraCaS. At present this chapter is being circulated in draft form to the project reviewers, and will be included in the final version of this deliverable after it is discussed at the FraCaS final review.
Chapter 1

The Relevance of Semantics for Commercial Applications

It is difficult to say very much of detail about the usefulness of semantics for practical applications, because the state of the art is such that there are no commercial applications of any type which involve a substantial semantic component. There are even very few research prototypes with any degree of semantic coverage. Thus the following paragraphs are more in the way of informed guesswork rather than hard fact: we attempt to list some of the main applications of NLP and say to what extent (if at all) semantics is already involved, and whether performance might be improved if it were.

To begin with, it is useful to distinguish three types of semantic processing that a system might undertake. Firstly, we distinguish lexical semantic processing, by which we mean processing that involves the meanings of words in some way. This might be simply by distinguishing between different senses of words, or positioning words in a semantic domain or hierarchy via a thesaurus, through to associating words with meaning postulates in order to be able to describe various complex lexically-based inferences etc. The second type of processing we will describe as 'structural semantics', by which we mean the process of recovering whatever information we can about the context-independent meaning of a sentence, on the basis of the words in it, their meanings, syntactic structure, and whatever semantic operations or properties may be detected on a purely linguistic or structural basis. The third type of processing we describe as 'contextual processing', namely the recovery of whatever context-dependent elements of meaning are necessary to understand the sentence.

Clearly there is a logical dependency between these types of processing, and equally clearly they make increasingly greater demands on system builders. Contextual processing in particular involves non-linguistic reasoning of various sorts, and as has been stressed in other FraCaS documents, currently represents one of the hardest research challenges in NLP.

We now briefly list some typical current and future NLP applications, and say which types of semantic processing they already use (in at least some instances), which types they could use
with profit, and which types of semantic processing are, in our view, absolutely necessary if satisfactory performance is ever to be achieved.

1. **Grammar correction and spell checking**: This might be in the context of a simple editor or word-processing package, or in the context of a document composition system which was attempting to enforce a ‘house style’ on the writer. There are probably some cost-benefit calculations to perform before adding semantic processing of any kind to such systems, but it is likely that detection of unintended ambiguities, and some kinds of error, could be performed more accurately with some lexical and structural semantic processing.

2. **Structured document composition**: By this we mean the next stage of sophistication from the preceding kind of system: for example, a tool for writers producing technical documentation, system specifications, or even legal documents. The system would carry out consistency checks on terminology, detect possible new technical terms, enforce the use of definitions at appropriate points, and in general be aware of the intended structure and form of the document. Detection of possible technical terms can be done on a purely syntactic basis, but is more accurate if some lexical semantics can also be used. Keeping track of definitions requires being able to relate different syntactic forms to the same concept.

3. **Information extraction**: This includes simple applications like message routing, through to template or form-filling from free text like newspaper or intelligence reports. Current systems usually work in domains too restricted for lexical ambiguity to be a problem, and most things that look ‘semantic’ are actually done by (hand-crafted) pattern matching. But accuracy, portability, and the ability to cope with different domains simultaneously, can be better achieved by doing at least lexical and structural semantics. It is also the experience of many participants in the MUC projects that contextual processing, especially anaphora resolution, improves performance considerably in some aspects of the task, even where the contextual processing itself might be done using rather minimal semantic apparatus.

4. **Interactive translation and machine aided translation**: This is intended to include systems which involve a dialogue (perhaps a very restrictive one) between a person and a machine (to arrive at a translation) or between two people (to solve a problem). The key point is that there is the possibility to fall back on a user to supply information needed to make translation decisions. Like all translation, interactive translation requires disambiguation or translation decisions that are based on lexical or structural semantic factors. Contextual processing is not absolutely necessary if translation proceeds on a sentence by sentence basis, for there is always the human agent to fall back on. But if it is available and sufficiently accurate, interactions are less complex and more natural. If the interactive translation is within a dialogue context then clearly at least some degree of contextual processing is usually necessary.

5. **Offline translation**: Usually large scale translation of texts, not involving human intervention (at least, not in the part of the process we are interested in: there may be pre- or post-editing involved in the overall system). As is well known, high quality translation between many language pairs (e.g., Japanese to English, although less true in the reverse direction) can only be achieved if contextually determined properties like (zero) anaphoric reference and definiteness can be satisfactorily resolved.
6. **Interfaces to information systems:** E.g. database query; front ends to advice giving or order processing systems; control of equipment. We assume that these will often be spoken language systems. Interfaces are generally intended to support dialogues and anaphoric or other reference to previous utterances or system output arise quite naturally and must be supported. Furthermore, relating linguistic content to e.g. a database model requires reasoning on the basis of a fully articulated representation of the meaning of the utterance.

7. **Text generation:** Systems that turn non-linguistic representations into sentences. Examples would be financial report generators; those that report on the running status of some piece of equipment; etc. Text generation almost by definition involves taking some meaning representation (whether linguistic or otherwise) and producing sentences with the same meaning. Thus at the very least an explicit lexical and structural semantic model must be employed, as well as syntax and morphology. If whole texts are being produced then sophisticated contextual processing is required to ensure that they are coherent and natural, rather than just sequences of independent sentences.

8. **Text to speech:** Speech synthesis either from free text or structured linguistic input. Properties like information structure (given/new, focus, etc.) are major determinants of intonation. To be able to produce fully natural sounding speech, a text to speech system should be able to determine these properties, although as yet only a few small scale experimental systems do so. In general, you need to understand the meaning of a sentence if you are to be able to speak it naturally: this is true just as much for machines as for humans.

In the following table we summarise these remarks about the current state of play with respect to whether at least some (usually not all) existing systems already use lexical (L), structural (S), or contextual (C) semantic processing. In the third column, we indicate which levels of processing would if also carried out improve performance (we assume) in these systems. (Of course, we assume that improved rather than additional semantic processing would always improve task performance.) In the last column we indicate whether (in our view) a system performing this task can ever achieve a satisfactory level of performance if it does not carry out this kind of semantic processing.

<table>
<thead>
<tr>
<th>Application</th>
<th>already uses</th>
<th>could use</th>
<th>must use</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Grammar correction</td>
<td>L</td>
<td>L S</td>
<td></td>
</tr>
<tr>
<td>2. Document composition</td>
<td>L S C</td>
<td>L S C</td>
<td></td>
</tr>
<tr>
<td>3. Information extraction</td>
<td>L S</td>
<td>L C</td>
<td>L</td>
</tr>
<tr>
<td>4. Interactive translation</td>
<td>L S</td>
<td>L C</td>
<td>L S</td>
</tr>
<tr>
<td>5. Offline translation</td>
<td>L S C</td>
<td>L S C</td>
<td>L S C</td>
</tr>
<tr>
<td>6. Information interfaces</td>
<td>L S C</td>
<td>L S C</td>
<td>L S C</td>
</tr>
<tr>
<td>7. Text generation</td>
<td>L S C</td>
<td>L S C</td>
<td></td>
</tr>
<tr>
<td>8. Text to speech</td>
<td>L S C</td>
<td>L S C</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 2

The Framework Tools

2.1 The CLEARS User Manual

2.1.1 Introduction

The CLEARS system (Computational Semantics Tool for Education and Research in Semantics) was developed as part of the FraCaS (Framework for Computational Semantics) project which is designed to encourage convergence between different semantic formalisms.¹ There are currently a wide variety of semantic formalisms used in teaching and research, for example, various versions of 'Montague-Grammar', DRT, Situation Semantics and Property Theory. As different as they look on first sight, they share many common assumptions, and provide similar treatments of many phenomena. The tool allows exploration and comparison of these different formalisms, enabling the user to get an idea of the range of possibilities of semantic construction. It is intended to be used as both a research tool and a tutorial tool. In this paper we describe how to use the system. The manual is divided into four parts. The first part of the paper serves as a quick start into CLEARS. The second part is the mere description of the features of the system. After that we show with selected examples how to use some the system in a framework of teaching or research. The last parts consists of appendices, that give a quick overview over commands and provide other useful information.

Quick Start Section 2.1.2 describes how to obtain and install the system, the prerequisites and — most important — how to start the system, which you can see in Figure 2.1. In section 2.1.3 we give a worked example of how to use the system and explain the motivation of the system as a teaching tool.

Description of the System In 2.1.4 we give an overview over the pull-down menus of CLEARS and describe shortly their usage. Then, in 2.1.5 we do the same for the

¹This section is a preliminary version of a user manual to appear in [Maier et al., 1996].
pop-up menus that are associated with parts of the graphic output and that provide
the basic for the interactive use of the tool

**Selected Topics in using CLEARS** Here we present some of the possibilities of using
CLEAR S as a system for teaching or research.

**Appendices** In the last part of the paper we provide a short reference guide. We present
overviews concerning the most important commands and tables that summarise the
parameters.

### 2.1.2 Installation

#### 2.1.2.1 How to obtain CLEAR S

You can anonymously ftp the system

```
ftp ftp.coli.uni-sb.de
> cd /pub/Fracas
> bin
> get clears.tar.gz
```

Alternatively
you can get it, from the CLEAR S homepage [http://www.coli.uni-sb.de/~fracas/](http://www.coli.uni-sb.de/~fracas/) using your favourite web-browser.

To install the system you need
<table>
<thead>
<tr>
<th>System</th>
<th>Developer’s Web Address</th>
<th>Freely Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sicstus Prolog 3</td>
<td>Swedish Institute for Computer Science \nhttp://sicst.se</td>
<td>no</td>
</tr>
<tr>
<td>Tcl 7.4 Patch Level 0</td>
<td>J. Ousterhout \nhttp://www.sml.com/people/john.ousterhout/</td>
<td>yes</td>
</tr>
<tr>
<td>Tk 4.0 Path Level 0</td>
<td>J. Ousterhout \nsee above</td>
<td>yes</td>
</tr>
</tbody>
</table>

To our knowledge, the SICStus Tcl/Tk Interface unfortunately works only with Patch Level 0 of the latest Tcl/Tk distribution. We hope that this will be fixed very soon.

You can install the system in the directory of your choice. However, we propose to create a directory named Fracas. For the rest of this manual, we use $FRACAS as the placeholder for your installation directory and $LIBRARY as a placeholder for the directory in which your local system provides linkable libraries\(^2\).

```plaintext
> mv clears.tar.gz $FRACAS
> gunzip clears.tar.gz
> tar -xvf clears.tar
```

This will create the appropriate subdirectories. You have to set two new environment values\(^3\) to use the program. We recommend to modify your .cshrc file, so that these variables are set permanently.

```plaintext
> setenv CLEARS_DIR $FRACAS
> setenv CLIG_DIR $FRACAS/Clig
```

### 2.1.2.2 The Tcl/Tk Environment

Before you start you have to make sure, that Tcl/Tk is installed at your site and you have set or modify the following environment variables to enable SICStus Prolog to find the appropriate libraries,

```plaintext
setenv LD_LIBRARY_PATH ...:$LIBRARY/tcl7.4:$LIBRARY/tk4.0
setenv TCL_LIBRARY $LIBRARY/tcl7.4
setenv TK_LIBRARY $LIBRARY/tk4.0
```

\(^2\)If in doubt ask your superuser. Usually this directory should be /usr/local/lib.

\(^3\)Shell Commands follow the csh syntax. If you run another shell, please check the corresponding commands in your manual.
Tcl/Tk is an interpreted Script language, that eases the pain of creating user-interfaces in X-Windows. The system, developed by John Ousterhout [Ousterhout, 1994], is freely available and seems to emerge as a standard for rapid prototyping of GUI-based systems. If you don’t have Tcl/Tk, you can get it via the World Wide Web from the page http://www.sunlabs.com/research/tcl/. You will also find lots of useful information concerning these packages. You can also ftp the system from ftp.aud.alcatel.com which is the primary site for the Tcl/Tk archive.

2.1.2.3 Running CLEARS

To find your local version of SICStus Prolog, you should ask your system administrator. Make sure you are in the $FRACAS directory.

Usually, you should be able to launch SICStus Prolog with the command

> sicstus

Now you can start the CLEARS system with either consulting or compiling the top-level file make.pl, which loads the files of this distribution. This means that at the Prolog prompt you should type in either one of these commands:

    | ?- compile(make).
    | ?- consult(make).

Compilation makes the Prolog parts of this distribution (and only these) about 8 times faster than consulted code, according to the SICStus manual [SICStus, 1995].

After loading the system you should see this welcome message:

Welcome to CLEARS
===============

Please start the program with 'clears'.
More information available with 'h.'

Now you should start the system, with

    | ?- clears.

This should invoke the graphical user interface of CLEARS. Your screen then should look similar to the one in Figure 2.2.
Figure 2.2: The FraCaS CLEARS Tool
2.1.2.4 Trouble Shooting

You may come across some behaviour of the system different from the description above. In most cases this will only indicate that you didn’t set your environment variables correctly.

No Environment Variables  If the system finds no value for the variable CLEARS\_DIR, it will start the following dialog:

ERROR while loading CLEARS:
Could not read the Environment Variable CLEARS\_DIR.

You have two possibilities:

(1) Go on and let the system set this variable TEMPORARILY to your actual directory

(2) Exit and set the variable either in your shell or your .cshrc File

Please enter your selection

If you choose the first option

However it is very likely\(^4\) that in this case you also did forget to set the CLIG\_DIR variable. But then, trying to start CLEARS will result in the following error message:

\{TCL ERROR: tcl\_eval/3 - couldn\’t read file "/graphbox.tcl": No such file or directory\}\n
Please exit SICStus Prolog with halt. and either set these environment variables permanently in your .cshrc File as shown in Section 2.1.7.2 or start the shell script clears, which is described in Section 2.1.2.5.

Tcl/Tk Library not found  A much more serious error is

\{EXISTENCE ERROR: tcl\_new(\_35): procedure tcltk:(tcl\_new/1) does not exist\}\n
after you tried to start CLEARS in Prolog. In this case, you may also have noticed, that during the loading of the library tcltk.pl a fatal link error occurred, like:

\(^4\)In case you did not already install the Clig system
library -ltcl: not found ld: fatal: File processing errors.

During the startup process, SICStus Prolog could not find the libraries for Tcl/Tk. Consult your system administrator and check the default setting for the TCLLIB variable in the SICStus Makefile. You should check the exact location and name of the libraries and what setting of the TCLLIB was used during installation!

2.1.2.5 The Shellscript clears

This simple shell script for the c-shell will automatically set the appropriate environment variables, starts SICStus Prolog and compiles the system. Again you have to issue clears at the Prolog prompt to start the system.

2.1.3 Semantic Composition at your fingertips

In this section, we give two worked example of possible interactive sessions with CLEARS. They should give a first impression of how to use the system in a teaching environment. In our examples we mainly focus on how to obtain a semantic representation for an input sentence.

2.1.3.1 A first example

Our first example will show step by step how to get a semantic representation for the sentence anna laughs. To represent the meaning of each word and each phrase in the derivation, we choose the Language of Generalised Quantifiers as a representation language. This is an extensional fragment of a formalism in a Montagovian spirit.

Assuming you successfully started the CLEARS system, the first step is always to choose the semantic formalism with which you want to work with. Therefore you have to click on the Parameter Menu from which you choose the first entry Semantics. Clicking on that menu opens a cascaded menu with all possible semantic representation formalisms available in CLEARS. You should choose the entry Gen. Quantifier. These two menus are displayed in Figure 2.3. Also for the next parameter to select click on the Parameter menu. Choose the entry Subj-VP Application which also opens a cascaded menu with two entries, from which you select Apply(VP,Subj).

Now click on the Input Menu and choose from its pull-down menu the entry Sentence. An editor window will appear on your screen, waiting for your input. As our example sentence is anna laughs, you should activate the entrance field⁵ and enter our example sentence. This already has been done in Figure 2.4. With clicking on the Send To CLEARS button

---

⁵ Activation is done by clicking with the left mouse button on that field.
Figure 2.3: Choosing the semantic formalism

Figure 2.4: The Sentence Editor of CLEARS
in the rightmost corner of our edit window, we send the input to the system for evaluation. CLEARS parses the sentence and draws a parse tree in its main output window, as Figure 2.5 shows. You will notice that the tree is annotated not only with the syntactic categories, but also with semantic operations at each non-terminal. But let us first take a look at the terminals, the lexical information. Each word of our input sentence is connected to its lexical semantics. From this nuclear semantic information we compose the meaning of the whole sentence. So in this section, you will see how Frege’s Principle of Compositionality [Frege, 1879] comes to life with a click of your mouse. For this purpose the semantic operations at each non-terminal node tell us how the meaning of that node is composed from the meaning of its children. So the meaning of the np node is the result of applying \textbf{Id} — which stands for Identity — to its only child. In this case this means percolation of the lexical semantics of \texttt{anna}, which is \texttt{anna} to the np node. Move your mouse to the \textbf{Id}-Command connected with the np-node and double-click with your right mouse button on it. Your CLEARS window should now look like Figure 2.6. Please derive now the meaning of the vp node by clicking on its semantic node. The result is shown in Figure 2.7. Now we will derive the meaning of the whole sentence. The operation connected with the s node is \texttt{app(2,1)}. Clicking on this node means applying the semantic information of the second daughter to the semantic

\footnotesize
6The preterminal nodes are not annotated with semantic interpretation. Because the semantic information of a preterminal is always identical to the lexical semantics of its (terminal) child, we choose to skip semantic information at that level, otherwise this would only involve unnecessary copying. We propose to consider the preterminal together with its child as a single unit.

7Again, please exclude preterminals from your consideration.

8For convenience, we use the style \texttt{Typewriter} for syntactic information and \texttt{Bold Face} for semantic information.

9You will notice that while moving your mouse over the graphic output, the cursor changes sometimes from an arrow to a hand. The hand indicates that these areas are clickable. They are connected with commands that drive the composition process or that manipulate the output.

21
Figure 2.6: Constructing the Semantics of NP

Figure 2.7: Constructing the Semantics of VP
information of the first daughter. In general, when clicking on an app operation, CLEARS fetches the semantic information of the daughters and constructs an application according to the order specified through the arguments of app. In our case, CLEARS takes anna and λ A. laughs(A) and combines them to the application

\[ \lambda A. \text{laughs}(A) \ (\text{anna}) \]

as Figure 2.8 shows. Now the final result is only one click away. We can reduce the semantic representation at the s node using \( \beta \)-reduction. That means a term of the form \( (\lambda x. M) N \) can be reduced to \( [N/x] M \), that is substituting all free occurrences of \( x \) in \( M \) with \( N \). The result (which — surprise, surprise — you achieve through clicking on the application) is shown in Figure 2.9. Congratulations, you interactively constructed your first semantic representation. Now you may want to see all the intermediate stages to recapitulate this process. Well, click on the Graphics menu and select the checkbutton Draw Stack. Click again on Graphics and select also the checkbutton Draw Boxes. Again click on Graphics and select the last entry Change Graphics. The result in Figure 2.10 shows all steps of your derivation of the meaning of anna laughs stacked on top of each other and separated by boxes.

2.1.3.2 A more elaborate example

After this introductory example, we present how to obtain the two different readings of the sentence every man loves a woman using \( \lambda \)-DRT as our representational formalism. You will see how to use Cooper Storage as a device for obtaining the different readings.
Figure 2.9: Final representation of *Anna laughs*

Figure 2.10: Seeing the whole construction process
To describe the parameter settings, we will print a Path showing the names of the menu entries which you have to click on. So Parameters/ Semantics/ Lambda-DRT means that you have to click on the top-level menu Parameters, choose the entry Semantics and click on the radiobutton for Lambda-DRT in the cascaded menu that appears.

Our parameter settings in this new format are

Parameters/Semantics/Lambda-DRT
Parameters/Subj VP Application/Apply(Subj,VP)
Parameters/Quantifier Store/Cooper Storage

Note that we did not change all possible parameters, but the necessary ones to obtain the correct result. You could also try to use Grammar 1 and Lexicon 1. However we recommend to delay the experiments with the parameters until you know more about them. An inconsistent parameter setting may cause subtle problems that are not easy to spot.

After changing the parameter, please enter the new sentence every man loves a woman into the edit window, which you, as you already know, get under Input/Sentence. After having sent the input to CLEARS, you should see a screen similar to Figure 2.11. Now double-click with the right mouse button on the Id operations below the two nbar nodes. After that move your mouse to the app(1,2) node of the first np and double-click on that node. This will result in constructing an application following the procedure outlined in the previous example. The next double-click will save the semantic representation of the np node in the storage, later to be discharged at the s node. The store consists of a sequence of variables, each of it is connected with a formula that can be discharged. Graphically the store is separated from the remaining formula by a surrounding box, it, as you see in Figure 2.12. Now you should double-click with the right mouse button 2 times at the app(1,2) operation associated with the second np node. After that, repeat two double-clicks at the semantic operations below the vp and the s node. At the s node you should now see the store, consisting of two entries, that means two variables and their associated formulas. Next to the store, you see the stripped semantic representation of our input sentence. The variables in the argument of this representation correspond with the dischargeable variables in the store.

To obtain a reading you have to discharge the variables in the store. Move your mouse to the store, to the variable R associated with the denotation of a woman. Click on the variable without releasing the button and you will see a pop-up menu, like that in Figure 2.13 appearing on the screen, from which you should choose the Discharge entry. This will remove this variable from the store, substituting its occurrence in the stripped semantics with the formula it is connected to. Now discharge the other variable and you get a representation without the store, but with the second DRS not fully merged. Double-click again on this

---

10 The conjunction is absolutely necessary. If you choose to change the grammar to Grammar 1, you have to change your lexicon to Lexicon 1, too.
11 This connection is displayed by the use of the :: operator between variable and formula
12 And please do it in that order.
13 The variables in the store with additional boxes around them.
Figure 2.11: Initial Representation of every man loves a woman
Figure 2.12: Representation of a Quantifier Store

Figure 2.13: The PopUp menu associated with dischargeable variables
semantic node to obtain the final representation. This final semantic representation of our input without drawing the whole tree, is shown in Figure 2.14. If you choose a different order of discharging your variables from the store, you get the other reading of that sentence, which you can see in Figure 2.15. This completes our guided tour through CLEARs. We saw how to choose parameters from pull-down menus and you saw how to manipulate a structure through a pop-up menu. Pop-up menus that contain commands providing the interactive behaviour of the system are connected to every class of structures of our parse tree. Usually on the left button you have commands that are useful in further analysis of the structure. On the middle button you find menus containing commands that allow to edit or manipulate the graphical output. The right button is usually associated with a single command, that should be the most frequently used command for this class of structures.

Especially, on the popup menu of semantic nodes invoked with the left button of the mouse, you find different commands for semantic construction. During the construction process you may choose between those commands to construct the next representation. In Figure 2.16 we show schematically how to obtain a semantic representation of an input sentence.

2.1.4 Pull-Down Menus

The CLEARs system automatically installs itself on top of the interactive grapher CLIg. A detailed description of the CLIg system is given in D15. Here we will just explain the specific menu bars and entries of CLEARs. These are the Input, Parameter, Graphics and Tools Menu, that you already have seen in Figure 2.1.
Figure 2.16: How to interactively obtain a semantic representation
2.1.4.1 The Input Menu

Here you choose your basic input, either a sentence or an individual semantic or syntactic representation that you want to have displayed. The menu contains the following entries:

Sentence lets you enter a single sentence to be analysed.

Discourse enables you to enter a discourse sentence by sentence. In the present version of CLEARS, this only works for DRT.

Semantic Representation allows you to enter a single semantic expression in our Prolog-representation

Syntactic Representation allows you to enter a syntax tree.

Show Lexicon displays the actual lexicon according to the parameter setting. This is especially useful, if you don’t know the coverage of the lexicon.

Show History displays all your input connected with the parameter setting and lets you choose one of your former results.

2.1.4.2 The Parameter Menu

In the parameter menu you will find all linguistically relevant parameters. Each of these is connected to a cascaded menu from which you have to choose one particular setting. Note that not all possible combinations of parameter settings are also feasible. In Section 2.1.4.3 we present incompatible settings.

Semantics lets you choose the semantic representation formalisms that are currently available in CLEARS.

- DRT Implementation of the first two chapters of [Kamp and Reyle, 1993]
- Lambda-DRT λ-DRT see [Bos et al., 1994; Kuschert, 1995]
- Compositional DRT A variant of a compositional DRT by [Muskens, 1994]
- Gen. Quantifier Language of Generalised Quantifiers
- Intensional Logic intensional Montagovian-style fragment based on [Montague, 1973]
- Lambda Language extensional Montagovian-style fragment based on [Dowty et al., 1981]

Situation Semantics based on the fragment described in D15/D16

Grammar lets you choose a grammar.

Grammar 1 is a very simple DCG grammar
Grammar 2 is a DCG style grammar with features

DRT Grammar is a GPSG-style Grammar according to chapter 0 of [Kamp and Reyle, 1993].

Lexicon lets you select a lexicon for your grammar. Lexicon and Grammar have to agree, so you should always take care to change Lexicon and Grammar together.

Lexicon 1 is suitable for Grammar 1
Lexicon 2 is suitable for Grammar 2
DRT Lexicon Coverage of the first two chapters of [Kamp and Reyle, 1993]. Only suitable for DRT-Grammar.

Parser lets you select a Parser for your Grammar. In the present system the choice of the grammar determines the parser.

DCG for Grammar 1 and 2
Chart Parser for DRT fragment

Typing lets you choose between typed and untyped $\lambda$-calculus.

Untyped for untyped $\lambda$-Calculus
Typed for typed $\lambda$-Calculus

SynSem Mapping describes the mapping between the grammar rules and how they are linked to semantic operations. This only shows effect when used in serial or radical-serial mode.

Rule2Rule enables Rule to Rule syntax semantics mapping. Grammar rules are linked to semantic operations.

Templates enables template based syntax to semantics mapping.

SynSem Mode describes the way in which the syntax tree is annotated with information used to extract semantic information.

Serial annotates the syntax tree with templates or syntactic rules that are described in 2.1.4.2. The semantic operations connected with these are then fetched in a subsequent step.

Parallel annotates the syntax tree immediately with semantic operations.

Radical Serial doesn’t annotate the tree. Used together with typed $\lambda$-calculus for type-driven semantic construction.

SUBJ-VP Application enables the application order of a VP and its subject at an S node. This allows to treat the meaning of an NP either as an entity or or as a higher typed object.

Apply(Subj, VP) applies the VP to the complement daughter at an S node.
Apply(VP, Subj) applies the complement daughter to the VP at an S node.

Logical Variables lets you choose the kind of variables for the λ-reducer to work with.

Logical Variables enables logical variables implemented as Prolog variables.

Constants enables logical variables implemented as constant terms, realised via the Prolog predicate `numbervars/3`.

Lambda Reducer lets you choose the computing strategy of our λ-reducers.

Unification Based enables unification based beta reduction.

Modified Unification enables β reduction using a copying stage ([Cooper et al., 1993])

Substitution Based enables full rewrite based lambda reduction including alpha conversion

Quantifier Store lets you choose the strategy of quantifier storing. The names of the parameters should be self explanatory.

Cooper Storage see [Cooper, 1983]

Nested Cooper Storage see [Keller, 1988]

None

Get Defaults selects a default parameter setting for a semantic representation. The user can modify the default settings in the file `$FRACAS/user_defaults.pl`, but this should only be done by the experienced user.

Check Setting checks your actual parameter setting.

2.1.4.3 Dependencies between Parameters

Not all possible combinations of our parameters lead to any output at all. These incompatibilities fall into two classes, logical incompatibility or implementational incompatibility. The latter class describes possibilities that are not yet implemented, as opposed to the former class, in which there are logical reasons for their incompatibility. Whereas parameter combinations that fall into the latter class may be allowed in future releases, the ones that fall into the former class will (very likely) never lead to any workable configuration of the system. The implementational problem at this stage concerns the Situation Semantics only working with Logical Variables.

For the class of logical incompatibilities we present tabulars with positive occurrences of parameter combinations. The negative cases immediately follow from the fact, that these positive cases describe exhaustively all possible combinations of one pairing. But this also implies that all other settings are freely combinable.
<table>
<thead>
<tr>
<th>Semantics</th>
<th>SUBJ-VP Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lambda-DRT, Compositional DRT, Intensional Logic</td>
<td>apply(subj_vp)</td>
</tr>
<tr>
<td>Lambda Language, Situation Semantics</td>
<td></td>
</tr>
<tr>
<td>Gen. Quantifier</td>
<td>apply(vp subj)</td>
</tr>
<tr>
<td>Semantics</td>
<td>SynSem Mapping</td>
</tr>
<tr>
<td>Situation Semantics</td>
<td>Rule2Rule</td>
</tr>
<tr>
<td><strong>Grammar</strong></td>
<td>Lexicon</td>
</tr>
<tr>
<td>Grammar 1</td>
<td>Lexicon 1</td>
</tr>
<tr>
<td>Grammar 2</td>
<td>Lexicon 2</td>
</tr>
<tr>
<td>DRT Grammar</td>
<td>DRT Lexicon</td>
</tr>
<tr>
<td><strong>Lambda Reducer</strong></td>
<td>Logical Variables</td>
</tr>
<tr>
<td>Unification Based</td>
<td></td>
</tr>
<tr>
<td>Modified Unification</td>
<td>Logical Variables</td>
</tr>
</tbody>
</table>

One negative instance has to be mentioned: The *untyped* $\lambda$-calculus only works with the SynSem Mode \textit{rad.ser}.

### 2.1.4.4 The Graphics Menu

The Graphics menu involves parameters that are concerned with the graphical output or with manipulating the construction process. Almost all entries are checkbuttons that allow to choose between two possibilities. The last entry allows to supply the changed parameters to the graphics on the fly.

**Draw Stack** If set, draws all available semantic information of one node stacked on top of each other.

**Draw Boxes** If set, draws boxes around semantic information.

**Storage Sequence** If set, draws the Quantifier store as a stack. If not set draws it as a sequence.

**Full Representation** If set, calculates the semantic representation of the whole tree in one step.

**Immediate Info** If set, a mother takes only the available semantic information of the daughters to combine the meaning. If not it calculates the semantic representation for each daughter before calculating her own representation.

**Skip Sem. Operation** Only useful in Serial Mode. If set, allows to go from a Template/Rule annotation of a node immediately to the semantic representation without showing the semantic operation.

**Change Graphics** Change Graphic parameters for the output on the fly.
2.1.4.5 The Tools Menu

Here you find useful tools to compare different semantic formalisms and different output.

**Translate** Allows to translate between different semantic formalisms. This is not implemented for all possibilities.

**Add Translation** Temporarily adds a translation in stacked and boxed form to the actual semantic output.

**Copy Structure** Starts a new window with a copy of the actual output.

**New CLEARS** Starts a fresh CLEARS process

2.1.4.6 Add-ons in the Help menu

**About CLEARS** Information about CLEARS.

**Authors** Information about the authors.

2.1.5 Pop-Up Menus

With several structures of the output representation one can associate Pop-Up Menus which get activated while clicking on that structure. One recognises a clickable structure on the change of the mouse cursor from an arrow to a hand, while moving over these regions. Pop-up menus are invoked by a single mouse-click, operations that lie usually on the right mouse button are started with a double-click. On the left mouse button one usually finds operations connected with further linguistic processing such as Application or Discharge. The middle button is used for manipulating the structure and the tree graphically, such as zooming or editing. The pop-up menus may also vary with the actual semantic formalism or the actual grammar, so we describe a superset of all possible menu entries, although there may be no such menu that involves all these commands.

The clickable structures include the semantic nodes, the syntactic nodes, the lexical nodes (syntax and semantic information) and the quantifier store.

2.1.5.1 Pop-Up Menu for Semantic Nodes

**Linguistic Processing** holds the commands to drive the composition process, allowing to choose an individual operation at each point in the derivation.

**Construct Meaning** allows different steps of the composition process to be done at once. This command also lies on the right mouse button for convenience.
Merge merges together two DRSs.
Reduce just reduces an Application.
DownUp Cancel invokes the DownUp Cancellation.
Meaning Postulate allows to select and apply a meaning postulate.
Store will store a representation in the store.

Editing offers commands to edit representations or to manipulate the look of the graphical output.

Edit allows to edit a representation
Undo allows to undo your last operation.
Zoom in displays only the selected semantic node on the screen.
Zoom out goes back to the display the whole tree.

2.1.5.2 Pop-Up Menu for Syntax Nodes

Again we have one menu dealing with tools for further inspection of the syntax node on the left mouse button, whereas the Editing menu lies on the middle button. The right button is not used in this version.

Inspection holds some commands for examining the syntactic features, provided that the grammar exploits such features.

Expand FS allows that the feature structure of that node is shown instead of the plain category
Expand Selection only selected features are displayed.
Select Feature allows to select a subset of all features. This may be used to display only those features relevant to the semantic phenomena under examination.
Show Category displays the category of a node instead of the feature structure.

Editing contains the same entries as the one for semantic nodes. Therefore we wont give an explanation here, but refer to 2.1.5.1.

2.1.5.3 Pop-Up Menu for Lexical Nodes

These nodes only have a subset of Operations that one finds at the corresponding non-terminal nodes.
2.1.5.4 Pop-Up Menu for Quantifier Store

For these objects, there is only one menus lying on the left mouse button.

**Discharge** takes the variable out of the store and substitutes its occurrence in the main-formula with its associated, stored formula.

**Edit** allows to edit the content of the store.

2.1.6 Selected Topics

Here we show how use the possibilities that **CLeARS** offers for certain representative tasks in a teaching or researching environment.

2.1.6.1 Dealing with Disambiguity

We go back to our example in Section 2.1.3.2 and show how you can obtain both readings of that sentence and displaying them at the same time on your screen. Assume you have constructed a semantic representation of the sentence **every man loves a woman**, up to the `s` node, so that your present output looks like in Figure 2.17. With the command **Copy Structure** from the **Tools** menu, we can create a copy of our actual output and display this copy in a new **CLeARS** window. Figure 2.18 shows both **CLeARS** processes with a copy of the same graphical output, So we have the possibility to choose for each window a different order to discharge the variables in the store. This allows us to display both readings at the same time. The final output is shown in 2.19. To display only the relevant semantic nodes in both processes, we choose the **Zoom in** command from the editing\(^{14}\) pop-up menu for semantic nodes.

2.1.6.2 Translation

Now we can also translate from one representation formalism to the other. In the current version, we have only implemented translation routines from the **DR Theories** to the ones based on predicate calculus. We follow the algorithm outlined in [Kamp and Reyle, 1993]. For our final result of the last section, shown in Figure 2.19 we show a translation to our **Lambda Language** in Figure 2.20.

\(^{14}\)The one you get clicking on the middle button, if the mouse is over a semantic node.
Figure 2.17: Fully expanded representation for every man loves a woman.
Figure 2.18: Two CLEARS processes displaying the same representation
Figure 2.19: The two final representations

Figure 2.20: Representations and Translations
2.1.7 Appendices

2.1.7.1 Starting Clig

Here we present the different commands at the SICStus Prolog top-level. Always make sure that you have set the appropriate Environment variables. Check 2.1.7.2 for this.

**clears**
starts the CLEARS system with default values for λ-DRT. You might change this default setting in the file $FRACAS/Clears/default_values.pl

**clears(+Sems)**
starts the CLEARS system with default values for the semantic representation Sems. An overview of possible values is given in 2.1.7.3. Choose a value under the *sems* value in that table.

**h**
shows a help screen, that shows you the possible commands.

2.1.7.2 Environment Variables

We assume that $FRACAS is the name of the directory in which you have installed CLEARS. $LIBRARY is the directory in which you find the tcl/tk libraries on your local machine.

```plaintext
setenv CLIG_DIRS $FRACAS/Clig
setenv CLEARS_DIRS $FRACAS
setenv TCLLIBRARY $LIBRARY/tcl7.4
setenv TCL_LIBRARY $LIBRARY/tk4.0
setenv LD_LIBRARY_PATH ...$LIBRARY/tcl7.4:$LIBRARY/tk4.0
```

2.1.7.3 Library Parameters

Here we present a tabular showing which library parameters are set when selecting from the CLEARS menu Parameters.
<table>
<thead>
<tr>
<th>Semantic Formalism</th>
<th>Parameter Menu</th>
<th>Library Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter Menu</td>
<td>Library Parameter</td>
<td></td>
</tr>
<tr>
<td>DRT</td>
<td>sems</td>
<td></td>
</tr>
<tr>
<td>Lambda-DRT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compositional-DRT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gen. Quantifier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intensional Logic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lambda Language</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Different Grammars</td>
<td>Grammar</td>
<td>grammar</td>
</tr>
<tr>
<td>Grammar 1</td>
<td>psg1</td>
<td></td>
</tr>
<tr>
<td>Grammar 2</td>
<td>psg2</td>
<td></td>
</tr>
<tr>
<td>DRT Grammar</td>
<td>drt,gram</td>
<td></td>
</tr>
<tr>
<td>Different Lexica</td>
<td>Lexicon</td>
<td>lexicon</td>
</tr>
<tr>
<td>Lexicon 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lexicon 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRT Lexicon</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Different Parsers</td>
<td>Parser</td>
<td>parser</td>
</tr>
<tr>
<td>DCG</td>
<td>dcg</td>
<td></td>
</tr>
<tr>
<td>Chart</td>
<td>chart</td>
<td></td>
</tr>
<tr>
<td>Typing</td>
<td>typing</td>
<td></td>
</tr>
<tr>
<td>Typed</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>Untyped</td>
<td>ut</td>
<td></td>
</tr>
<tr>
<td>Syntax-Semantic Mapping</td>
<td>SynSem Mapping</td>
<td>synsem</td>
</tr>
<tr>
<td>Rule2Rule</td>
<td>rr</td>
<td></td>
</tr>
<tr>
<td>Templates</td>
<td>tm</td>
<td></td>
</tr>
<tr>
<td>Annotation of Syntax Trees</td>
<td>SynSem Mode</td>
<td>serpar</td>
</tr>
<tr>
<td>Serial</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radical Serial</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dealing with type-raised NPs</td>
<td>Subj-VP Application</td>
<td>subjvp</td>
</tr>
<tr>
<td>Apply(Subj, VP)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apply(VP, Subj)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>different style of variables</td>
<td>Logical Variables</td>
<td>logvars</td>
</tr>
<tr>
<td>Logical Variables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constants</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computation of $\beta$-reduction</td>
<td>Lambda Reducer</td>
<td>lambdared</td>
</tr>
<tr>
<td>Unification Based</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modified Unification</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Substitution Based</td>
<td></td>
<td></td>
</tr>
<tr>
<td>selects the quantifier storage</td>
<td>Quantifier Scope</td>
<td>qscope</td>
</tr>
<tr>
<td>Cooper Storage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nested Cooper Storage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.2 Extending CLiG with Interaction and User Defined Graphics

D15 describes the CLiG grapher and its use as a general tool for visualizing common linguistic graphical notations. The main purpose of the program is its use as a graphical interface for applications in computational linguistics. These applications can display their data by creating description strings and sending them to the grapher. These strings are hierarchical textual descriptions of the graphical structures. E.g., the following text is the description for the graphical structure seen in Figure 2.21:

{drs {plain-text B}
  {tree {plain-text s}
   {plain-text B}
   {tree {plain-text vp1}
    {tree {plain-text vp}
     {tree {plain-text v}
      {plain-text owns}}
     {tree {plain-text np}
      {tree {plain-text pn}
       {plain-text ulysses}}}}}}}

With its description string language, CLiG provides a wide range of possibilities to create the graphical representations one has in mind. Objects can e.g. be ordered in stacks or sequences, can be put in boxes or can each be assigned colors separately. The basic notations like trees, texts or DRS boxes can be nested freely within each other, producing complex graphical structures like e.g. Figure 2.22. However, for certain applications, the grapher’s built-in facilities might not be sufficient to produce the desired output. However, CLiG has been designed in a way that makes it relatively easy to add new graphical features. Section 2.2.2 describes how a user can add his or her own graphical objects to the grapher.

By linking objects in the description string to pieces of code, an application defines interactive graphics where a user can perform application-dependent actions by clicking on objects with the mouse cursor. Interactive graphics are under the control of the application which creates the description strings. The application decides which objects behave in what way by sending the appropriate code with their descriptions. Interaction between grapher and application is freely programmable and therefore very flexible, but may also require some effort on the side of the programmer of the application. CLiG is extendible in the sense that the application programmer can add the amount of interactive behavior needed for the specific application. Section 2.2.1 explains how interfaces between applications and the grapher can be build.

---

15 This section is a preliminary version of a user manual to appear in [Konrad, 1996].
2.2.0.4 Tcl/Tk as the programming environment

CLIG has been implemented using the software package TCL/Tk, a programming system for graphical user interfaces. TCL, the tool command language, is a simple interpreted scripting language providing basic programming facilities like variables, procedure definitions and loops. Tk is a toolkit for the X Windows System that extends the core TCL-language with commands to build Motif-like graphical user interfaces. The TCL/Tk package can be used via the interpreter program WISH that contains the language interpreter for TCL and the Tk toolkit. CLIG’s source code is a set of TCL scripts which gets interpreted by WISH.

The use of TCL/Tk has some obvious advantages and one minor drawback. One advantage is portability: TCL/Tk is free and widely available for different OS-platforms. The X-Windows functionality used by Tk gets emulated for foreign systems like Windows or Mac OS. Another advantage is the rapid prototyping of code. Since TCL gets interpreted, code extensions or modifications for the graher can be loaded on-the-fly without the need for recompilations. Modified code simply overwrites old, while new code can be added following CLIG’s simple code conventions (see section 2.2.2.2). In contrast to C or C++, TCL/Tk hides low-level details of the X-Windows environment from its users, making it generally easier to write TCL/Tk graphical user interfaces than C/C++ user interfaces. The minor disadvantage of
Tcl/Tk is the slower execution speed of the interpreted Tcl code which can be circumvented by writing C routines for time-critical algorithms. Thanks to the speed of modern work stations, this kind of optimization is rarely necessary. CLIG currently does not contain any non-Tcl code.

2.2.1 Interaction

An interactive graphical structure contains parts which are mouse sensitive; in CLIG these parts are linked to small Tcl programs called scripts. A Tcl script in an interactive graphical structure could e.g. open a pop-up menu or ask an application for more information on whatever a user has clicked on. The linking of code to graphical objects in the description strings can be done by the special commands clickable and active.

2.2.1.1 The clickable command

The clickable command uses one single script which gets executed when the user double-clicks on the associated object. The command takes two arguments, the object to be drawn and the Tcl script linked to it. The following example creates a mouse sensitive text box which produces a short message when clicked by the user:

{clickable {bigbox plain-text "click me!"}
   {puts "YOU CLICKED ME!"}}

2.2.1.2 The active command

The active command can define multiple possibilities for the same object or define special behaviors. It allows the use of several event-script pairs for the same object. An event can be any X-Window event known in Tk such as <Leave>, <1>, <Double-3> etc. The general syntax for active is

{active <object> {{event} <script>} ... {{event} <script>}}

If you wanted the message from the last example to appear when the middle mouse button gets clicked once, you would express it like this:

{active {bigbox plain-text "click me!"}
   {<2> {puts "YOU CLICKED ME!"}}}
2.2.1.3 Application meets Clig

The concept of interfaces between applications and Clig is shown in Figure 2.23. The application “talks” to the grapher by sending it description strings. Some systems like Sicstus3 have their own Tcl/Tk interface which can be used for this. Another way to create an interface to the grapher is the use of the send mechanism which can be used by any Tk application. We are also currently working on a very simple interface which uses Clig’s standard input and output. In all these cases, the main drawing routine clig gets called by the application with an description string containing clickable or active commands. The commands use Tcl scripts which in response to a mouse action force the application to do something. The following example is taken from the larger description string which produces the graphical structure in Figure 2.22:

```
{bigbox clickable
 {plain-text "app(1,2)"
 {prolog_event "click(sem_node, d1, 3)"}}}
```

The example structure is created by the educational tool developed in the FraCaS project. The tool uses the Tcl/Tk interface provided by Sicstus3 Prolog. In the example, the text app(1,2) is linked to a script specifying that a prolog_event command gets executed when the text gets double-clicked with the mouse. The prolog_event command results in an event which gets noticed by a loop in Prolog waiting for such an event. The string click(sem_node, d1, 3) informs Prolog that the node 3 has been clicked on. The main loop then can take the appropriate action and will display a new graph where node 1 has been applied to node 2.

The concrete realization of the interface between an application and Clig depends on what the host system offers as a means of communicating with Tcl/Tk. We are currently experimenting with different general approaches like using a TCP/IP channel or simple standard input/output piping to allow a common interface.

2.2.2 Adding graphical objects

This section describes how user defined objects can be included into the grapher. Subsection 2.2.2.1 explains how code extensions appear in the description strings which are used for all representation and interfacing purposes within the grapher. Subsection 2.2.2.2 explains the conventions necessary for code extensions.

2.2.2.1 Description strings

The description of a graphical object in Clig is done with a hierarchical list in Tcl-syntax, the description string. The general syntax of a description string is
\begin{verbatim}
{<command> <parameter_1> ... <parameter_n>}
\end{verbatim}

Most commands take description strings as their parameters. If a command takes only one description as parameter, the command is said to modify the description. The general syntax for such a modifying command is

\begin{verbatim}
{<modify-command> <command> <parameter_1> ... <parameter_n>}
\end{verbatim}

E.g. a description string like \{neg plain-text "x"\} is a Tcl-list with 3 members, with the first two of them being commands. The \texttt{neg} command modifies the output of the \texttt{plain-text} command by putting a negation in front of the text.

Whenever \textsc{Clig} draws a graph, it really interprets a description string like this as a piece of Tcl-code which lacks some information, e.g. the exact positioning of the objects. The description string gets executed by an internal interpreter which recursively fills in the missing information after it has calculated the correct positions of all objects in the drawing area. Each command in the description string has an equivalent Tcl procedure which gets called with the missing information and all parameters. Extending \textsc{Clig} means writing such procedures and adding them to the source code.

As an example, take a look at the code which is responsible for drawing negations:

\begin{verbatim}
proc neg {x y where mtags obj} { ;# draw negated drs
  seq $x $y $where $mtags \n  [list \{sign neg\} \{plain-text "\} \$obj]}
\end{verbatim}

Without going into details, the code for \texttt{neg} simply calls another command used in \textsc{Clig}, \texttt{seq}, with three graphical objects, the negation sign, a space character and the object which should be negated. The \texttt{seq} command gets some additional parameters it inherits from the \texttt{neg-call}, namely \texttt{x} and \texttt{y} for the exact position of the object, \texttt{where} for the canvas which should be used (\textsc{Clig} uses two canvases, one for actual drawing and one for calculating the size of an object) and \texttt{mtags} which is used for active (clickable) regions. The \texttt{mtags} parameter contains a set of labels and usually is only manipulated by the \texttt{active} and \texttt{clickable} procedures for grouping together objects with the same scripts linked to them. All other commands do not change \texttt{mtags} but must inherit this parameter to each of its daughters. By inheriting labels to their daughters, clickable objects correctly execute their associated script when one of their daughter objects are clicked on.

\subsection*{2.2.2.2 Code conventions}

Each graphical command for \textsc{Clig} must have exactly 5 parameters: the \texttt{x} and \texttt{y} positions of the top left corner, the \texttt{where} canvas, the \texttt{mtags} parameter and an additional parameter \texttt{obj}
containing everything that was in the object description string except the command name itself.

Each command must return the width and height of the drawn object as a list. A good example of the use of the sizes would be the code for underline which simply underlines another object:

```tcl
proc underline {x y where mtag obj} {
    ;# drawing
    set size [execobj $obj $x $y $where $mtag]
    ;#
    set yline [expr $y+[lindex $size 1]+2]
    $where create line $x $yline \[expr $x+[lindex $size 0]] $yline
    ;# returning the complete size of the underlined
    ;# object:
    list [lindex $size 0] [expr $yline-$y]}
```

The underline procedure first draws the object obj by calling execobj, the main drawing function which tries to execute the description string obj at position x,y. execobj, the interpreter procedure, returns the width and height of the object which underline uses to calculate where (y position + 2 points) and with which width to draw the line under the object. underline returns the width of the object and the height + 2 points as the size of the object drawn.

This example covers almost everything one has to know about writing code for extending the grapher. A procedure has to have a unique name, must use the 5 formal parameters described above and must return the correct width and height of the object it draws—and then it is a legal extension of the grapher. An application can simply load the additional code using the methods described in section 2.2.2.4 and then use the new procedures in the same way as the “factory” commands Clig provides. A user can even replace the built-in commands by simply overwriting them with his own code since Tcl will not report an error for redefining procedures. In this way, an application might modify the graphical output for its special needs, e.g. by either optimizing for beautiful layout or speed, without the need for actually changing the original code.

The simple convention used here has a few practical drawbacks, e.g., the layout algorithm of an object can only use rectangular regions for calculating the positions of its daughters. In the case of objects with very irregular shape, this may result in a waste of space and/or may result in an aesthetically unsatisfying result. Another drawback ist the strict top-down approach of drawing the graphics. Some graphical structures must calculate the size of their daughters before they can actually draw them; in this case a bottom-up approach would be better. However, the nasty side effect of having to draw objects unnecessarily because of repeated size calculations with Clig’s top-down algorithm can be circumvented by using the calcsize command described in the next section. 

47
2.2.2.3 Canvases and calculating sizes

Sometimes a command has to know the size of an object before it gets drawn. E.g., the stack command stacks several objects on top of each other; the command uses the size of the biggest box as the maximum x size of the whole object and centers all objects within this maximum size. Therefore, before any object gets drawn, stack calculates all sizes and stores the maximum in the variable xsize. A simplified version of this calculation routine would look like this:

```c
foreach item $stack {
    set size [calcsize $item]
    set xsize [max $xsize [lindex $size 0]]
}
```

The procedure which calculates the size of all objects, calcsize, actually draws the object on an invisible canvas and returns the size which every Clig drawing procedure returns.

The calcsize procedure is almost identical to execobj except that it does not need any coordinates nor mtags which only play a role in the visible drawing area. While execobj uses the visible .graphics canvas, calcsize simply uses the canvas .invisible. This special canvas never gets displayed by the grapher; it only exists for evaluating sizes. Procedures can take advantage of this distinction by minimizing what actually gets drawn during size evaluation. Since drawing in an invisible area does not have to look good or even complete, procedures may leave out any actual drawing whenever they encounter .invisible as their where parameter, as long as they can calculate the correct size of the object they draw.

As an additional optimization, calcsize uses a memoization technique for storing sizes it already has calculated.

2.2.2.4 Including your own code

To include your own code into Clig, you can simply change the file extend.tcl in the Clig-directory so that it loads your files when you start the grapher. The only problem with this is that the extensions get loaded whether they are needed or not and that you have to modify extend.tcl whenever you install the grapher. An application using Clig usually can do better by simply performing the source command of Tcl. In this way, an extension gets loaded only when needed by the specific application.

2.2.2.5 An example: Circles

Supposed that you wanted to write a package for Clig for displaying e.g. finite state machines and therefore needed circles to represent the states of the machine. In this case you would have to write your own code, since circles are not supported in the standard release of Clig.
The first step would be to create a new file `circle.tcl` and to add this file to the `extend.tcl` loading list as described in section 2.2.2.4. The file `circles.tcl` will be loaded whenever you run CLIG. Every time you change your code, you can include your changes simply by exiting your old CLIG and restarting it again. An even easier way would be to use the **Load...** option from the CLIG main menu. Although this option usually is used to load object descriptions, it also is able to load arbitrary Tcl source code.

Now you can add the circle code. You want to use the commands `state` for a normal state and `final` for a final state in your finite state package. A state simply consists of a circle of a certain size, while a final state will be displayed as two concentric circles. Additionally, you want to have states to have arbitrary sizes, so that `{final 20}` will display a finite state with a radius of 20 points.

The code for a normal state could look like this:

```tcl
procedure state {x y where mtags rad} {
    set rad_size [expr $rad*2]
    $where create oval $x $y \ 
    [expr $x+$rad_size] \ 
    [expr $y+$rad_size] \ 
    -outline black -tags $mtags
    list $rad_size $rad_size
}
```

Since you wanted the state to have a radius of `rad`, the size of the complete object will be 2 `rad` in x and 2 `rad` in y direction. The procedure draws an oval with a black outline which bears the labels `mtags`. The labels are necessary for interactive graphs, see section 2.2.1. The code does draw an oval whenever the procedure gets called, but this may not always be necessary. Trees and other complex structures often check for the size of their daughters just to see where they can be placed. The following refinement of the code leaves out this unnecessary drawing of the circle:

```tcl
proc state {x y where mtags rad} {
    set rad_size [expr $rad*2]
    if {%where!=".invisible"} {
        $where create oval $x $y \ 
        [expr $x+$rad_size] \ 
        [expr $y+$rad_size] \ 
        -outline black -tags $mtags
        list $rad_size $rad_size
    }
}
```

This version only draws the circle when the destination canvas is not equal to `.invisible`. The code for final states is very similar to the code for states; it just puts a smaller circle inside the outer circle.
proc final \{x y where mtags rad\} \{
    set rad\_size [expr $rad*2]
    set offset 3
    if \{$where!=".invisible"\} \{
        $where create oval \$x \$y \{
            [expr $x+$rad\_size] \"
            [expr $y+$rad\_size] \"
            -outline black -tags $mtags
        \}
        $where create oval [expr $x+$offset] \{
            [expr $y+$offset] \{
            [expr $x+$rad\_size-$offset] \{
            [expr $y+$rad\_size-$offset] \{
            -outline black -tags $mtags}
        \}
        list $rad\_size $rad\_size\}
    \}
\}

After rerunning Clig, you should have the possibility to display states and final states as displayed. For a quick test, just type in the following test file in a text editor and load it with Clig:

    # example: two states
    clig \{Seq \{state 20\\}
        \{final 20\}\}

2.2.2.6 Colors

The color command provides an easy way to add color information to graphs. User defined object code must include the following to be compatible with the color command:

- the command globals main\_color to make the global variable main\_color known to the procedure
- each drawing command (like e.g. drawing a line) must use main\_color for the main drawing color. Most drawing commands, e.g. create line use -fill as the option for specifying the foreground color.

The code for underline contains only one drawing command, the create line code which draws the underlining line. With -fill $main\_color, the line’s color can be changed by a color command:

    proc underline \{x y where mtag obj\} \{ ;# line under object
        global main\_color

50
set size [execobj $obj $x $y $where $mtag]
set yline [expr $y+[lindex $size 1]+1]
$where create line $x $yline \
    [expr $x+[lindex $size 0]] $yline \
    -tags $mtag -fill $main_color
list [lindex $size 0] [expr $yline-$y]{
Figure 2.22: A complex nested graphical structure
Figure 2.23: Application-to-Clig interaction.
2.3 Ekntool User Manual

2.3.1 Introduction

EKNTOOL is a tool supporting the development of grammars producing semantic output in EKN notation. The tool includes programs to perform operations on EKN-style representations and to convert these representations in Latex format and a perl interface that invokes Prolog to parse a sentence producing an output in EKN notation, puts the output in Latex format, and displays the result using xdvi. The tool comes with several demonstration grammars, including the 'fracas main' grammar.

EKNTOOL derives from the stdrt tool developed by Robin Cooper, Julian Day and Phil Kime for dyana-2. Unlike stdrt, ekntool allows different grammars to be used, and new grammars are included. In particular, a new default grammar is specified - the situation-theoretic grammar developed by Robin Cooper and Massimo Poesio as part of FRACAS project. Many changes have also been made to the original code to make the system more usable by the general public and more portable to other sites.

The tool has been tested under SunOS 4, using sicstus version 2.1, latex 2e, and perl version 5.

2.3.2 Interacting with ekntool

2.3.2.1 Starting Ekntool

You can start the perl interface by calling the perl script ekntool contained in the directory which contains the code. After establishing a connection with prolog, the program prompts the user for some input, which can be either a sentence or a command (see below).

You can also use the system directly from Prolog. To do so, start sicstus, load the file main.pl contained in the ekntool home directory, and then type

?- loop.

This will start a loop during which of reading sentences and displaying the output. The output is displayed in the same form used by the perl script.

2.3.2.2 The Perl Interpreter

After typing ekntool, the user is presented with a prompt. The perl interpreter can be used both to parse sentences and to execute commands (to set switches, get help, and so
forth).

In order to have the system process a sentence, simply type the sentence at the prompt. Single sentences can terminate either with punctuation marks (currently, only period, '·', is recognized) or with carriage return. The sentences in a multiple sentence texts should be separated by punctuation marks.

Typing 'help' will show you what commands are available. Currently (January 1996) these include:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'help'</td>
<td>Displays this summary</td>
</tr>
<tr>
<td>'test'</td>
<td>Tests entire current database. See help on 'test'</td>
</tr>
<tr>
<td>'help &lt;subject&gt;'</td>
<td>Displays help on 'subject'</td>
</tr>
<tr>
<td>'load &lt;file&gt;'</td>
<td>Loads database of discourses 'file'</td>
</tr>
<tr>
<td>'loadpln &lt;file&gt;'</td>
<td>Loads PLN database of PLN objects 'file'</td>
</tr>
<tr>
<td>'view'</td>
<td>View currently loaded database</td>
</tr>
<tr>
<td>'viewpln'</td>
<td>View currently loaded PLN database</td>
</tr>
<tr>
<td>'set &lt;switch&gt; &lt;value&gt;'</td>
<td>Set 'switch' to 'value'</td>
</tr>
<tr>
<td>'set'</td>
<td>Show current switch settings</td>
</tr>
<tr>
<td>'reload' or 'rl'</td>
<td>Reload prolog sources and lexicon</td>
</tr>
<tr>
<td>'print'</td>
<td>Prints results of last parse to current printer</td>
</tr>
<tr>
<td>'q' or 'quit'</td>
<td>Quits interface</td>
</tr>
<tr>
<td>'ref'</td>
<td>Refreshes the screen</td>
</tr>
<tr>
<td>'last'</td>
<td>Parses last discourse again</td>
</tr>
</tbody>
</table>

Among the commands, two particularly important ones are 'quit' (to quit the program) and 'set' (to set various parameters of the system). ‘help [command]’ will display online info about that command.

2.3.2.3 The Prolog Interpreter

The prolog interpreter\textsuperscript{16} is started by typing 'loop.'. The interpreter is similar to the perl interpreter in that it accepts both sentences and commands. The conventions for multiple sentences texts are similar to those used in the perl interpreter. The currently accepted commands are 'q' (for quit), 'h' (for 'help'), and 'ptq' (to run a test suite; currently being revised).

\textsuperscript{16}This code is derived from a program written for FraCaS by Steve Pulman.
2.3.2.4 Grammars and Grammar Switching

EKNTOOL comes with three grammars, fracas_main, fracas_undersp, and stdrt. It is possible to use any of those; the default is fracas_main. The default can be changed by setting the value of the environment variable EKNTOOLGRAMMAR.

To switch grammar from within the perl interpreter, do:

```perl
set grammar <grammar_name>
```

where `<grammar_name>` is one of the accepted grammars (currently).

From prolog, do:

```prolog
?- load_grammar(<grammar_name>).
```

2.3.2.5 Writing Permissions

In order to run the program you need writing permission in the directory from which the program is invoked, since ekntool creates various files and directories for its output. You also need to be able to overwrite the files "stdrt.tex", "done", and "disouts/file1" in your current directory.

2.3.3 The Main Grammar

The fracas_main grammar covers basic anaphora, quantification, negation, and tense. It can deal with simple discourses. Here are some of the sentences handled by the grammar:

```plaintext
john walks
smith walked.
mary arrived
a man walks
every representative ran
he leaves
every representative of a company left
a representative of every company arrived
a representative of nlpcom ran
every representative that owned a computer left
a representative that owned every workstation arrived
a company that owned every computer of nlpcom left
smith hired jones
a company likes robin
```
every customer likes smith
john owns a workstation
john hired every consultant
john hired every consultant who owns a computer
a man likes a woman
every customer likes a computer
every man likes every woman
john likes a man that likes bill
john likes a man that likes him
a company that liked john hired him
john likes a man that likes him
a man likes a man that likes him
a man likes every man that likes him
john doesn't walk
john does not walk
john didn't walk
john did not walk
john does not like a computer
every company that owns a computer likes it
every farmer likes a donkey who likes him
john does not like a man that likes him
john does not like a man that doesn't like him
john likes a man who does not like him.
a man that liked a man that didn't like him
john walks. he talks
a man walks. he talks.
every man walks. he talks.
john likes mary. she likes him
every man owns a workstation. he likes it.
every farmer who owns a donkey beats it. john beats it
anna doesn't own a workstation. she likes it.
if bill walks then he talks.
if a man walks then he talks
if every man walks then he talks
if a man likes bill then he hires him
if every man loves bill then he beats him
if a man likes a woman then he loves her
if a farmer owns a donkey then he beats it
if john likes bill then bill talks
if bill walks then he talks. he runs.
if a man walks then he talks. he runs.

2.3.4 Installation

2.3.4.1 The Files

The release 0.4 of ekntool consists of the following files:

- A README file.
- the perl scripts ekntool and showekn_perl
• the sicstus source files:

  1. the load files load.pl, load_pred.pl and main.pl;
  2. the files containing programs for operations on ekn expressions: ekn.pl,
     ekn_ops.pl, traverse.pl;
  3. the code for the conversion between prolog and \LaTeX, in display.pl and
     showekn.pl;
  4. the code for the interface between perl and prolog in perl_prolog.pl;
  5. various auxiliary files, in extras.pl, mp_util.pl, utils.pl
  6. the grammars, in the subdirectory grammars. Each grammar

     \texttt{<grammar\_name>}

     is in a directory with the same name. All grammars include a file
     called \texttt{load.pl} called by \texttt{ekntool}, a file called \texttt{lexicon.pl}, possibly a file called
     \texttt{db} that contains the test suite for the grammar, and additional files containing the
     actual code.

• the latex style files \texttt{ekn.sty}, which contains the definitions of the macros used to draw
  DRT-style boxes.

• the PLN database plndb.

The code is version-controlled using \texttt{rcs}; most of the source files are in the RCS directory, in
"\texttt{v}" form. They can be extracted using the "\texttt{co}" command.

2.3.4.2 Porting the System to a New Host

To move the code to another directory or system, you have to fix the \texttt{ekntool} perl script and
the \texttt{load_pred.pl} file. In particular, you have to

  1. modify the variables \texttt{$EKNTOOLHOME} and \texttt{$EKNTOOL\_LATEX} in the \texttt{perl} script.
  2. in \texttt{load_pred.pl}, you have to fix the definition of the predicate \texttt{ekntool\_home}.

Both of these changes are done automatically by editing the Makefile file in the top directory
and replacing the name of the variable \texttt{$EKNTOOLHOME}, then typing 'make' (see below).

The following programs are used by ekntool:

• sicstus prolog: it is invoked by calling "sicstus".
• latex: it is invoked by calling "latex".
• xdvii: it is invoked by calling "xdvi".

Check before using the system that all of these programs are accessible.
2.3.4.3 Installing the System

ekntool comes in a tar file. Save the tar file in the directory where you want the code to be located, and type

tar -xf <tar file name>

this will create a subdirectory called ekntool which, in turn, will have a subdirectory called grammars. Change directory to the directory ekntool, and edit the file Makefile changing the value of the variable EKNTOOLHOME to the value returned by typing pwd in the ekntool directory; then type

make

This will compile all the files, including the grammar files.

2.3.4.4 Common Problems

The prolog files should be compiled. This should take place automatically by running ‘make’ after untarring the files.

When making a change to one of the grammars, make sure to recompile the file before starting ‘ekntool’ again. This is especially true if the file to be modified is the ‘lexicon.pl’ file. If you add a new lexical entry for ‘pavarotti’ and also update the word list at the beginning of that file, ekntool will think that the word ‘pavarotti’ is in the lexicon, and will call prolog. The lack of a lexical item is especially bad because it makes the system backtrack to death, so you will get in this case a particularly unilluminating ‘Sicstus Error!’ message.

The perl interpreter and the prolog interpreter are kept synchronized by having the former wait for prolog to create a file called ‘done’ in the directory in which ekntool is executed. If prolog enters a non-terminating loop, the perl interpreter hangs waiting for the file being created. The same may happen if latex breaks when processing the input. In order to see what prolog or latex are doing, set the debug switch to 3. (See online help.)

When processing a sentence, the perl interpreter checks if all words are in the lexicon before sending the sentence to prolog. The interpreter expects to find the list of words in a file called lexicon.pl in the directory containing the current grammar. The list of words should be on a line initiated by \%??. A common problem is to add a word to the lexicon, but not to this list of words.
2.3.5 Writing a New Grammar

It is possible to add new grammars to the system. In order to add a new grammar, you have to create a subdirectory of the directory $EKNTOOLHOME/grammars. This directory must contain a file called load.pl that will load the files that define the grammar, as well as a file called lexicon.pl that contains at the beginning a specification of the lexicon. (This is only necessary if using the perl interpreter; see below.) The only constraint on the grammar is that it must define a three-places predicate

\[ \text{top}(X, M, H) \]

where \( X \) is the input sentence, \( M \) is the result of semantic interpretation in EKN format as defined in $EKNTOOLHOME/ekn.pl, and \( H \) is the history of the derivation (to be used by the system when the ‘verbose’ option is set). The predicate \( \text{top} \) will be invoked repeatedly by ekntool until all solutions have been found.

The input to the grammar is a list of words. The current convention is for periods (‘.’) in the natural language input to be converted into slashes (‘/’); thus, the sentence "Smith runs." will result in \( \text{top} \) being invoked with first argument \([\text{smith, runs,/]}.\)

The history list should be a list of the form:

\[ <hl> ::= [<component>+] \]

(note that it cannot be an empty list!) whereas each component is of the form:

\[ <component> ::= [<tag>, M, {<component>, <hl>}+] \]

The ‘tag’ can be anything you want. We use two different kinds of tags: one for syntactic categories, the other for semantic operations. The system doesn’t interpret those, it just prints them in bold font.

To use the new grammar from prolog it is sufficient to add a directory for the new grammar. In order to switch to the new grammar from ekntool (i.e., in order to be able to do

\[ \text{set grammar <grammar_name>} \]

from the perl interpreter) it will also be necessary to modify the procedure ‘switch’ in the ekntool code. switch first checks that the value of the switch is one of those allowed, then sets the parameters accordingly. In order to add a new grammar, it is necessary to add the grammar name to the list of values allowed.
A grammar may include a ‘test suite’ contained in a test file. The test suite for a grammar should be put in a file called db in that grammar's directory. The file should consist of a list of sentences, one for line. The number of readings for a sentence may also be specified at the end of the sentence, separated by a <TAB> from the end of the sentence. (See the db file in $EKNTOOLHOME for an example.) If the directory for a grammar includes a file called db, the perl interpreter loads that file and uses the test suite when the command test is used.

2.3.6 EKN in Prolog

Here is a brief description of how the EKN notation is converted into prolog notation.

We use prolog variables to represent EKN parameter constants and prolog constants to represent all other constants.\textsuperscript{17} Literals (the basic infons of Situation Theory) are represented by prolog terms of the form infon(\textit{Rel}, \textit{Args}, \textit{Pol}) where \textit{Rel} is a term denoting a relation, \textit{Args} is a list of terms and \textit{Pol} is the ‘polarity’ of the literal (0 or 1). For example, the literal \textit{meets}(\textit{X}, \textit{Y}) is represented by the prolog term infon(meets, [\textit{X}, \textit{Y}], 1).

Infons can be conjoined or disjoined. We use terms of the form /\[\textit{P}, \textit{Q}\] to indicate the conjunction of the infons \textit{P} and \textit{Q}, and terms of the form \textbackslash /[\textit{P}, \textit{Q}] to represent their disjunction.

Propositions are represented by terms of the form \textit{S} \textbackslash= \textit{I}, where \textit{S} is a term denoting a situation and \textit{I} is a code denoting an infon (which may be formed by conjoining or disjoining other infons). Propositions can also be conjoined and disjoined. Propositions can also be negated: the negation of a proposition \textit{P} is represented by the prolog term \texttt{~P}.

We represent assignments as either

- a prolog list whose elements are infixed terms of the form \textit{i}X where \textit{i} is an index and \textit{X} is a prolog term representing an EKN expression, or
- a prolog list whose elements are prolog terms representing EKN expressions.

An example of assignment of the first type is [\textit{i}X, \textit{j}Y]; of the second type, [\textit{X}, \textit{Y}]. Expressions of the second type are interpreted as representing assignments whose domain is the sequence of natural numbers from 1 up to the number of elements in the list. For example, [\textit{X}, \textit{Y}] is taken to be equivalent to [\textit{i}X, \textit{j}Y].

If \textit{A} is a prolog term representing an assignment and \textit{X} a prolog term representing an arbitrary EKN expression, then \texttt{A}~\texttt{~X} is the prolog term that represents an abstract. For example, the abstract represented in EKN as in (2.1a) is represented by the prolog term in (2.1b).

\textsuperscript{17}Parameter constants are implemented as prolog variables to take advantage of unification for doing substitutions in e.g. \(\beta\)-conversion. However, parameter constants could also be implemented as prolog constants (as in one of the \(\beta\)-conversion tools included in the framework library).
(2.1) a. $\lambda [i \rightarrow X \ j \rightarrow Y].\text{meets}(X, Y)$
b. $[i>\text{X}, j>\text{Y}]^\wedge \text{infon(meets, [X, Y], 1)}$

An application is represented by the prolog term $\text{X}\text{Y}$, where $\text{X}$ is a prolog term denoting an abstract and $\text{Y}$ a prolog term denoting an assignment.

Restricted objects are represented by prolog terms of the form $\text{X}/\text{Y}$, where $\text{X}$ is any prolog term representing an EKN expression and $\text{Y}$ is a proposition (not necessarily atomic).

2.3.7 Troubleshooting

When starting the per1 interpreter, the following sequence of events should occur:

1. the screen is cleared
2. ‘Starting sicstus process’ appears on the screen
3. ‘Getting current lexicon from lexicon.pl’ appears
4. The prompt appears.

If all this works, i.e., if you get the prompt ok, then before typing a sentence, you should type set debug 3. In this way, you'll get the reports from sicstus and from-leX.
Chapter 3

A Semantic Test Suite

How can the semantic competence of NLP systems be adequately measured? The question is important both from a practical and a theoretical perspective. Its practical importance derives from an increasingly felt need for fair and accurate means of comparing the quality of competing language processing systems — e.g. for purposes of funding, or of assessing the possibilities for commercial development — but also as a yardstick for the developers of such systems themselves. Test suites are theoretically important insofar as they force those who design them to think their way through to a clear conception of the semantic capacities that various types of NLP systems should have. In particular, we believe that test suite designers should have a well-developed idea of a core competence that ought to be common to all NLP systems that pretend to a genuine semantic component. That there should be such a core, that it should be substantial, and that it should go well beyond the semantic components of any NLP system currently running, is something of which we are firmly convinced.

In the light of the view expressed elsewhere in this and other FraCaS deliverables (see for instance section 4.(a).iii.A in Deliverable 16) that inferential ability is not only a central manifestation of semantic competence but is in fact centrally constitutive of it, it shouldn’t be a surprise that we regard inferencing tasks as the best way of testing an NLP system’s semantic capacity. Such tests take the following form: the system is given some natural language input $T$, then is presented with a claim $S$ and asked to determine whether $S$ follows from $T$. Since tests should be neutral as regards the semantic representation formalisms they use, the test claim $S$ should of course also be offered in the form of a natural language sentence.

Tasks of this sort form a continuum. At one end we find cases where the input $T$ consists of a small number of sentences (sometimes only one) which are directly relevant to the test sentence $S$, much in the style of problems in introductory logic books of the type: “Formalize the argument below, then derive the conclusion from the premises or else construct a counterexample”. Tasks at the other end of the spectrum rather resemble the problems one often finds in books that are used in the kind of “text analysis” that is nowadays often taught in high schools: the student is presented with a text and then with certain questions about
it. The questions vary, but some at least are yes-no questions the answer to which can be
determined by straight logical deduction from what the text actually says, questions such as:
“Did so-and-so do such-and-such before the new president was elected?”; “Is person $x$
related to person $y$?”; etc. On the whole the questions one finds in such tests are not quite as
simple-minded as those one would want to put to an NLP system in order to assess its logic.

But even when the questions they present are quite simple, tasks of this type tend to be
non-trivial insofar as they require searching through a large quantity of potential premises
and discarding all that is irrelevant. This is something with which human beings seem to have
comparatively little difficulty, but which seriously affects the efficiency of NLP systems with
the kind of architecture that most current systems have. Premise search is a problem that
will be familiar to anyone with some knowledge of theorem proving from large data bases.

While the difference between the two types of inferential tasks we have just mentioned is
obviously a matter of degree, it is a difference that should be firmly kept in mind when
perusing the list of “tests” given below. Some of the items on this list may seem very
elementary indeed, and the ability to solve them in the form in which they are presented here
— i.e. as tasks of the first kind — should perhaps not be counted for very much. But it is
quite a different thing to be able to confirm or reject inferences of the exemplified patterns
from substantial bits of text.

The complexity of this second kind of task grows not only with the size of the premise set, but
also with the variety of inference principles that the system has at its disposition. Already
a theorem prover that includes all inference schemata instantiated in our list faces a serious
complexity problem; but of course, this is only a small sample from the totality of inference
patterns that a competent natural language theorem prover should have at its command.

An additional complicating factor is that a fair test suite should contain not only valid infer-
ences (i.e. not only queries of the form “Does $S$ follow from the given input $T$?” to which
the answer is “yes”), but also invalid ones. Otherwise it would be vulnerable to conscious
and unconscious cheating. But when can the system be certain that $S$ does not follow from $T$;
how can it distinguish between cases where no proof of $S$ from $T$ exist and cases where it
hasn’t yet managed to find such a proof? We know that there can be no complete answer to
this question, for first order logic is undecidable and the expressive power of natural language
includes that of first order logic (and in fact goes well beyond it).

One could avoid this last problem somewhat by designing test suites for which it is guaranteed
that the queried conclusion $S$ either follows from $T$ or else is refutable from it. This fineses
the decision problem and might be recommendable at least as a first start. But in the long
run we should probably be more demanding. It seems to be part of the logical competence
of the human language user that he can not only see of many propositions that they follow
from a given input, but also that he can see of countless others that they do not follow.
We suspect that this latter ability is no less important in the semantic processing of natural
language than the ability to see that many things do follow. If so, then NLP systems should
be equipped with this facility too, and in that case it would be fair to test them for it.

Although these introductory remarks have been very sketchy and left a number of knots

64
untied, we hope they have made one thing clear: The list below is not only incomplete in that the great majority of inference patterns are not represented; it is also non-specific in that it leaves open precisely what a test should look like that is based on the patterns that are represented.

The inferences are grouped into linguistic and semantic phenomena loosely following the structure of the FraCaS deliverable D7:ch.3 Some Basic Linguistic Data and Their Importance and are presented in the form of a list of premises followed by a query followed by an answer and in some cases comments. Unless indicated otherwise the premises constitute the only source of information available for the inference.

3.1 Generalized quantifiers

3.1.1 Conservativity

Q As are Bs ↔ Q As are As who are Bs

(3.1) An Italian became the world’s greatest tenor.
     Was there an Italian who became the world’s greatest tenor?
     [Yes]

(3.2) Every Italian man wants to be a great tenor.
     Some Italian men are great tenors.
     Are there Italian men who want to be a great tenor?
     [Yes]

(3.3) All Italian men want to be a great tenor.
     Some Italian men are great tenors.
     Are there Italian men who want to be a great tenor?
     [Yes]

(3.4) Each Italian tenor wants to be great.
     Some Italian tenors are great.
     Are there Italian tenors who want to be great?
     [Yes]

(3.5) The really ambitious tenors are Italian.
     Are there really ambitious tenors who are Italian?
     [Yes]
(3.6) No really great tenors are modest.
Are there really great tenors who are modest? [No]

(3.7) Some great tenors are Swedish.
Are there great tenors who are Swedish? [Yes]

(3.8) Many great tenors are German.
Are there great tenors who are German? [Yes]

(3.9) Several great tenors are British.
Are there great tenors who are British? [Yes]

(3.10) Most great tenors are Italian.
Are there great tenors who are Italian? [Yes]

(3.11) A few great tenors sing popular music.
Some great tenors like popular music.
Are there great tenors who sing popular music? [Yes]

(3.12) Few great tenors are poor.
Are there great tenors who are poor? [Not many]

(3.13) Both leading tenors are excellent.
Leading tenors who are excellent are indispensable.
Are both leading tenors indispensable? [Yes]

(3.14) Neither leading tenor comes cheap.
One of the leading tenors is Pavarotti.
Is Pavarotti a leading tenor who comes cheap? [No]
(3.15) At least three tenors will take part in the concert.
Are there tenors who will take part in the concert?  
[Yes]  

(3.16) At most two tenors will contribute their fees to charity.
Are there tenors who will contribute their fees to charity?  
[At most two]  

3.1.2 Monotonicity (upwards on second argument)  

Q As are Bs and all Bs are Cs → Q As are Cs  

(3.17) An Irishman won the Nobel prize for literature.
Did an Irishman win a Nobel prize?  
[Yes]  

(3.18) Every European has the right to live in Europe.
Every European is a person.
Every person who has the right to live in Europe can travel freely within Europe.
Can every European travel freely within Europe?  
[Yes]  

(3.19) All Europeans have the right to live in Europe.
Every European is a person.
Every person who has the right to live in Europe can travel freely within Europe.
Can all Europeans travel freely within Europe?  
[Yes]  

(3.20) Each European has the right to live in Europe.
Every European is a person.
Every person who has the right to live in Europe can travel freely within Europe.
Can each European travel freely within Europe?  
[Yes]  

67
(3.21) The residents of member states have the right to live in Europe. All residents of member states are individuals. Every individual who has the right to live in Europe can travel freely within Europe. Can the residents of member states travel freely within Europe? [Yes]

(3.22) No delegate finished the report on time. Did no delegate finish the report? [Don't know]

(3.23) Some delegates finished the survey on time. Did some delegates finish the survey? [Yes]

(3.24) Many delegates obtained interesting results from the survey. Did many delegates obtain results from the survey? [Yes]

(3.25) Several delegates got the results published in major national newspapers. Did several delegates get the results published? [Yes]

(3.26) Most Europeans are resident in Europe. All Europeans are people. All people who are resident in Europe can travel freely within Europe. Can most Europeans travel freely within Europe? [Yes]

(3.27) A few committee members are from Sweden. All committee members are people. All people who are from Sweden are from Scandinavia. Are at least a few committee members from Scandinavia? [Yes]
(3.28) Few committee members are from Portugal.
All committee members are people.
All people who are from Portugal are from southern Europe.
Are there few committee members from southern Europe?
[Don’t know]

(3.29) Both commissioners used to be leading businessmen.
Did both commissioners used to be businessmen?
[Yes]

(3.30) Neither commissioner spends a lot of time at home.
Does neither commissioner spend time at home?
[Don’t know]

(3.31) A least three commissioners spend a lot of time at home.
Do at least three commissioners spend time at home?
[Yes]

(3.32) At most ten commissioners spend a lot of time at home.
Do at most ten commissioners spend time at home?
[Don’t know]

3.1.3 Monotonicity (downwards on second argument)

Q As are Bs and all Cs are Bs → Q As are Cs

(3.33) An Irishman won a Nobel prize.
Did an Irishman win the Nobel prize for literature?
[Don’t know]

(3.34) Every European can travel freely within Europe.
Every European is a person.
Every person who has the right to live in Europe
can travel freely within Europe.
Does every European have the right to live in Europe?
[Don’t know]
(3.35) All Europeans can travel freely within Europe.
Every European is a person.
Every person who has the right to live in Europe can travel freely within Europe.

Do all Europeans have the right to live in Europe?
[Don’t know]

(3.36) Each European can travel freely within Europe.
Every European is a person.
Every person who has the right to live in Europe can travel freely within Europe.

Does each European have the right to live in Europe?
[Don’t know]

(3.37) The residents of member states can travel freely within Europe.
All residents of member states are individuals.
Every individual who has the right to live anywhere in Europe can travel freely within Europe.

Do the residents of member states have the right to live anywhere in Europe?
[Don’t know]

(3.38) No delegate finished the report.

Did any delegate finish the report on time?
[No]

(3.39) Some delegates finished the survey.

Did some delegates finish the survey on time?
[Don’t know]

(3.40) Many delegates obtained results from the survey.

Did many delegates obtain interesting results from the survey?
[Don’t know]

(3.41) Several delegates got the results published.

Did several delegates get the results published in major national newspapers?
[Don’t know]
Most Europeans can travel freely within Europe.
All Europeans are people.
All people who are resident in Europe
can travel freely within Europe.
Are most Europeans resident in Europe?
[Don’t know]

A few committee members are from Scandinavia.
All committee members are people.
All people who are from Sweden are from Scandinavia.
Are at least a few committee members from Sweden?
[Don’t know]

Few committee members are from southern Europe.
All committee members are people.
All people who are from Portugal are from southern Europe.
Are there few committee members from Portugal?
[Yes]

Both commissioners used to be businessmen.
Did both commissioners used to be leading businessmen?
[Don’t know]

Neither commissioner spends time at home.
Does either commissioner spend a lot of time at home?
[No]

A least three commissioners spend time at home.
Do at least three commissioners spend a lot of time at home?
[Don’t know]

At most ten commissioners spend time at home.
Do at most ten commissioners spend a lot of time at home?
[Yes]

3.1.4 Monotonicity (upwards on first argument)

Q As are Bs and all As are Cs → Q Cs are Bs
(3.49) A Swede won a Nobel prize.

Every Swede is a Scandinavian.

Did a Scandinavian win a Nobel prize?

[Yes]

(3.50) Every Canadian resident can travel freely within Europe.

Every Canadian resident is a resident of the North American continent.

Can every resident of the North American continent travel freely within Europe?

[Don’t know]

(3.51) All Canadian residents can travel freely within Europe.

Every Canadian resident is a resident of the North American continent.

Can all residents of the North American continent travel freely within Europe?

[Don’t know]

(3.52) Each Canadian resident can travel freely within Europe.

Every Canadian resident is a resident of the North American continent.

Can each resident of the North American continent travel freely within Europe?

[Don’t know]

(3.53) The residents of major western countries can travel freely within Europe.

All residents of major western countries are residents of western countries.

Do the residents of western countries have the right to live in Europe?

[Don’t know]

(3.54) No Scandinavian delegate finished the report on time.

Did any delegate finish the report on time?

[Don’t know]

(3.55) Some Irish delegates finished the survey on time.

Did any delegates finish the survey on time?

[Yes]
(3.56) Many British delegates obtained interesting results from the survey. Did many delegates obtain interesting results from the survey? [Don’t know]

(3.57) Several Portuguese delegates got the results published in major national newspapers. Did several delegates get the results published in major national newspapers? [Yes]

(3.58) Most Europeans who are resident in Europe can travel freely within Europe. Can most Europeans travel freely within Europe? [Don’t know]

(3.59) A few female committee members are from Scandinavia. Are at least a few committee members from Scandinavia? [Yes]

(3.60) Few female committee members are from southern Europe. Are few committee members from southern Europe? [Don’t know]

(3.61) Both female commissioners used to be in business. Did both commissioners used to be in business? [Yes, if both commissioners are female. Otherwise there are more than two commissioners]

(3.62) Neither female commissioner spends a lot of time at home. Does either commissioner spend a lot of time at home? [No, if both commissioners are female. Otherwise there are more than two commissioners]

(3.63) At least three female commissioners spend time at home. Do at least three commissioners spend time at home? [Yes]
At most ten female commissioners spend time at home.

Do at most ten commissioners spend time at home?

[Don’t know]

### 3.1.5 Monotonicity (downwards on first argument)

\( Q \) \( A \)s are \( B \)s and all \( C \)s are \( A \)s \( \rightarrow \) \( Q \) \( C \)s are \( B \)s

(3.65) A Scandinavian won a Nobel prize.

**Every Swede is a Scandinavian**

Did a Swede win a Nobel prize?

[Don’t know]

(3.66) Every resident of the North American continent can travel freely within Europe.

**Every Canadian resident is a resident of the North American continent.**

Can every Canadian resident freely within Europe?

[Yes]

(3.67) All residents of the North American continent can travel freely within Europe.

**Every Canadian resident is a resident of the North American continent.**

Can all Canadian residents travel freely within Europe?

[Yes]

(3.68) Each resident of the North American continent can travel freely within Europe.

**Every Canadian resident is a resident of the North American continent.**

Can each Canadian resident travel freely within Europe?

[Yes]

(3.69) The residents of western countries can travel freely within Europe.

All residents of major western countries are residents of western countries.

Do the residents of major western countries have the right to live in Europe?

[Yes]
(3.70) No delegate finished the report on time.
Did any Scandinavian delegate finish the report on time?
[No]

(3.71) Some delegates finished the survey on time.
Did any Irish delegates finish the survey on time?
[Don’t know]

(3.72) Many delegates obtained interesting results from the survey.
Did many British delegates obtain interesting results from the survey?
[Don’t know]

(3.73) Several delegates got the results published in major national newspapers.
Did several Portuguese delegates get the results published in major national newspapers?
[Don’t know]

(3.74) Most Europeans can travel freely within Europe.
Can most Europeans who are resident outside Europe travel freely within Europe?
[Don’t know]

(3.75) A few committee members are from Scandinavia.
Are at least a few female committee members from Scandinavia?
[Don’t know]

(3.76) Few committee members are from southern Europe.
Are few female committee members from southern Europe?
[Yes]

(3.77) Both commissioners used to be in business.
Did both female commissioners used to be in business?
[Yes, if both commissioners are female. Otherwise there are more than two commissioners]
(3.78) Neither commissioner spends a lot of time at home.
Does either female commissioner spend a lot of time at home?
[No, if both commissioners are female. Otherwise there are more than two commissioners]

(3.79) A least three commissioners spend time at home.
Do at least three male commissioners spend time at home?
[Don’t know]

(3.80) At most ten commissioners spend time at home.
Do at most ten female commissioners spend time at home?
[Yes]

3.2 Plurals

A number of inferences pertaining to plurals are covered under the headings of generalized quantifiers and elsewhere. Here we concentrate on conjoined NPs; bare, existential and definite plurals; dependent plurals; and collective and distributive readings and scope ambiguity.

3.2.1 Conjoined Noun Phrases

(3.81) Smith, Jones and Anderson signed the contract.
Did Jones sign the contract? [Yes]

(3.82) Smith, Jones and several lawyers signed the contract.
Did Jones sign the contract? [Yes]

(3.83) Either Smith, Jones or Anderson signed the contract.
Did Jones sign the contract? [Don’t know]

(3.84) Either Smith, Jones or Anderson signed the contract.
If Smith and Anderson did not sign the contract, did Jones sign the contract? [Yes]
Exactly two lawyers and three accountants signed the contract.

Did six lawyers sign the contract? [No]

No scope relations between the two conjoined NPs

Exactly two lawyers and three accountants signed the contract.

Did six accountants sign the contract? [No]

No scope relations between the two conjoined NPs

Conjoined Nbars

Nbar conjunction tends to be quite ambiguous. This may be the result of a syntactic ambiguity between (i) genuine Nbar conjunction, and (ii) NP conjunction where the determiner of one of the NPs is ellided.

Every representative and client was at the meeting.

Was every representative at the meeting? [Yes, on one reading]

Arguably NP conjunction: every representative and every client

Every representative and client was at the meeting.

Was every representative at the meeting? [Don’t know, on one reading]

NBar conjunction: everyone who is both a representative and a client

Every representative or client was at the meeting.

Was every representative and every client at the meeting? [Yes]

With disjunction, NP conjunction seems unavailable.

3.2.2 Definite Plurals

Definite plurals can often be non-anaphoric and behave like universally quantified noun phrases (3.90). However, as with (generic) bare plurals, the force of the quantification can also be less than universal (3.91). Whether this lessening of quantificational force is due to the noun phrase or to the predicate of which the NP is an argument is unclear (3.92, 3.93).
(3.90) The chairman read out the items on the agenda.
Did the chairman read out every item on the agenda?

[Yes]

Non-anaphoric, universal plural definite

(3.91) The people who were at the meeting voted for a new chairman.
Did everyone at the meeting vote for a new chairman?

[Don’t know]

Some people may have abstained from the vote

(3.92) All the people who were at the meeting voted for a new chairman.
Did everyone at the meeting vote for a new chairman?

[Yes]

(3.93) The people who were at the meeting all voted for a new chairman.
Did everyone at the meeting vote for a new chairman?

[Yes]

Closely related to this, plural definites can have a collective/institutional or even generic interpretation:

(3.94) The inhabitants of Cambridge voted for a Labour MP.
Did every inhabitant of Cambridge vote for a Labour MP?

[Don’t know]

(3.95) The Ancient Greeks were noted philosophers.
Was every Ancient Greek a noted philosopher?

[Don’t know]

(3.96) The Ancient Greeks were all noted philosophers.
Was every Ancient Greek a noted philosopher?

[Yes]
3.2.3 Bare Plurals

Bare plurals can exhibit existential, (quasi) universal, generic or dependent plural behaviour. The circumstances giving rise to these different behaviours a poorly understood, so we only give a few illustrative examples.

(3.97) Software faults were blamed for the system failure.
   
   Was the system failure blamed on one or more software faults? [Yes]
   E

   Existential bare plural

(3.98) Software faults were blamed for the system failure.
   Bug # 32-985 is a known software fault.
   
   Was Bug # 32-985 blamed for the system failure? [Don’t know]
   E

   Existential interpretation: not every software fault contributed to the failure.

(3.99) Clients at the demonstration were all impressed by the system’s performance.
   Smith was a client at the demonstration.
   
   Was Smith impressed by the system’s performance? [Yes]

   (Quasi) universal bare plural

(3.100) Clients at the demonstration were impressed by the system’s performance.

   Were most clients at the demonstration impressed by the system’s performance? [Yes]

   (Quasi) universal bare plural

(3.101) University graduates make poor stock-market traders.
   Smith is a university graduate.
   
   Is Smith likely to make a poor stock market trader? [Yes]

   Generic interpretation
(3.102) University graduates make poor stock-market traders.  
Smith is a university graduate.  
Will Smith make a poor stock market trader?  
[Don’t know]

Generic interpretation

3.2.4 Dependent Plurals

(3.103) All APCOM managers have company cars.  
Jones is an APCOM manager.  
Does Jones have a company car?  
[Yes]

(3.104) All APCOM managers have company cars.  
Jones is an APCOM manager.  
Does Jones have more than one company car?  
[Don’t know]

3.2.5 Negated Plurals

(3.105) Just one accountant attended the meeting.  
Did no accountants attend the meeting?  
[No]

(3.106) Just one accountant attended the meeting.  
Did no accountant attend the meeting?  
[No]

(3.107) Just one accountant attended the meeting.  
Did any accountants attend the meeting?  
[Yes]

(3.108) Just one accountant attended the meeting.  
Did any accountant attend the meeting?  
[Yes]
3.2.6 Collective and Distributive Plurals

(3.111) Smith signed one contract.
Jones signed another contract.

\[
\text{Did Smith and Jones sign two contracts?} \\
\text{[Yes, on a collective/cumulative reading of the conclusion]}
\]

(3.112) Smith signed two contracts.
Jones signed two contracts.

\[
\text{Did Smith and Jones sign two contracts?} \\
\text{[Yes, on a distributive reading of “Smith and Jones”]}
\]

(3.113) Smith signed two contracts.
Jones also signed them.

\[
\text{Did Smith and Jones sign two contracts?} \\
\text{[Yes]}
\]

3.3 (Nominal) Anaphora

In the examples below we make the assumption (unless otherwise indicated) that there is no context beyond that provided by the mini-discourse. This is so that we can do away with explicit co-indexing of pronouns with their antecedents, on the grounds that context will provide only (or sometimes a strictly limited number) of possible antecedents.
3.3.1 Intra-Sentential

(3.114) Mary used her workstation.
Was Mary’s workstation used?  
[Yes]

(3.115) Mary used her workstation.
Does Mary have a workstation?  
[Yes]

(3.116) Mary used her workstation.
Is Mary female?  
[Yes]

(3.117) Every student used her workstation.  
Mary is a student.  
Did Mary use her workstation?  
[Yes]

(3.118) Every student used her workstation.  
Mary is a student.  
Does Mary have a workstation?  
[Yes]

(3.119) No student used her workstation.  
Mary is a student.  
Did Mary use a workstation?  
[No]
3.3.2 Inter-Sentential

(3.120) Smith attended a meeting.
She chaired it.

Did Smith chair a meeting?
[Yes]

(3.121) Smith delivered a report to ITEL.
She also delivered them an invoice.
And she delivered them a project proposal.

Did Smith deliver a report, an invoice and a project proposal to ITEL?
[Yes]

Keeping track of same entities across more than two sentences.

(3.122) Every committee has a chairman.
He is appointed its members.

Does every committee have a chairman appointed by members of the committee?
[Yes]

Modal subordination.

3.3.3 Plural Anaphora

(3.123) ITEL has sent most of the reports Smith needs.
They are on her desk.

Are there some reports from ITEL on Smith’s desk?
[Yes]

(3.124) Two out of ten machines are missing.
They have been removed.

Have two machines been removed?
[Yes]
(3.125) Two out of ten machines are missing.
They have been removed.

Have eight machines been removed?
[Don’t know]

Set difference can’t be used to construct plural antecedents

(3.126) Two out of ten machines are missing.
They were all here yesterday.

Were ten machines here yesterday?
[Yes]

(3.127) Smith took a machine on Tuesday, and Jones took a machine on Wednesday.
They put them in the lobby.

Did Smith and Jones put two machines in the lobby?
[Yes, on a distributive reading of the question]

Construction of plural antecedents from separate constituents.

(3.128) John and his colleagues went to a meeting.
They hated it

Did John’s colleagues hate the meeting?
[Yes]

(3.129) John and his colleagues went to a meeting.
They hated it

Did John hate the meeting?
[Yes, on one possible reading]

“They” can be resolved to John and his colleagues

(3.130) John and his colleagues went to a meeting.
They hated it

Did John hate the meeting?
[Don’t know, on one possible reading]

“They” can also be resolved to John’s colleagues but not John
(3.131) Each department has a dedicated line.
They rent them from BT.

Does every department rent a line from BT? [Yes]

Dependent plural anaphora

(3.132) Each department has a dedicated line.
The sales department rents it from BT.

Does the sales department rent a line from BT? [Yes]

Paycheque pronoun

3.3.4 E-type and Donkey Pronouns

(3.133) GFI owns several computers.
ITEL maintains them.

Does ITEL maintain all the computers that GFI owns? [Yes]

E-type anaphora

(3.134) Every customer who owns a computer has a service contract for it.
MFI is a customer that owns exactly one computer.

Does MFI have a service contract for all its computers? [Yes]

Donkey sentence

(3.135) Every customer who owns a computer has a service contract for it.
MFI is a customer that owns several computers.

Does MFI have a service contract for all its computers? [Yes]

This pattern of inference, unlike (3.134), tends to some theory dependence. Although the inference seems correct in this example, compare with (3.136)

(3.136) Every executive who had a laptop computer brought it to take notes at the meeting.
Smith is a executive who owns five different laptop computers.

Did Smith take five laptop computers to the meeting? [Don’t know]

85
Similar to (3.135), except for tense and pragmatic plausibility.

(3.137) There are 100 companies.
ICM is one of the companies and owns 150 computers.
It does not have service contracts for any of its computers.
Each of the other 99 companies owns one computer.
They have service contracts for them.

Do most companies that own a computer have a service contract for it? [Yes]

Proportion problem

3.3.5 Functional and Subsectional

Due to the heavy domain dependence of functional (or better perhaps, relational) anaphora, it is hard to state general inferences that don’t assume considerable background knowledge unless this is given explicitly.

(3.138) Every report has a cover page.
R-95-103 is a report.
Smith signed the cover page.

Did Smith sign the cover page of R-95-103? [Yes]

3.3.6 Simple Reflexives

(3.139) A company director awarded himself a large payrise.

Has a company director awarded and been awarded a payrise? [Yes]

(3.140) John said Bill had hurt himself.

Did John say Bill had been hurt? [Yes]
(3.141) John said Bill had hurt himself.
Did anyone say John had been hurt?

[Don’t know]

3.4 Ellipsis

In nearly all cases the inferences presented here have conclusions that are simply reconstructions of the ellided constituent. Unfortunately, an inference test suite is not well suited to illustrating prohibitions on ellipsis resolution. For example, an ill-formed discourse like

John was in Paris yesterday. *So did Bill.

doesn’t even get as far as supporting any inferences.

3.4.1 VP Ellipsis

(3.142) John spoke to Mary.
So did Bill
Did Bill speak to Mary?
[Yes]

Basic example.

(3.143) John spoke to Mary.
So did Bill
John spoke to Mary at four o’clock.
Did Bill speak to Mary at four o’clock?
[Don’t know]

Temporal resolution of tense in antecedent is not carried across to ellipsis.

(3.144) John spoke to Mary at four o’clock.
So did Bill
Did Bill speak to Mary at four o’clock?
[Yes]

Explicit temporal adverbials are carried across to ellipsis.
(3.145) John spoke to Mary at four o’clock.
And Bill did at five o’clock
Did Bill speak to Mary at five o’clock? [Yes]

*Explicit temporal adverbials are not carried across if overridden.*

(3.146) John has spoken to Mary.
Bill is going to.
Will Bill speak to Mary? [Yes]

*Tense agreement not necessary between ellipsis and antecedent.*

(3.147) John spoke to Mary on Monday.
Bill didn’t.
Did Bill speak to Mary on Monday? [No]

*Polarity agreement not necessary between ellipsis and antecedent.*

(3.148) Has John spoken to Mary?
Bill has.
Has Bill spoken to Mary? [Yes]

*Mood agreement not necessary between ellipsis and antecedent.*

(3.149) John has spoken to Mary.
The students have too.
Have the students spoken to Mary? [Yes]

*Number agreement not necessary.*

3.4.2 Gapping

(3.150) John went to Paris by car, and Bill by train.
Did Bill go to Paris by train? [Yes]

*Basic example*
(3.151) John went to Paris by car, and Bill by train to Berlin.

Did Bill go to Berlin by train? [Yes]

Another basic example

(3.152) John went to Paris by car, and Bill to Berlin.

Did Bill go to Berlin by car? [Yes]

Another basic example

(3.153) John is going to Paris by car, and the students by train.

Are the students going to Paris by train? [Yes]

Subject-verb agreement not necessary

(3.154) John went to Paris by car.

Bill by train.

Did Bill go to Paris by train? [Yes]

Cross-sentential gapping

3.4.3 One Anaphora

(3.155) John owns a car.

Bill owns one too.

Does Bill own a car? [Yes]

Basic example

(3.156) John owns a car.

Bill owns one too.

Is there a car that John and Bill own? [Don't know]

It needn't be the same car that John and Bill own.
(3.157) John owns a red car.
     Bill owns a blue one.
Does Bill own a blue car?
     [Yes]

(3.158) John owns a red car.
     Bill owns a blue one.
Does Bill own a red car?
     [Don't know]

(3.159) John owns a red car.
     Bill owns a fast one.
Does Bill own a fast car?
     [Yes]

(3.160) John owns a red car.
     Bill owns a fast one.
Does Bill own a fast red car?
     [Yes, on one possible reading]
The one anaphor may be resolved via the property of being a red car

(3.161) John owns a red car.
     Bill owns a fast one.
Does Bill own a fast red car?
     [Don't know, on one possible reading]
Or the one anaphor may just be resolved via the property of being a car

(3.162) John owns a fast red car.
     Bill owns a slow one.
Does Bill own a slow red car?
     [Yes]
When semantically parallel (e.g. fast/slow) modifiers appear on the antecedent and one-anaphor, it appears that all non-parallel modifiers must form part of the resolution
3.4.4 Sluicing

(3.163) John had his paper accepted.
Bill doesn’t know why.
Does Bill know why John had his paper accepted?
[No]

3.4.5 Phrasal Ellipsis

(3.164) John spoke to Mary.
And to Sue.
Did John speak to Sue?
[Yes]
*PP ellipsis (subcategorized)*

(3.165) John spoke to Mary.
On Friday.
Did John speak to Mary on Friday?
[Yes]
*PP ellipsis: adds PP to antecedent*

(3.166) John spoke to Mary on Thursday.
And on Friday.
Did John speak to Mary on Friday?
[Yes]
*PP ellipsis: replaces PP in antecedent*

(3.167) Twenty men work in the Sales Department.
But only one woman.
Do two women work in the Sales Department?
[No]
*NP ellipsis*
(3.168) Five men work part time.  
And forty five women.  
______________________________  
Do forty five women work part time?  
[Yes]  
*NP ellipsis*

(3.169) John found Mary before Bill.  
______________________________  
Did John find Mary before Bill found Mary?  
[Yes, on one possible reading]  
*NP ellipsis*

(3.170) John found Mary before Bill.  
______________________________  
Did John find Mary before John found Bill?  
[Yes, on one possible reading]  
*NP ellipsis*

(3.171) John wants to know how many men work part time.  
And women.  
______________________________  
Does John want to know how many women work part time?  
[Yes]  
*Nbar ellipsis*

(3.172) John wants to know how many men work part time, and which.  
______________________________  
Does John want to know which men work part time?  
[Yes]  
*Determiner ellipsis*

3.4.6 Antecedent Contained Deletion

Antecedent contained deletion is a notorious problem for copying approaches to ellipsis, since the antecedent clause contains the ellipsis and some way must be found of removing it from the copy.

(3.173) Bill spoke to everyone that John did.  
John spoke to Mary.  
______________________________  
Did Bill speak to Mary?  
[Yes]
(3.174)  Bill spoke to everyone that John did.
Bill spoke to Mary.

Did John speak to Mary?  
[Don’t know]

3.4.7 Configurational Effects

There are a number of syntactic and other configurational constraints on what can constitute the antecedent to an ellipsis. These constraints varying depending on the type of ellipsis (VP, phrasal, gapping, etc).

(3.175)  John said Mary wrote a report, and Bill did too.

Did Bill say Mary wrote a report?  
[Yes, on one possible reading/parse]

(3.176)  John said Mary wrote a report, and Bill did too.

Did John say Bill wrote a report?  
[Yes, on one possible reading/parse]

(3.177)  John said that Mary wrote a report, and that Bill did too.

Did Bill say Mary wrote a report?  
[Don’t know]

Note that the first sentence in (3.175) and (3.176) is syntactically ambiguous, depending on whether the conjunctive clause conjoins with the main or subordinate clause of John said Mary wrote a report. In (3.177) the conjunctive clause unambiguously conjoins with the subordinate clause, and only one interpretation of the ellipsis is possible. This appears to indicate that the antecedent clause to a VP ellipsis must be adjacent to the elliptical clause. However, as the examples below show, this is not correct.

(3.178)  John wrote a report, and Bill said Peter did too.

Did Bill say Peter wrote a report?  
[Yes]

*Embedded elliptical clause*
(3.179) If John wrote a report, then Bill did too.

John wrote a report.

Did Bill write a report? [Yes]

Elliptical and antecedent clause embedded (in parallel)

(3.180) John wanted to buy a car, and he did.

Did John buy a car? [Yes]

Embedded antecedent clause

(3.181) John needed to buy a car, and Bill did.

Did Bill buy a car? [Don’t know]

Other configurational effects of the kinds illustrated in Deliverable 7 are hard to exemplify using inference suites.

3.4.8 Ellipsis and Anaphora

The following inferences illustrate differences between strict and sloppy interpretations of anaphors in elliptical clauses

(3.182) Smith represents his company and so does Jones.

Does Jones represent Jones’ company? [Yes, on one reading]

Sloppy identity

(3.183) Smith represents his company and so does Jones.

Does Jones represent Smith’s company? [Yes, on one reading]

Strict identity

(3.184) Smith represents his company and so does Jones.

Does Smith represent Jones’ company? [Don’t know]
(3.185) Smith claimed he had costed his proposal and so did Jones.

Did Jones claim he had costed his own proposal? [Yes, on one reading]

*Sloppy identity on both pronouns*

(3.186) Smith claimed he had costed his proposal and so did Jones.

Did Jones claim he had costed Smith's proposal? [Yes, on one reading]

*Sloppy identity “he”, strict on “his”*

(3.187) Smith claimed he had costed his proposal and so did Jones.

Did Jones claim Smith had costed Smith's proposal? [Yes, on one reading]

*Strict identity on both pronouns*

(3.188) Smith claimed he had costed his proposal and so did Jones.

Did Jones claim Smith had costed Jones' proposal? [Don't know]

*Can't have strict identity on “he” and sloppy identity on “his”*

(3.189) John is a man and Mary is a woman.

John represents his company and so does Mary.

Does Mary represent her own company? [Yes, on one reading]

*Sloppy identity, gender agreement not necessary*

(3.190) John is a man and Mary is a woman.

John represents his company and so does Mary.

Does Mary represent John's company? [Yes, on one reading]

*Strict identity, gender agreement not necessary*
(3.191)  Bill suggested to Frank’s boss that they should go to the meeting together, and Carl to Alan’s wife.

If it was suggested that Bill and Frank should go together, was it suggested that Carl and Alan should go together?  

[Yes]

*Plural pronouns resolved in parallel*

(3.192)  Bill suggested to Frank’s boss that they should go to the meeting together, and Carl to Alan’s wife.

If it was suggested that Bill and Frank should go together, was it suggested that Carl and Alan’s wife should go together?  

[Don’t know]

*Plural pronouns resolved in parallel*

(3.193)  Bill suggested to Frank’s boss that they should go to the meeting together, and Carl to Alan’s wife.

If it was suggested that Bill and Frank’s boss should go together, was it suggested that Carl and Alan’s wife should go together?  

[Yes]

*Plural pronouns resolved in parallel*

(3.194)  Bill suggested to Frank’s boss that they should go to the meeting together, and Carl to Alan’s wife.

If it was suggested that Bill and Frank’s boss should go together, was it suggested that Carl and Alan should go together?  

[Don’t know]

*Plural pronouns resolved in parallel*

(3.195)  Bill suggested to Frank’s boss that they should go to the meeting together, and Carl to Alan’s wife.

If it was suggested that Bill, Frank and Frank’s boss should go together, was it suggested that Carl, Alan and Alan’s wife should go together?  

[Yes]

*Plural pronouns resolved in parallel*

### 3.4.9 Ellipsis and Quantification

Scope parallelism turns out to be rather tricky to illustrate through inference suites. This is because of the entailment relation: \( \exists v \models \forall \exists \).
(3.196) A lawyer signed every report, and so did an auditor.
That is, there was one lawyer who signed all the reports.
Was there one auditor who signed all the reports? [Yes]

3.5 Adjectives

The inferences below carve up adjectives into (a by no means exhaustive) cross-cutting set of
dimensions. Typical inferences are given for example adjectives.

3.5.1 Affirmative and Non-Affirmative

Affirmative adjectives map the denotation of the predicate they modify onto a subset of the
denotation. So for example, an old man is a man. Most adjectives are affirmative, but a few
like former and fake are not. Given that someone is a former student, one cannot conclude
that they are now a student. But it is not entirely clear whether one can conclude that they
are not now a student — they may have become one again.

(3.197) John has a genuine diamond.
Does John have a diamond? [Yes]
Affirmative adjectives: Adj N \(\Rightarrow\) N

(3.198) John is a former university student.
Is John a university student? [No/Don’t know]
Non-affirmative: Adj N \(\not\Rightarrow\) N
(Opinions differ about whether “Adj N \(\Rightarrow\) \(\neg\)N”)

(3.199) John is a successful former university student.
Is John successful? [Yes (for a former university student)]
Ordering between affirmative and non-affirmative adjectives affects which adjectival
predications are and aren’t affirmed

97
(3.200) John is a former successful university student.
Is John successful?
[Don’t know]

(3.201) John is a former successful university student.
Is John a university student?
[Don’t know]
*John may currently be an unsuccessful university student*

### 3.5.2 No Comparison Class

Gradable adjectives (e.g. *big, small*) usually assume some form of comparison class (i.e. ‘big for an N’). But some others do not e.g. *four-legged*, or the adjectival phrase *ten foot long*. Adjectives not requiring a comparison class permit straightforward predication without reference to a nominal property providing a comparison class: a ten foot long alligator is ten foot long.

(3.202) Every mammal is an animal.
Is every four-legged mammal a four-legged animal?
[Yes]

\[ N1 \rightarrow N2 \Rightarrow [Adj(N1) \rightarrow Adj(N2)] \]

(3.203) Dumbo is a four-legged animal.
Is Dumbo four-legged?
[Yes]

\[ Adj(N)(x) \Rightarrow Adj(x) \]

### 3.5.3 Opposites

*Large* and *small* (applied to the same comparison class) are opposites. If something is a small N it cannot be a large N, and vice versa. Some things can be neither large nor small Ns.

(3.204) Mickey is a small animal.
Is Mickey a large animal?
[No]

\[ Small(N) \Rightarrow \neg Large(N) \]
(3.205) Dumbo is a large animal.

Is Dumbo a small animal?  

[No]

Large(N) ⇒ ¬Small(N)

(3.206) Fido is not a small animal.

Is Fido a large animal?  

[Don’t know]

¬Small(N) ∨ Large(N)

(3.207) Fido is not a large animal.

Is Fido a small animal?  

[Don’t know]

¬Large(N) ∨ Small(N)

(3.208) Mickey is a small animal.
Dumbo is a large animal

Is Mickey smaller than Dumbo?  

[Yes]

“Small” and “large” are related via the comparative “smaller”

(3.209) Mickey is a small animal.
Dumbo is a large animal

Is Mickey larger than Dumbo?  

[No]

“Small” and “large” are related via the comparative “larger”

3.5.4 Extensional Comparison Classes

Adjectives like large and small depend only on the extension of the comparison class they depend on.

(3.210) All mice are small animals.
Mickey is a large mouse.

Is Mickey a large animal?  

[No]
(3.211) All elephants are large animals.
Dumbo is a small elephant.
Is Dumbo a small animal? [No]

(3.212) All mice are small animals.
All elephants are large animals.
Mickey is a large mouse.
Dumbo is a small elephant.
Is Dumbo larger than Mickey? [Yes]
Assume comparative relations exemplified in (3.208) and (3.209)

(3.213) All mice are small animals.
Mickey is a large mouse.
Is Mickey small? [Yes for a mouse
?? No for an animal]
Adjectives requiring a comparison class cannot usually be predicated in the absence of a common noun, unless some comparison class is clear from the wider context.

3.5.5 Extensional and Intensional Comparison Classes

Some adjectives require an ‘intensional’ comparison class: different inferences may follow when two distinct but co-extensive predicates provide the comparison class.

(3.214) All legal authorities are law lecturers.
All law lecturers are legal authorities.
Are all fat legal authorities fat law lecturers? [Yes]
Extensional comparison class

(3.215) All legal authorities are law lecturers.
All law lecturers are legal authorities.
Are all competent legal authorities competent law lecturers? [Don’t know]
Intensional comparison class
(3.216) John is a fatter politician than Bill.
Is John fatter than Bill? [Yes]

Extensional

(3.217) John is a cleverer politician than Bill.
Is John cleverer than Bill? [Don’t know]

Intensional

Note that both intensional and extensional comparison class adjectives support comparatives.

3.5.6 Default Comparison Classes

Comparison class adjectives can sometimes pick up a default comparison class from the subject NP. For example, knowing that Kim is a person provides a default scale for assessing cleverness in people. If Kim were known to be a dog, the assessment scale would be different.

(3.218) Kim is a clever person.
Is Kim clever? [Yes]

(3.219) Kim is a clever politician.
Is Kim clever? [Don’t know]

3.6 Comparatives

3.6.1 Phrasal Comparatives

(3.220) The PC-6082 is faster than the ITEL-XZ.
The ITEL-XZ is fast.
Is the PC-6082 fast? [Yes]
(3.221) The PC-6082 is faster than the ITEL-XZ.  
Is the PC-6082 fast?  
[Don’t know]

(3.222) The PC-6082 is faster than the ITEL-XZ.  
The PC-6082 is fast.  
Is the ITEL-XZ fast?  
[Don’t know]

(3.223) The PC-6082 is faster than the ITEL-XZ.  
The PC-6082 is slow.  
Is the ITEL-XZ fast?  
[No]

(3.224) The PC-6082 is as fast as the ITEL-XZ.  
The ITEL-XZ is fast.  
Is the PC-6082 fast?  
[Yes]

(3.225) The PC-6082 is as fast as the ITEL-XZ.  
Is the PC-6082 fast?  
[Don’t know]

(3.226) The PC-6082 is as fast as the ITEL-XZ.  
The PC-6082 is fast.  
Is the ITEL-XZ fast?  
[Don’t know]

(3.227) The PC-6082 is as fast as the ITEL-XZ.  
The PC-6082 is slow.  
Is the ITEL-XZ fast?  
[No]

(3.228) The PC-6082 is as fast as the ITEL-XZ.  
Is the PC-6082 faster than the ITEL-XZ?  
[Don’t know]

(3.229) The PC-6082 is as fast as the ITEL-XZ.  
Is the PC-6082 slower than the ITEL-XZ?  
[No]
(3.230)  ITEL won more orders than APCOM did.
Did ITEL win some orders?  [Yes]

(3.231)  ITEL won more orders than APCOM did.
Did APCOM win some orders?  [Don’t know]

(3.232)  ITEL won more orders than APCOM did.
APCOM won ten orders.
Did ITEL win at least eleven orders?  [Yes]

Inferences (3.233)–(3.235) are similar to (3.230)–(3.232). Note however, that if “APCOM” can be interpreted as referring to a particular order (e.g. “the APCOM contract”), as it can in (3.233), the sentence ITEL won more orders than APCOM is ambiguous between a reading like that in (3.230)–(3.232), and one where ITEL won more than just the APCOM order — see (3.236)

(3.233)  ITEL won more orders than APCOM.
Did ITEL win some orders?  [Yes]

(3.234)  ITEL won more orders than APCOM.
Did APCOM win some orders?  [Don’t know]

(3.235)  ITEL won more orders than APCOM.
APCOM won ten orders.
Did ITEL win at least eleven orders?  [Yes]

(3.236)  ITEL won more orders than the APCOM contract.
Did ITEL win the APCOM contract?  [Yes]

(3.237)  ITEL won more orders than the APCOM contract.
Did ITEL win more than one order?  [Yes]
(3.238) ITEL won twice as many orders than APCOM. 
APCOM won ten orders
Did ITEL win twenty orders? [Yes]

3.6.2 Clausal Complement

(3.239) ITEL won more orders than APCOM lost. 
Did ITEL win some orders? [Yes]

(3.240) ITEL won more orders than APCOM lost. 
Did APCOM lose some orders? [Don’t know]

(3.241) ITEL won more orders than APCOM lost. 
APCOM lost ten orders. 
Did ITEL win at least eleven orders? [Yes]

3.6.3 Measure Phrases

(3.242) The PC-6082 is faster than 500 MIPS. 
The ITEL-ZX is slower than 500 MIPS
Is the PC-6082 faster than the ITEL-ZX? [Yes]

3.6.4 Differential Comparatives

(3.243) ITEL sold 3000 more computers than APCOM. 
APCOM sold exactly 2500 computers. 
Did ITEL sell 5500 computers? [Yes]
3.6.5 Attributive Comparatives

(3.244) APCOM has a more important customer than ITEL.
Does APCOM have a more important customer than ITEL is?
[Yes, on one reading of the premise]

(3.245) APCOM has a more important customer than ITEL.
Does APCOM has a more important customer than ITEL has?
[Yes, on one reading of the premise]

3.6.6 Comparatives and Quantifiers

(3.246) The PC-6082 is faster than every ITEL computer.
The ITEL-ZX is an ITEL computer.
Is the PC-6082 faster than the ITEL-ZX?
[Yes]

(3.247) The PC-6082 is faster than some ITEL computer.
The ITEL-ZX is an ITEL computer.
Is the PC-6082 faster than the ITEL-ZX?
[Don’t know]

(3.248) The PC-6082 is faster than any ITEL computer.
The ITEL-ZX is an ITEL computer.
Is the PC-6082 faster than the ITEL-ZX?
[Yes]

(3.249) The PC-6082 is faster than the ITEL-ZX and the ITEL-ZY.
Is the PC-6082 faster than the ITEL-ZX?
[Yes]

(3.250) The PC-6082 is faster than the ITEL-ZX or the ITEL-ZY.
Is the PC-6082 faster than the ITEL-ZX?
[Yes, on one reading of the premise]
3.7 Temporal Reference

Inference patterns involving temporal reference are complicated by the interplay between tense, aspectual information, lexical semantics, defeasible interpretation principles such as narrative progression, rhetorical relations, a theory of action and causation, world knowledge, interaction between plurality, genericity and temporal/aspectual phenomena etc. Some of the inferences are very basic, some are more involved. The more complex examples give ample illustration of the fact that temporal phenomena are usually discourse phenomena.

3.7.1 Standard Use of Tenses

(3.251) ITEL has a factory in Birmingham.

Does ITEL currently have a factory in Birmingham?

[Yes]

(3.252) Since 1992 ITEL has been in Birmingham.

It is now 1996.

Was ITEL in Birmingham in 1993?

[Yes]

(3.251) and (3.252) are instances of the subinterval property. This works only with stative verbs. C.f. the following example involving an accomplishment verb in the simple past:

(3.253) ITEL has developed a new editor since 1992.

It is now 1996

Did ITEL develop a new editor in 1993?

[Don’t know]

Similarly with activity verbs and adverbial modification

(3.254) ITEL has expanded since 1992.

It is now 1996

Did ITEL expand in 1993?

[Don’t know]

Also, the position of the “since” adverbial affects the range of readings available:

(3.255) Since 1992 ITEL has made a loss.

It is now 1996.

Did ITEL make a loss in 1993?

[Yes]
(3.256) ITEL has made a loss since 1992.

It is now 1996.

Did ITEL make a loss in 1993?

[Don’t know, on one reading of the premise]

(3.257) ITEL has made a loss since 1992.

It is now 1996.

Did ITEL make a loss in 1993?

[Yes, on one reading of the premise]

(3.258) In March 1993 APCOM founded ITEL.

Did ITEL exist in 1992?

[No]

(3.258) involves the lexical semantics of found.

3.7.2 Temporal Adverbials

3.7.2.1 Indexicals

Non-context dependent indexicals are reasonably straightforward:


It lasted 2 days.

Was the conference over on July 8th, 1994?

[Yes]

Context dependent indexicals (e.g. today, yesterday) are evaluated with respect to some temporal reference point (e.g. now):

(3.260) Yesterday APCOM signed the contract.

Today is Saturday, July 14th.

Did APCOM sign the contract Friday, 13th?

[Yes]

3.7.2.2 ‘Before’, ‘After’ (Temporal Subordinate Clauses)

Ignoring counterfactual readings, before and after have the following transitivity properties: if X, Y and Z are either all state or accomplishment or achievement or activity denoting sentences we have

107
\([3.261]\)
\[
\begin{align*}
X \triangleright Y. \\
Y \triangleright Z. \\
X \triangleright Z.
\end{align*}
\]
where \(\triangleright \in \{\text{before, after}\}

\((3.262)\)  
Smith left after Jones left.  
Jones left after Anderson left.  
\[
\text{Did Smith leave after Anderson left?} \\
\text{[Yes]}
\]

In general transitivity does not hold when we mix aspectual classes in the premises:

\((3.263)\)  
Smith was present after Jones left.  
Jones left after Anderson was present.  
\[
\text{Was Smith present after Anderson was present?} \\
\text{[Don’t know]}
\]

If \(X\) and \(Y\) are either all accomplishment or achievement denoting sentences with simple tenses \textit{before} and \textit{after} are inverses of each other:

\((3.264)\)  
\(X\) before \(Y\) iff \(Y\) after \(X\).

\((3.265)\)  
Smith left.  
Jones left.  
Smith left after Jones left.  
\[
\text{Did Jones leave before Smith left?} \\
\text{[Yes]}
\]

\((3.266)\)  
Smith left.  
Jones left.  
Jones left before Smith left.  
\[
\text{Did Smith leave after Jones left?} \\
\text{[Yes]}
\]

\((3.267)\)  
Jones revised the contract.  
Smith revised the contract.  
Jones revised the contract before Smith did.  
\[
\text{Did Smith revise the contract after Jones did?} \\
\text{[Yes]}
\]
(3.268) Jones revised the contract.
    Smith revised the contract.
    Jones revised the contract after Smith did.
    Did Smith revise the contract before Jones did?
    [Yes]

In general this is not so with activity verbs:

(3.269) Smith swam.
    Jones swam.
    Smith swam before Jones swam.
    Did Jones swim after Smith swam?
    [Don’t know]

However we do get

(3.270) Smith swam to the shore.
    Jones swam to the shore.
    Smith swam to the shore before Jones swam to the shore.
    Did Jones swim to the shore after Smith swam to the shore?
    [Yes]

Here the PP to the shore provides an end point or conclusion for the activity.

Before and after are not inverses for state denoting sentences:

(3.271) Smith was present.
    Jones was present.
    Smith was present after Jones was present.
    Was Jones present before Smith was present?
    [Don’t know]

(3.272) Smith was present.
    Jones was present.
    Smith was present before Jones was present.
    Was Jones present after Smith was present?
    [Don’t know]

(3.273) Smith was writing a report.
    Jones was writing a report.
    Smith was writing a report before Jones was writing a report.
    Was Jones writing a report after Smith was writing a report?
    [Don’t know]
(3.274) Smith was writing a report.
Jones was writing a report.
Smith was writing a report after Jones was writing a report.

Was Jones writing a report before Smith was writing a report?
[Don't know]

Also before, but not after, can have a counterfactual meaning. Whether this is a distinct sense of before is open to debate:

(3.275) Smith left the meeting before he lost his temper.
Did Smith lose his temper?
[Don’t know]

With when things are even more complicated. The problem is that it is often very difficult to tease apart the temporal from the causal dimension of when, c.f.

(3.276) When they opened the M25, traffic increased.

3.7.2.3 ‘In’, ‘For’ and ‘On’ Temporal Adverbials

In and for adverbials can be used as tests for the aspectual class of verb phrases (or sentences).

Did Smith live in Birmingham in 1992?
[Don’t know]

Stative

(3.278) Smith wrote his first novel in 1991.
Did Smith write his first novel in 1992?
[No]

(Unrepeatable) accomplishment

(3.279) Smith wrote a novel in 1991.
Did Smith write it in 1992?
[No]

(Unrepeatable) accomplishment

Did Smith write a novel in 1992?

[Don’t know]

(Repeatable) accomplishment

(3.281) Smith was running a business in 1991.

Was Smith running it in 1992?

[Don’t know]

Activity


Did Smith discover it in 1992?

[No]

(Unrepeatable) achievement


Did Smith discover a new species in 1992?

[Don’t know]

(Repeatable) achievement

(3.284) Smith wrote a report in two hours.

Smith started writing the report at 8 am.

Had Smith finished writing the report by 11 am?

[Yes]

Accomplishment

(3.285) Smith wrote a report in two hours.

Did Smith spend two hours writing the report?

[Don’t know]

Smith may have written the report in less than two hours. It is unclear whether there are two different readings for the premise: one where Smith takes exactly two hours, and one where he does it within two hours.

(3.286) Smith wrote a report in two hours.

Did Smith spend more than two hours writing the report?

[No]
(3.287) Smith wrote a report in two hours.

Did Smith write a report in one hour?
[Don’t know]

(3.288) Smith wrote a report in two hours.

Did Smith write a report?
[Yes]

(3.289) Smith discovered a new species in two hours.

Did Smith spend two hours discovering the new species?
[No]

Achievements are typically (more or less) instantaneous

(3.290) Smith discovered a new species in two hours.

Did Smith discover a new species?
[Yes]

(3.291) Smith discovered many new species in two hours.

Did Smith spend two hours discovering new species?
[?Yes]

Repeated achievement can last two hours

(3.292) Smith was running his own business in two years.

Did Smith spend two years running his own business?
[Don’t know]

Premise refers to time taken to inception of activity, not duration of activity.

(3.293) Smith was running his own business in two years.

Did Smith spend more than two years running his own business?
[Don’t know]

Cf similar inference for accomplishment, (3.286)
(3.294) Smith was running his own business in two years. Did Smith run his own business? [Yes]

(3.295) In two years Smith owned a chain of businesses. Did Smith own a chain of business for two years? [Don’t know]
States behave like activities.

(3.296) In two years Smith owned a chain of businesses. Did Smith own a chain of business for more than two years? [Don’t know]

(3.297) In two years Smith owned a chain of businesses. Did Smith own a chain of business? [Yes]

(3.298) Smith lived in Birmingham for two years. Did Smith live in Birmingham for a year? [Yes]
State

(3.299) Smith lived in Birmingham for two years. Did Smith live in Birmingham for exactly a year? [No]

(3.300) Smith lived in Birmingham for two years. Did Smith live in Birmingham? [Yes]

(3.301) Smith ran his own business for two years. Did Smith run his own business for a year? [Yes]
Activity

113
(3.302) Smith ran his own business for two years.  
Did Smith run his own business? [Yes]

(3.303) Smith wrote a report for two hours.  
Did Smith write a report for an hour? [Yes]

Accomplishment

(3.304) Smith wrote a report for two hours.  
Did Smith write a report? [Don’t know]

He may not have finished it

(3.305) #Smith discovered a new species for an hour.

(3.306) Smith discovered new species for two years.  
Did Smith discover new species? [Yes]

Repeated achievement

3.7.2.4 Quantificational Adverbials

(3.307) In 1994 ITEL sent a progress report every month.  
Did ITEL send a progress report in July 1994? [Yes]

Quantificational adverbials also introduce scope ambiguities with respect to other quantified NPs

(3.308) Smith wrote to a representative every week.  
Is there a representative that Smith wrote to every week? [Yes on one scoping; don’t know on another scoping]

3.7.3 Anaphoric Dimension

Rhetorical relations like narrative progression are defeasible interpretation principles. They depend on a theory of action and causation and general world knowledge (c.f. (3.309) and
(3.309) Smith left the house at a quarter past five.
   She took a taxi to the station
   and caught the first train to Luxembourg.

(3.310) Smith lost some files.
   They were destroyed when her hard disk crashed.

(3.311) Smith had left the house at a quarter past five.
   Then she took a taxi to the station.
   Did Smith leave the house before she took a taxi to the station?
   [Yes]

### 3.7.4 Adverbs of Quantification

(3.312) ITEIL always delivers reports late.
   In 1993 ITEIL delivered reports.
   Did ITEIL deliver reports late in 1993?
   [Yes]

(3.313) ITEIL never delivers reports late.
   In 1993 ITEIL delivered reports.
   Did ITEIL deliver reports late in 1993?
   [No]

### 3.7.5 Some more Complex Examples

   Today is the 15th of May, 1995.
   She is still in Paris.
   Was Smith in Paris on the 7th of May, 1995?
   [Yes]

(3.315) When Smith arrived in Katmandu
   she had been travelling for three days.
   Had Smith been travelling the day before she arrived in Katmandu?
   [Yes]
(3.316) Jones graduated in March and has been employed ever since. Jones has been unemployed in the past. 

Was Jones unemployed at some time before he graduated? [Yes]

(3.317) Every representative has read this report. No two representatives have read it at the same time. No representative took less than half a day to read the report. There are sixteen representatives. 

Did it take the representatives more than a week to read the report? [Yes]

(3.318) While Jones was updating the program, Mary came in and told him about the board meeting. She finished before he did. 

Did Mary’s story last as long as Jones’s updating the program? [No]

(3.319) Before APCOM bought its present office building, it had been paying mortgage interest on the previous one for 8 years. Since APCOM bought its present office building it has been paying mortgage interest on it for more than 10 years. 

Has APCOM been paying mortgage interest for a total of 15 years or more? [Yes]

(3.320) When Jones got his job at the CIA, he knew that he would never be allowed to write his memoirs. 

Is it the case that Jones is not and will never be allowed to write his memoirs? [Yes]

(3.321) Smith has been to Florence twice in the past. Smith will go to Florence twice in the coming year. Two years from now will Smith have been to Florence at least four times? [Yes]
(3.322) Last week I already knew that when, in a months time, Smith would discover that she had been duped she would be furious.

Will it be the case that in a few weeks Smith will discover that she has been duped; and will she be furious? [Yes]

(3.323) No one gambling seriously stops until he is broke.

No one can gamble when he is broke.

Does everyone who starts gambling seriously stop the moment he is broke? [Yes]

(3.324) No one who starts gambling seriously stops until he is broke.

Does everyone who starts gambling seriously continue until he is broke? [Yes]

(3.325) Nobody who is asleep ever knows that he is asleep.

But some people know that they have been asleep after they have been asleep.

Do some people discover that they have been asleep? [Yes]

3.8 Verbs

3.8.1 Aspectual Classes

See also the inference pertaining to in and for adverbials.

(3.326) ITEL built MTALK in 1993.

Did ITEL finish MTALK in 1993? [Yes]

(3.327) ITEL was building MTALK in 1993.

Did ITEL finish MTALK in 1993? [Don’t know]
(3.328)  ITEI won the contract from APCOM in 1993.
Did ITEI win a contract in 1993?  [Yes]

(3.329)  ITEI was winning the contract from APCOM in 1993.
Did ITEI win a contract in 1993?  [Don’t know]

Did ITEI own APCOM in 1990?  [Yes]

3.8.2 Distributive and Collective Predication

(3.331)  Smith and Jones left the meeting.
Did Smith leave the meeting  [Yes]

(3.332)  Smith and Jones left the meeting.
Did Jones leave the meeting  [Yes]

(3.333)  Smith, Anderson and Jones met.
Was there a group of people that met?  [Yes]

3.9 Attitudes

3.9.1 Epistemic, Intentional and Reportive Attitudes

(3.334)  Smith knew that ITEI had won the contract in 1992.
Did ITEI win the contract in 1992?  [Yes]

(3.335)  Smith believed / said / denied / feared / hoped that
ITEI had won the contract in 1992.
Did ITEI win the contract in 1992?  [Don’t know]
(3.336) **ITED managed to win the contract in 1992.**
**Did ITED win the contract in 1992?**
[Yes]

(3.337) **ITED tried/wanted to win the contract in 1992.**
**Did ITED win the contract in 1992?**
[Don’t know]

(3.338) **It is true that ITED won the contract in 1992.**
**Did ITED win the contract in 1992?**
[Yes]

(3.339) **It is false that ITED won the contract in 1992.**
**Did ITED win the contract in 1992?**
[No]

### 3.9.2 Preceptive Attitudes: “See” with Bare Infinitive Complements

#### 3.9.2.1 Inferences we do not get

(3.340) **Smith saw Jones sign the contract**
If Jones signed the contract, his heart was beating
**Did Smith see Jones’ heart beat?**
[Don’t know]

(3.341) **Smith saw Jones sign the contract**
When Jones signed the contract, his heart was beating
**Did Smith see Jones’ heart beat?**
[Don’t know]

#### 3.9.2.2 Veridicality

\[ a \text{ saw } \phi \rightarrow \phi \]

(3.342) **Smith saw Jones sign the contract**
**Did Jones sign the contract?**
[Yes]
3.9.2.3 Substitution

\( a \text{ saw } \phi(b), b = c \rightarrow a \text{ saw } \phi(c) \)

\(\begin{align*}
(3.343) & \quad \text{Smith saw Jones sign the contract} \\
& \quad \text{Jones is the chairman of ITEL} \\
& \quad \text{Did Smith see the chairman of ITEL sign the contract?} \\
& \quad [\text{Yes}] \\
\end{align*}\)

3.9.2.4 Existential instantiation

\( a \text{ saw } \phi(b) \rightarrow \exists x \ a \text{ saw } \phi(x) \)

\(\begin{align*}
(3.344) & \quad \text{Helen saw the chairman of the department answer the phone} \\
& \quad \text{The chairman of the department is a person} \\
& \quad \text{Is there anyone whom Helen saw answer the phone?} \\
& \quad [\text{Yes}] \\
\end{align*}\)

3.9.2.5 Conjunction distribution

\( a \text{ saw } \phi \land \psi \rightarrow a \text{ saw } \phi \text{ and } a \text{ saw } \psi \)

\(\begin{align*}
(3.345) & \quad \text{Smith saw Jones sign the contract and his secretary make a copy} \\
& \quad \text{Did Smith see Jones sign the contract?} \\
& \quad [\text{Yes}] \\
\end{align*}\)

3.9.2.6 Disjunction distribution

\( a \text{ saw } \phi \lor \psi \rightarrow a \text{ saw } \phi \text{ or } a \text{ saw } \psi \)

\(\begin{align*}
(3.346) & \quad \text{Smith saw Jones sign the contract or cross out the crucial clause} \\
& \quad \text{Did Smith either see Jones sign the contract or see Jones cross out the crucial clause?} \\
& \quad [\text{Yes}] \\
\end{align*}\)
Chapter 4

Computational Lexical Semantics in Review

This chapter was contributed by Ted Briscoe, University of Cambridge Computer Laboratory, ejb@cl.cam.ac.uk

4.1 Introduction

In this short overview, whose title I have shamelessly adapted from Beth Levin’s [1995] influential review of linguistic work on (verbal) lexical semantics, I try to rereview this and related work in the light of the last decade’s research and also to identify developments relevant to computational lexical semantics. In the final section, I summarise how the Acquilex (Esprit Basic Research) project has attempted to extend and exploit this work in the development of a prototype computational lexicon incorporating lexical semantic information which integrates with computational approaches to proof/model-theoretic compositional semantics (of the type explored in the FraCaS (CEC LRE) project).

The study of lexical semantics and of the lexicon has seen a resurgence of activity during the last decade. Hudson [1995] convincingly argues that linguistic theory provided little justification for the then prevalent mainstream assumptions that:

1. the lexicon is a distinct component of the grammar,

2. the lexicon is a discrete list of lexical entries, and

3. the lexicon contains only intralinguistic information.

Assumption (2) has been largely abandoned and replaced by a conception of the lexicon as an inheritance network incorporating at least is-a links (e.g. [Pollard and Sag, 1987])
and possibly default inheritance (e.g. [Evans and Gazdar, 1989b]). One important implication of this development for lexical semantics is the possibility of treating the polysemy/homonymy distinction more satisfactorily by treating polysemy in terms of such links rather than ignoring it and enumerating all word senses as a list of distinct lexical entries (e.g. [Pustejovsky and Boguraev, 1993]). Assumption (1) is the subject of much more focussed debate as a result of the work on network-based lexical knowledge representation languages (LKRLs); in particular, Gazdar & Evans propose that whilst syntax and compositional semantics can be expressed within a monotonic unification-based formalism, lexical processes and relations require a more expressive non-monotonic formalism such as their path equation LKRL, DATR; whilst, within HPSG, it has been proposed that the lexicon is represented as a set of typed feature structures related by subsumption, effectively reducing the lexicon to the syntactic/semantic formalism. However, the status of lexical rules within the HPSG framework has been contentious and problematic (e.g. [Riehmann, 1993; Calcagno, 1995]). Assumption (3) has received less attention, but nevertheless can be considered in a more focussed manner in the light of the advent of LKRLs. Both DATR and the HPSG lexicon are couched in formalisms which allow only a proper subset of the operations defined in general purpose KRLs which have been proposed within the AI literature. It remains to be seen whether lexical relations and processes can be adequately expressed within these more restricted formalisms.

These brief remarks, I hope make it clear that lexical (semantic) work has progressed somewhat beyond the situation which Hudson criticises and which he exemplifies with the GPSG ([Gazdar et al., 1985]) lexical entry for weep repeated below:

```
<weep,
 [ [-N, [+V], [BAR 0], [SUBCAT 1]],
  {wept},
  WEEP`
```

in which one sense and syntactic realisation is represented by a minimal description of orthography/phonology, syntactic (sub)category, irregular morphological variant form, and unanalysed semantic primitive, as part of a large list of unrelated other entries. In what follows I attempt to flesh out these developments in more detail. I will review lexical (semantic) research under the four subheadings of argument structure, event structure, qualia structure and inheritance structure. These are borrowed from Pustejovsky [1991; 1995], whose work has done much to rekindle and refocus work on lexical semantics. I don't attempt to be at all comprehensive either in coverage of active topics of research or in citing relevant work; rather, I attempt to identify the important ideas and give main references from which further work can be found.

### 4.1.1 Argument Structure

Work on argument structure, defined as a specification of the number, syntactic and semantic type, and syntactic realisation of a predicate's arguments, has been the focus of most gen-
ervative research on lexical semantics and particularly on the syntactico-semantic interface. Levin [1995] exclusively reviews such work which primarily addresses the related problems of linking (capturing generalisations concerning the syntactic realisation of different arguments) and of diathesis alternations (the same argument may be realised differently or omitted). For example, in John broke the vase the verb break appears with two arguments, an agent and a theme / affected object. In general, agents (if realised) are subjects and themes are objects, as here. However, as break can undergo the causative/inchoative alternation, the theme can be realised as subject The vase broke. These facts are seen as lying at the syntactico-semantic interface since they appear to be motivated by aspects of the lexical semantics of the predicate. For instance, break and eat both undergo (in)transitivities (John ate (chips)) but only break undergoes causative/inchoative because it is a unaccusative change-of-state verb. Frequently, alternations modify the semantic class or entailments of a predicate, for example slide is extended from a change-of-position to a change-of-possession verb when it undergoes the dative alternation – John slid the computer to the door / John slid *the door/Bill the computer. Such phenomena and treatments of it have been exhaustively surveyed in [Levin, 1993].

Attempts to express rules of linking and alternation have either been couched in terms of case/semantic/thematic roles (e.g. [Fillmore, 1968]) or in terms of predicate decomposition (e.g. [Carter, 1976]). One problem with both approaches from the perspective of computational lexical semantics is that usually neither approach is related to a proof-theoretic framework. One exception is the work of Dowty [1989] which treats thematic roles as bundles of entailments prototypically associated with predicates when the associated argument is realised. This captures both the fact that alternations typically affect meaning and that the entailments associated with a thematic role, such as agent, are not uniform across all predicates (e.g. volitionality The drunk collided with the lamppost / The drunk wanted a scotch). It also supports a prototypic semantic account of linking in which, for example, the argument associated with the greatest number of prototypically agentive entailments is realised as subject. Dowty modifies Davidson's event-based semantic framework to include proto-thematic roles as predicates relating event variables to participants denoted by arguments. The representation of John broke the vase would be roughly: \( \exists (e, x) \land \text{vase}(x) \land \text{break}(e) \land p_{agte}(e, j) \land p_{pat}(e, x) \).

4.1.2 Event Structure

Event structure refers to the (Vendlerian) classification of predicates into states, processes and transitions (achievements or accomplishments). Dowty [1979] provides a decompositional account of event structure in terms of the primitives BE, BECOME, CAUSE, and DO. He provides a model-theoretic interpretation for three of these primitives. His and further work has explored the syntactico-semantic consequences of these distinctions relating to aspect, postmodification, and diathesis alternation and refinements and extensions have been proposed (eg. [Moen and Steedman, 1988; Pustejovsky, 1995]). For example, as (1a,b) illustrates, durative PP postmodifiers can modify processes, but when used with transitions must be interpreted as modifying the resulting state. This can be captured in an event-based framework which posits subeventualities of running and being at home broadly of the type proposed by Dowty, as sketched in (1c).
(1) a John ran for an hour.
   b John ran home for an hour (at lunchtime).
   c \( \exists (e, e') \land run(e, j) \land become(e') \land at\_home(e', j) \land for(e', 1h) \)
   d John ran (to the store)

Similarly, (1d) illustrates that a process can be coerced into a transition by the addition of a PP argument denoting a bounded path. Since the addition of a PP is often treated as a diathesis alternation, this shows that there can be a close connection between the phenomena of argument structure and event structure. Pustejovsky & Busa [1995] push this further by arguing that subevent headedness in transitions, that is whether the process or resultant state is the head, determines which subevent argument structure is realised syntactically. Many transition predicates are inherently marked for left/right headedness but when this is unspecified the resultant ambiguity over syntactic realisation leads to the causative/inchoative alternation. Pursuing the implications of contextual effects on aspectual interpretation has led some to argue that there are no inherent lexically-encoded distinctions and that the appropriate interpretation is entirely the result of the interaction of tense/aspect marking, choice of construction, modifiers and extralinguistic context (e.g. [Zaenen, 1993]). Nevertheless, some type of subevent decomposition still seems needed in order to account for scopal effects with modifiers, and so forth.

4.1.3 Qualia Structure

Pustejovsky [1991; 1995] attempts to integrate most work on lexical semantics into the framework of generative lexicon theory. However, the most innovative component of this theory, extending work by Moravskik [1975], is his proposal concerning qualia structure as an alternative to purely relational or decompositional descriptions of word senses. There are four qualia, constitutive, the relation between an object and its constituent parts, formal, that which distinguishes it within a larger domain, telic, its purpose or function, and agentive, factors involved in its origin. For instance, an entry for book might specify that it is constituted of text, its formal properties are that it consists of bound pages, a CD-ROM or whatever, its purpose is to be read and it was created by writing.

Qualia structure serves several roles in the theory. However, perhaps the main advantage of this approach to the specification of lexical meaning is that it provides a natural mechanism for co-specification of related polysemic senses. Thus, in examples like he picked up and began to browse one of the books on the table the verb pick up selects the formal physical dimension and browse the constitutive (or possibly telic) dimension. The fact that the two predicates are readily conjoined suggests that an approach which enumerates distinct polysemic senses of this type as separate lexical entries will be inadequate.

Qualia structure in nominals is also utilised in accounts of adjectival modification (fast motorway, fast waltz) and logical metonymy (John began a book/cigarette). Verbal qualia structure is utilised as a means of representing subevents in transitions; for example, the formal role
of *break* is the resultant state of being broken, whilst the process of breaking which brings this about is treated as the agentive role. It is not clear to me that this extended use of qualia structure can have an identical semantics to the nominal cases, nor that it makes much sense to talk of the purpose or physical properties of eventualities in general. A more general question concerning the representation of word sense in terms of qualia is determining the semantic status of the various dimensions in a given utterance. Intuitively, it seems that one or more dimensions are asserted and the remainder presupposed in any given usage. However, this question has not been addressed thoroughly to date.

### 4.1.4 Inheritance Structure

Inheritance structure is defined in terms of is-a or subtyping relations between elements of lexical entries. For example, the lexical entry for *novel* will inherit elements from *book* but will specialise the constitutive dimension from text to narrative. Inheritance allows for generalisation and economy in the statement of the lexicon as well as capturing relations between entries. The precise form of inheritance allowed is intimately connected to the LRKL adopted: [Pustejovsky and Boguraev, 1993] assume a fully general language such as KL-ONE in which multiple conflicting default inheritance is possible; [Pustejovsky, 1995] utilises subsumption in typed feature structures, thus limiting himself to multiple orthogonal (non-conflicting) non-default inheritance, as in HPSG ([Pollard and Sag, 1987; Carpenter, 1992]); [Copestake and Briscoe, 1995] motivate the use of multiple default inheritance in typed default feature structures for lexical semantic representation; [Evans and Gazdar, 1989b; Evans and Gazdar, 1989a] utilise multiple default inheritance of path equations yielding a system with similar expressive power, though reentrancy between paths in a feature structure cannot be directly manipulated.

In general, systems which make use of simpler inheritance schemes need another formal operation, usually dubbed a lexical rule, which generates derived lexical entries from basic ones. Lexical rules have been used to generate inflectional, derivational, diathesis alternated, and sense extended variants of basic entries. However, in most cases alternative analyses have been proposed which utilise underspecified representations and/or inheritance to generate the required variants. Pustejovsky & Boguraev argue that a system with the expressive power of KL-ONE allows for guided inference to lexically related concepts; for example, from the agentive dimension of *prisoner* 'imprison' it is possible to follow links to 'escape' and so forth generating a space of related and potentially relevant concepts. This step – from constrained inference in terms of is-a/default links between linguistically significant syntactic and semantic information, in the sense that it directly affects syntagmatic processes, to open-ended inference in terms of concepts – represents the boundary between linguistic and real-world knowledge and inference. To the extent that clearly linguistic lexical processes can be dealt with using a constrained framework such a distinction acquires more justification.
4.2 Lexical Semantics in Acquilex

The Acquilex projects have in part been concerned with the development of a computationally-tractable theory of the lexicon focussing on lexical semantics. In what follows, I summarise relevant publications and give pointers for further reading. The papers I reference are mostly available electronically at http://www.cl.cam.ac.uk/Research/NL/acquilex/acqhome.html.

4.2.1 Typed Default Feature Structures

The LKRL we utilise is an extension of the HPSG scheme to Typed Default Feature Structures (TDFs). TDFs are 'double' typed feature structures in which the second defeasible feature structure must be subsumed by the first indefeasible one (that is it must be strictly more specific in terms of the type system). An associative order-independent operation of typed default unification is defined over TDFs. This operation can be used to implement default inheritance in a type hiearchy and also more generally allow any two TDFs with compatible types to be unified. Specificity of defaults is defined in terms of the type hierarchy, so a default value associated with a more specific (sub) feature structure overrides the default value associated with the less specific one. This requires that TDFs record the type of the (sub) feature structure which introduced a default value into a given TDF in order to avoid ordering effects from a series of unifications. Nixon diamonds cannot arise in the default inheritance system based on the type hierarchy as only orthogonal inheritance is allowed. However, unification of two conflicting default values from TDFs of the same type will result in the least specific default value (i.e. $\top$); that is, the system is sceptical.

The underlying logic and formal details of TDFs and the associated operation of typed default unification is given in [Lascarides et al., 1994] and in [Lascarides and Copestake, 1995]. Here we give some informal examples and motivation for these extensions to the formal framework proposed by [Carpenter, 1992]. TDFs can be abbreviated by collapsing identical parts of the double feature structures and marking the distinct default components by a slash, as in Figure 4.2.1 in which the reentrancy between path $F:G$ and $H$ is default as is the path consisting of the attribute $H$. We ignore details in what follows since they are identical to the default component of the TDFS in the initial description.

Default inheritance in the TDFS system can be used to capture the fact that many lexical hyponymy relations are not absolute; for example, the telic role of a book is 'read' and novel,
monograph and dictionary might all be defined as subtypes of book. However, in the case of dictionary the telic role will need to be overridden to 'refer-to'. Furthermore, as we shall see below there are reasons for thinking that the specification of telic roles as well as other aspects of lexical semantic specification must remain default 'beyond the lexicon'. The approach to defaults exemplified by the TDFS scheme, in contrast to most other approaches to default unification, allows exactly this by explicitly marking default specifications and ensuring that this marking persists through the default unification operation.

In addition to inheritance via subtyping, the TDFS lexical framework incorporates lexical rules which generate derived lexical entries from basic lexical entries represented as TDFSs. Most lexical rules are specified as type-to-type mappings without further constraints on the content of the input and output TDFSs beyond those specified in the relevant type declarations. In order to constrain the application of lexical rules to account for exceptions but allow for novel as well as conventionalised application (that is, deal with semi-productivity), a probabilistic interpretation is placed on TDFSs which means that lexical rules may generate 'neological' unattested entries with very low probabilities associated with them. Further details are provided in [Briscoe and Copestake, 1995] where the approach is exemplified with reference to verbal diathesis alternations.

4.2.2 Systematic Polysemy

Any approach to lexical semantics which wishes to break away from the inadequate and unrevealing sense enumeration paradigm ([Pustejovsky, 1995, 39f]) needs to distinguish systematic polysemy from idiosyncratic polysemy and homonymy. For example, the polysemy of book between the physical object and what it represents seems to be common to all words for representational artifacts not only including types of book but also films, pictures, and so forth. It is this type of lexical semantic ambiguity which a more adequate account of the lexicon should express generatively. [Copestake and Briscoe, 1995] and [Copestake, 1995] argue that systematic polysemy can be a result of constructional polysemy or semi-productive sense extension elaborating [Cruse, 1986] and others’ informal distinction between sense modification and sense change.

4.2.2.1 Constructional Polysemy

Constructional polysemy is a consequence of sense modification in context where modification can be a consequence of further specification of an underspecified lexical entry, broadening of a specific entry with a default component, or coercion of quale to asserted status through semantic selection by predicates. (2a) represents a case of underspecification where the telic role of reel is ‘(un)wind’ but the nature of the material to be (un)wound is unspecified and acquired from context.
(2) a The film/fishing/cotton reel broke.
   b The cloud (of flies) was dense.
   c John enjoyed the book.

(2b) represents a case of broadening where cloud has a default constituency of water vapour, but this can be overridden by an of complement. (2c) is a case of type coercion of book by the predicate enjoy, which selects for an eventive complement (and, by default, the one specified by the telic role — in this case ‘reading’). The availability of both senses of book in the case of type coercion is illustrated by the possibility of what [Pustejovsky, 1995] terms co-predication — John picked up and finished his beer — supporting a treatment in which both senses are accessible from a single lexical entry. However, our current account does not deal with cases of coercion which involve non-constituent coordination, such as John enjoys films and mending antique clocks as a consequence of our decision to treat the coercion as internal to enjoy in order to deal with co-predication.

We argue that in these cases there is no question of semi-productivity or semi-regularity. Rather, if the appropriate linguistic context occurs, the sense is modified in entirely predictable ways. We provide a fully formal account of each of these processes in the TDFS framework which serve to flesh out and formalise a number of mechanisms and phenomena which [Pustejovsky, 1995] discusses under the rubrics of co-composition, co-predication and type coercion. The regularity of the polysemy in type coercion of course only extends as far as the eventive interpretation of the bare NP. The precise event is open to further contextual specification as in Judging by the number of pages missing, my pet goat obviously enjoyed that book. I return to this below.

4.2.2.2 Semi-Regular Sense Extension

Semi-regular sense extension can be distinguished from constructional polysemy on the basis that it results in generation of a variant lexical entry by lexical rule and such rules are not fully productive. For example, the various conventionalised subcases of nominal ‘grinding’ which yield mass nouns denoting a substance originating from the count noun sense, such as wood-grinding (oak tree, oak table), and food-grinding (a haddock, haddock fillet are quite regular within the appropriate lexical classes. However, there are exceptions caused by blocking (cow/beef, pig/pork), that is, pre-emption by synonymy. Furthermore, the general process of grinding, though applicable in the limit to any count noun denoting an individuated entity has very variable acceptability/conventionality, and conventionalised subcases differ cross-linguistically, suggesting the need for a linguistic overlaying of an ultimately more general conceptual transfer. An indication that grinding involves sense change rather than sense modification comes from the relative unacceptability of co-predication as compared to the book/contents polysemy discussed above: ?Sam fed and carved his lamb, Sam picked up and enjoyed that novel.

Similar arguments can be made concerning the treatment of diathesis alternations; these also involve fairly regular sense extensions but are subject to lexical gaps which are not entirely
predictable by restricting the application of lexical rules to semantic classes, even if these classes are narrowly specified (e.g. [Pinker, 1989]). [Briscoe and Copestake, 1995] show that verbs which undergo the dative alternation cannot be accurately specified in terms of narrow semantic classes, though a broad class specification is certainly necessary to restrict application of the rule. Nor does it seem likely that exceptions can be accounted for entirely in terms of blocking. Instead we propose that ultimately, acceptability is judged in terms of attested frequency and that low frequency lexical entries are utilised only in neologisms, for rhetorical effect or whatever.

The account of lexical rules in the TDFS framework captures semi-productivity by incorporating conditional probabilities that word forms will be associated with lexical entries into the control component of lexical rule application and providing a means for computing the overall probability of an interpretation. Probabilities are not a component of the TDFS LKRL but are more appropriately seen as part of the performance component of a language processing system. Thus, the acceptability of *He designed Bill a new house* as opposed to *He created Bill a new house* in American English is related simply to the non-occurrence of *create* in this construction in the experience of American English speakers and not to blocking or semantic differences between the two predicates. On the other hand, neological use of *create* in this construction is interpretable because the lexical rule of dative can apply to *create* even though the resultant lexical entry is marked as unattested and of consequent low probability.

Blocking of sense extensions is not absolute but rather carries additional implicatures when it is over ridden, as in *The drunken revellers were all too eager to tear into the roast cow*. In this sense blocking is a barrier to the establishment or institutionalisation of a sense rather than to its use. [Briscoe et al., 1995] provide an account of blocking which captures the pragmatic markedness of unblocking. The probabilistic account to some extent subsumes this within a more computationally tractable framework, but fails to deal with the implicatures generated by unblocking.

### 4.2.3 Semantic/Pragmatic Interactions

The framework of lexical semantic description we have developed is intended to be compatible with extant approaches to (computational) compositional semantics. Lexical semantic representations are couched in an event-based version of a default conditional variant of the predicate calculus, which is itself described using TDFSs. [Lascarides, 1995] and [Lascarides et al., 1996] following [Frank and Reyle, 1995] interpret these TDFS descriptions within an extended version of discourse representation theory integrated with an approach to discourse interpretation based on a conditional default logic (SDRT/DICE, [Lascarides and Asher, 1993]). Thus, the process of compositional semantic interpretation can be implemented using successive applications of typed default unification, and this allows us to formalise underspecification, broadening, coercion, co-composition and co-predication as an interaction between lexical semantic specification and compositional semantic interpretation.

The semantic part of a TDFS may contain default specifications which have persisted from the lexicon through compositional semantics. For example, *John enjoyed the book* will have a
logical representation like $\exists (e, e', x, y) \land john(x) \land enjoy(e, x, e') \land book(y) \land P(e', x, y) \land \neg read(e', x, y)$ where $P$ is a predicate variable over eventuality and $\neg read$ states that this is a reading event by default. For an example like *My goat enjoyed that book* this lexically-specified default is pragmatically inappropriate, so the interface to discourse interpretation must allow for such defaults to be overridden pragmatically. [Lascarides, 1995] gives a DRT-based semantics for the asterisk notation and shows how it can be integrated into the DICE framework in terms of two general principles: Defaults Survive and Discourse Wins. These two principles ensure that in the absence of conflicting discourse/pragmatic information lexically-specified defaults survive in the logical representation of the meaning of the input produced during parsing. However, in the presence of such conflicting information, for example that goats don’t read, the discourse component both overrides the lexically-specified default and can supply a more appropriate refinement of the event variable in terms of more specific pragmatic information, for example, that goats eat indiscriminately. [Briscoe et al., 1990] provide some empirical evidence in support of this account of the lexical semantic / discourse processing interface, by demonstrating that logical metonymy is utilised only a) where the lexically-specified default interpretation is correct but the context may be uninformative or b) where the local context clearly overrides the lexically-specified default. Otherwise a non-metonymic complement is utilised. Less modular accounts of this interface (e.g. [Hobbs et al., 1990]) cannot explain the subset of a) cases in which the context is neutral other than by incorporating a lexical default into the pragmatic component. Evidence that these really are lexical defaults relating to word senses rather than concepts per se comes from examples like *John enjoyed the doorstop* where although *doorstop* denotes a book, the telic role for the this lexical item does not provide a coherent refinement of the metonymic event leading to oddity.

The approach reported in [Lascarides et al., 1996] extends this account of the lexical semantic / discourse processing interface to account for coherence effects in co-predication and to provide a more adequate account of zeugma. The absence of crossed readings in examples like *John has a file and so does Bill* and the zeugmatic effects in ones like *John banked the money and then the plane* are usually explained in terms of the ‘inaccessibility’ of multiple lexical entries (senses) in coordination constructions. However, examples like *This chicken is corn-fed and delicious or This thesis has thousands of pages and is unreadable* which are not zeugmatic but involve polysemic senses suggest that this account is incomplete at best. In our framework the related senses of *thesis* are a consequence of constructional polysemy and are represented within one entry, but the senses of *chicken* result from the lexical rule of meat-grinding, suggesting that the former but not the latter should be zeugmatic given our theory combined with the standard account. However, it is also possible to construct zeugmatic examples using the same senses of *thesis* and *chicken* – *This chicken is beheaded and delicious, This thesis is excellent and on fire* – suggesting that the standard account is not capturing the relevant constraint. The causal relevance of the two conjuncts is critical to the discourse coherence of these coordinations. DICE captures these conditions in terms of conditions on the inference of the discourse relation parallel which is triggered by clausal coordination. It is shown that the interaction of polysemic senses with these conditions is what blocks crossed readings and causes zeugma.
4.3 Conclusions

The approach that we have developed to lexical semantics and the organisation of the lexicon generally contra the proposals of [Hudson, 1995] and of most computational systems (e.g. [Hobbs et al., 1990]) is modular in that we argue for a restricted LKRL with well-defined and restricted interfaces to syntax, compositional semantics and pragmatics. This more modular account offers several theoretical and computational advantages, resulting in a stronger, more falsifiable theory which can be couched in a computationally-tractable formalism. Further research will be needed to decide whether the framework is empirically adequate. A number of immediate gaps and inadequacies can be identified: the interface between TDFS and DICE has not been implemented, the account of (un)blocking requires further work, and some types of non-constituent coordination involving type coercion remain outside the capabilities of the current proposals.
Chapter 5

Evaluation of the Framework: further implementation and extension

A draft of this chapter is being circulated separately to the project reviewers. It is the intention to update the draft in light of discussion at the review meeting in February to complete the final version of this deliverable.
Bibliography


[Cooper et al., 1993] Cooper, Robin; Lewin, Ian; and Black, Alan 1993. Prolog and natural language semantics. Lecture notes for computational semantics, Dept. of Artificial Intelligence, Univ. of Edinburgh.


134


