Industrial Parsing of Software Manuals

Editors:

Richard F. E. Sutcliffe
University of Limerick

Heinz-Detlev Koch
University of Heidelberg

Annette McElligott
University of Limerick
Dedicated to

Dr. A. Daly Briscoe
## Contents

1. Industrial Parsing of Software Manuals: an Introduction 1  
   1.1 Introduction ............................................. 1  
   1.2 IPSM Test Corpus ................................. 4  
       1.2.1 Why Software Manuals? ......................... 4  
       1.2.2 The 600 Utterance Corpus ...................... 5  
       1.2.3 The 60 Utterance Subset ....................... 5  
   1.3 Analysis of Parser Performance .................. 6  
       1.3.1 Three Phases of Analysis ....................... 6  
       1.3.2 Analysis of Particular Constructs .............. 6  
       1.3.3 Coverage ....................................... 7  
       1.3.4 Efficiency .................................... 7  
       1.3.5 Accuracy of Analysis .......................... 8  
   1.4 Structure of the Book .............................. 8  
       1.4.1 Introductory Chapters .......................... 8  
       1.4.2 Parsing Chapters ............................... 9  
       1.4.3 Appendices .................................... 10  
   1.5 Discussion ........................................... 10  
   1.6 Acknowledgements .................................. 11  
   1.7 References ........................................ 11  

2. Dependency-Based Parser Evaluation: a Study with a Software Manual Corpus 13  
   2.1 Introduction .......................................... 13  
   2.2 Dependency-Based Evaluation ...................... 15  
   2.3 Manual Normalization of Parser Outputs .......... 17  
   2.4 Automated Transformation from Constituency to Dependency .......................... 19  
   2.5 Conclusion ......................................... 22  
   2.6 References .......................................... 22  

3. Comparative Evaluation of Grammatical Annotation Models 25  
   3.1 Introduction .......................................... 25  
   3.2 Diversity in Grammars .............................. 26
5.3.2 Remarks on the Present Evaluation ................. 63
5.3.3 Current Evaluation Setting ........................ 64
5.4 Analysis I: Original Grammar, Original Vocabulary .... 65
  5.4.1 Observations about Morphological Analysis and Disambiguation ......................... 65
  5.4.2 Observations about Syntax ...................... 70
5.5 Analysis II: Original Grammar, Additional Vocabulary .. 72
  5.5.1 Observations about Morphological Disambiguation 72
5.6 Analysis III: Altered Grammar, Additional Vocabulary ... 77
  5.6.1 Observations about Morphological Disambiguation 77
  5.6.2 Observations about Syntax ...................... 80
5.7 Converting Parse Tree to Dependency Notation .... 83
5.8 Summary of Findings ................................. 85
5.9 References ........................................... 86

6. Using the Link Parser of Sleator and Temperly to Analyse a Software Manual Corpus 89
  6.1 Introduction ........................................ 89
  6.2 Description of Parsing System ....................... 90
  6.3 Parser Evaluation Criteria .......................... 92
  6.4 Analysis I: Original Grammar, Original Vocabulary .. 94
    6.4.1 Pre-Processing .................................. 94
    6.4.2 Results ........................................ 95
  6.5 Analysis II: Original Grammar, Additional Vocabulary .. 97
  6.6 Analysis III: Altered Grammar, Additional Vocabulary .. 99
  6.7 Converting Parse Tree to Dependency Notation .... 99
  6.8 Summary of Findings ................................ 101
  6.9 References ........................................ 102

7. Using PRINCIPAR to Analyse a Software Manual Corpus 103
  7.1 Introduction ........................................ 103
  7.2 Description of Parsing System ....................... 104
    7.2.1 Parsing by Message Passing .................... 104
    7.2.2 Implementation ............................... 109
  7.3 Parser Evaluation Criteria .......................... 110
  7.4 Analysis I: Original Grammar, Original Vocabulary .. 112
    7.4.1 Setting-Up the Experiment ..................... 112
    7.4.2 Results ........................................ 112
    7.4.3 Causes of Errors ................................ 114
  7.5 Analysis II: Original Grammar, Additional Vocabulary .. 115
  7.6 Analysis III: Altered Grammar, Additional Vocabulary .. 116
  7.7 Converting Parse Tree to Dependency Notation .... 116
  7.8 Summary of Findings ................................ 116
8. Using the Robust Alvey Natural Language Toolkit to Analyse a Software Manual Corpus
8.1 Introduction ............................................. 119
8.2 Description of Parsing System .......................... 121
  8.2.1 The Basic ANLT ................................... 121
  8.2.2 The Robust ANLT ................................. 124
8.3 Parser Evaluation Criteria .............................. 127
8.4 Analysis I: Original Grammar, Original Vocabulary .... 128
  8.4.1 Pre-Processing ................................... 128
  8.4.2 Results ........................................... 130
8.5 Analysis II: Original Grammar, Additional Vocabulary ... 130
8.6 Analysis III: Altered Grammar, Additional Vocabulary 130
8.7 Converting Parse Tree to Dependency Notation .......... 131
8.8 Summary of Findings .................................. 132
8.9 References ............................................. 135

9. Using the SEXTANT Low-Level Parser to Analyse a Software Manual Corpus
9.1 Introduction ............................................. 139
9.2 Description of Parsing System .......................... 140
  9.2.1 Preparing Processing ............................... 141
  9.2.2 Parsing ............................................. 145
  9.2.3 List Recognition ................................... 146
9.3 Parser Evaluation Criteria .............................. 149
9.4 Analysis I: Original Grammar, Original Vocabulary .... 150
9.5 Analysis II: Original Grammar, Additional Vocabulary .. 150
9.6 Analysis III: Altered Grammar, Additional Vocabulary 150
9.7 Converting Parse Tree to Dependency Notation .......... 150
9.8 Summary of Findings .................................. 150
9.9 References ............................................. 150

10. Using a Dependency Structure Parser without any Grammar Formalism to Analyse a Software Manual Corpus
10.1 Introduction ............................................ 159
10.2 Description of Parsing System .......................... 160
10.3 Parser Evaluation Criteria .............................. 160
10.4 Analysis I: Original Grammar, Original Vocabulary .... 160
10.5 Analysis II: Original Grammar, Additional Vocabulary .. 160
10.6 Analysis III: Altered Grammar, Additional Vocabulary 160
10.7 Converting Parse Tree to Dependency Notation .......... 160
10.8 Summary of Findings .................................. 160
11. Using the TOSCA Analysis System to Analyse a Software Manual Corpus 179
   11.1 Introduction ........................................... 179
   11.2 Description of Parsing System ....................... 181
       11.2.1 The TOSCA Analysis Environment .......... 181
       11.2.2 The Tagger ..................................... 183
       11.2.3 The Parser ..................................... 184
   11.3 Parser Evaluation Criteria .......................... 189
   11.4 Analysis I: Original Grammar, Original Vocabulary .. 190
       11.4.1 Efficacy of the Parser ....................... 193
       11.4.2 Efficiency of the Parser .................... 199
       11.4.3 Results ...................................... 199
   11.5 Analysis II: Original Grammar, Additional Vocabulary .. 201
   11.6 Analysis III: Altered Grammar, Additional Vocabulary .. 201
   11.7 Converting Parse Tree to Dependency Notation ...... 202
   11.8 Summary of Findings ................................ 205
   11.9 References .......................................... 206

Appendix I. 60 IPSM Test Utterances 207
Appendix II. Sample Parser Outputs 213
Appendix III. Collated References 259
Index 269
Industrial Parsing of Software Manuals:
an Introduction

Richard F. E. Sutcliffe\textsuperscript{1}
Annette McElligott
Heinz-Detlev Koch

University of Heidelberg
University of Limerick

1.1 Introduction

Parsing is the grammatical analysis of text. A parser is a computer
program which can carry out this analysis automatically on an input
provided in machine readable form. For example, given an utterance
such as

L22: Select the text you want to protect.

a parser might produce the following output:

\[ s, \[ vp, \[ v, select], \]
\[ np, \[ det, the], \]
\[ n, text], \]
\[ rc, \[ pro, you], \]
\[ vp, \[ v, want], \]
\[ ic, \[ to, to], \]
\[ v, protect]]]]]]]]]]]
This can be interpreted as saying that the input was a sentence comprising a single verb phrase, that the verb phrase consisted of the verb ‘select’ followed by a noun phrase, that the noun phrase comprised a determiner ‘the’, a noun ‘text’ and a relative clause, and so on. To understand the output it is necessary to know what each non terminal (‘s’, ‘vp’ etc.) means, and in precisely what kinds of structure each can occur within the output. This in turn requires an understanding of the linguistic formalism on which the parser is based.

The task of language engineering is to develop the technology for building computer systems which can perform useful linguistic tasks such as machine assisted translation, text retrieval, message classification and document summarisation. Such systems often require the use of a parser which can extract specific types of grammatical data from pre-defined classes of input text.

There are many parsers already available for use in language engineering systems. However, each uses a different linguistic formalism and parsing algorithm, and as a result is likely to produce output which is different from that of other parsers. To make matters worse, each is likely to have a different grammatical coverage and to have been evaluated using different criteria on different test data. To appreciate the point, study Appendix II where you will find eight analyses of the utterance L22. None of these bears any resemblance to the one shown above.

Suppose you wish to build a language engineering system which requires a parser. You know what syntactic characteristics you want to extract from an utterance but you are not interested in parsing per se. Which parsing algorithm should you use? Is there an existing parser which could be adapted to the task? How difficult will it be to convert the output of a given parser to the form which you require? What kind of coverage and accuracy can you expect? This book sets out to provide some initial answers to these questions, taking as its starting point one text domain, that of software instruction manuals.

The book is derived from a workshop Industrial Parsing of Software Manuals (IPSM'95) which was held at the University of Limerick, Ireland, in May 1995. Research teams around the world were invited to participate by measuring the performance of their parsers on a set of 600 test sentences provided by the organisers. The criteria to be used for measuring performance were also specified in advance. Eight groups from seven countries took up the challenge. At the workshop, participants described their parsing systems, presented their results and
outlined the methods used to obtain them.

One finding of IPSM'95 was that the articles produced for the proceedings (Sutcliffe, McElligott & Koch, 1995) were rather disparate, making direct comparison between systems difficult. Each group had conducted a slightly different form of analysis and the results were reported using tables and figures in a variety of configurations and formats. To take the aims of the workshop further, and to make the information available to a wider audience, each group was asked to carry out a more tightly specified analysis by applying their parser to a subset of the original IPSM corpus and presenting their findings in a standard fashion. The results of this second phase of work are contained in the present volume.

Another issue which developed out of the workshop relates to standardisation of parse trees. Each parser used in IPSM'95 produces a different type of output. This makes direct comparisons of performance difficult. Moreover, it is an impediment to structured language engineering, as we have already noted. In an ideal situation it would be possible to link existing tools such as lexical analysers, part-of-speech taggers, parsers, semantic case frame extractors and so on in various ways to build different systems. This implies both that the tools can be physically linked and that output data produced by each one can be made into a suitable input for the next component in the chain. Physical linkage is difficult in itself but has been addressed by such paradigms as GATE (Cunningham, Wilks & Gaizauskas, 1996). What can be done about the widely differing outputs produced by parsers?

Dekang Lin has on a previous occasion suggested that any parse can be converted at least partially into a dependency notation and that this form could comprise a standard by which the output from different systems could be compared (Lin, 1995). The idea was discussed in detail at the workshop and in consequence each group was requested to investigate the extent to which a dependency system could capture the data produced by their parser.

In the remainder of this introduction we describe in more detail the objectives and background of the IPSM project. In Section 1.2 we justify the use of computer manual texts as the basis of the study, describe the characteristics of the test data which was used, and explain exactly how it was produced. Section 1.3 outlines the three phases of analysis which were carried out on each parser, the kinds of information which were determined for each phase, and the means by which this was presented in tabular form. Section 1.4 describes the structure of the book and in particular explains the set of standard sections which are used for all the parsing chapters. Finally, Section 1.5 briefly discusses the findings of the project as a whole.
<table>
<thead>
<tr>
<th>Type</th>
<th>Dynix Count</th>
<th>Selected</th>
<th>Lotus Count</th>
<th>Selected</th>
<th>Trados Count</th>
<th>Selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>117</td>
<td>12</td>
<td>091</td>
<td>09</td>
<td>135</td>
<td>14</td>
</tr>
<tr>
<td>IMP</td>
<td>032</td>
<td>03</td>
<td>068</td>
<td>07</td>
<td>041</td>
<td>04</td>
</tr>
<tr>
<td>IVP</td>
<td>001</td>
<td>00</td>
<td>018</td>
<td>02</td>
<td>000</td>
<td>00</td>
</tr>
<tr>
<td>3PS</td>
<td>006</td>
<td>01</td>
<td>005</td>
<td>01</td>
<td>000</td>
<td>00</td>
</tr>
<tr>
<td>PVP</td>
<td>004</td>
<td>04</td>
<td>013</td>
<td>01</td>
<td>010</td>
<td>01</td>
</tr>
<tr>
<td>NP</td>
<td>040</td>
<td>04</td>
<td>005</td>
<td>00</td>
<td>012</td>
<td>01</td>
</tr>
<tr>
<td>QN</td>
<td>000</td>
<td>00</td>
<td>000</td>
<td>00</td>
<td>002</td>
<td>00</td>
</tr>
<tr>
<td>Total</td>
<td>200</td>
<td>20</td>
<td>200</td>
<td>20</td>
<td>200</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 1.1: IPSM Corpus broken down by utterance type and source document. Each column marked 'Count' shows the number of utterances of the given type which occurred in software manual shown in the first row. Each column marked ‘Selected’ shows the number of these which were used in the reduced set of 60 utterances. Examples of the various utterance types are shown in Table 1.2.

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Typically, there are multiple search menus on your system, each of which is set up differently.</td>
</tr>
<tr>
<td>IMP</td>
<td>Move the mouse pointer until the I-beam is at the beginning of the text you want to select.</td>
</tr>
<tr>
<td>IVP</td>
<td>To move or copy text between documents</td>
</tr>
<tr>
<td>3PS</td>
<td>Displays the records that have a specific word or words in the TITLE, CONTENTS, SUBJECT, or SERIES fields of the BIB record, depending on which fields have been included in each index.</td>
</tr>
<tr>
<td>PVP</td>
<td>Modifying the Appearance of Text</td>
</tr>
<tr>
<td>NP</td>
<td>Automatic Substitution of Interchangeable Elements</td>
</tr>
<tr>
<td>QN</td>
<td>What do we mean by this?</td>
</tr>
</tbody>
</table>

Table 1.2: Examples of the utterance types used in Table 1.1.

1.2 IPSM Test Corpus

1.2.1 Why Software Manuals?

Many studies on parsing in the past have been carried out using test material which is of little practical interest. We wished to avoid this by selecting a class of documents in which there is a demonstrated commercial interest. Software instruction manuals are of crucial importance to the computer industry generally and there are at least two good rea-
In tro duction

/5

sons for wishing to parse them automatically. The first is in order to translate them into different languages. Document translation is a major part of the software localisation process, by which versions of a software product are produced for different language markets. The second reason is in order to create intelligent on-line help systems based on written documentation. SIFT (Hyland, Koch, Sutcliffe and Vossen, 1996) is just one of many projects investigating techniques for building such systems automatically.

1.2.2 The 600 Utterance Corpus

Having decided on software documentation, three instruction manuals were chosen for use in IPSM. These were the *Dynix Automated Library Systems Searching Manual* (Dynix, 1991), the *Lotus Ami Pro for Windows User’s Guide Release Three* (Lotus, 1992) and the *Trados Translator’s Workbench for Windows User’s Guide* (Trados, 1995). A study had already been carried out on Chapter 5 of the Lotus manual, which contained 206 utterances. For this reason it was decided to use 200 utterances from each of the three manuals, making a total of 600. This corpus was then used for the initial analysis carried out by the eight teams and reported on at the IPSM workshop.

1.2.3 The 60 Utterance Subset

Following the workshop, we wished to carry out a more detailed and constrained study on the eight parsers in order to allow a more precise comparison between them. Unfortunately it was not feasible for all the teams to undertake a detailed study on the entire 600 utterance corpus. For this reason a 60 utterance subset was created. The following method was used to achieve this:

1. Each utterance in the original set of 600 was categorised by type using the classes Sentence (S), Imperative (IMP), Infinitive Verb Phrase (IVP), Third Person Singular (3PS), Progressive Verb Phrase (PVP), Noun Phrase (NP) and Question (QN). The analysis is shown in Table 1.1 with examples of each type shown in Table 1.2.

2. A selection was made from each manual for each utterance type such that the proportion of that type in the 60 utterance subset was as close as possible to that in the original 600 utterance corpus.

---

3We use the term *utterance* to mean a sequence of words separated from other such sequences, which it is desired to analyse. Some such utterances are sentences. Others (for example headings) may comprise a single verb phrase (e.g. ‘Proofing a Document’), a noun phrase (e.g. ‘Examples of Spell Check’) or some other construct.
1.3 Analysis of Parser Performance

1.3.1 Three Phases of Analysis

Each participant was asked to carry out a study to determine how well their parser was able to extract certain categories of syntactic information from the set of 60 test utterances. Three phases of analysis were requested. In Analysis I, the parser had to be used with its original grammar and lexicon. It was permissible however to alter the lexical analysis component of the system. For Analysis II, the lexicon of the parser could be augmented but no changes to the underlying grammar were allowed. Finally, Analysis III allowed changes to both the lexicon and the grammar.

The purpose of the three phases was to gain insight into how robust the different systems were and to provide lower and upper bounds for their performance in the task domain. Because of the diversity of parsing methods being used, the criteria for each phase had to be interpreted slightly differently for each system. Such differences are discussed in the text where they arise.

Participants were requested to provide their results in the form of a series of standard tables. The precise analysis which was carried out together with the format of the tables used to present the data are described in the following sections.

1.3.2 Analysis of Particular Constructs

The first piece of information provided for each parser is a table showing which forms of syntactic analysis it could in principle carry out. These forms are explained below using the example utterance ‘If you press BACKSPACE, Ami Pro deletes the selected text and one character to the left of the selected text.’.

- **A: Verbs recognised**: e.g. recognition of ‘press’ and ‘deletes’.
- **B: Nouns recognised**: e.g. recognition of ‘BACKSPACE’, ‘text’, ‘character’ and ‘left’.
- **C: Compounds recognised**: e.g. recognition of ‘Ami Pro’.
- **D: Phrase boundaries recognised**: e.g. recognition that ‘the selected text and one character to the left of the selected text’ is a noun phrase.
• **E: Predicate-Argument relations recognised:** e.g. recognition that the argument of ‘press’ is ‘BACKSPACE’.

• **F: Prepositional phrases attached:** e.g. recognition that ‘to the left of the selected text’ attaches to ‘one character’ and not to ‘deletes’.

• **G: Co-ordination/Gapping analysed:** e.g. recognition that the components of the noun phrase ‘the selected text and one character to the left of the selected text’ are ‘the selected text’ and ‘one character to the left of the selected text’, joined by the coordinator ‘and’.

In each chapter, the above information is presented in Tables X.1 and X.2, where X is the chapter number.

### 1.3.3 Coverage

An indication of the coverage of the parser is given in Tables X.3.1, X.3.2 and X.3.3. Each is in the same format and shows for each of the three sets of utterances (Dynix, Lotus and Trados) the number which could be accepted. A parser is deemed to accept an utterance if it can produce some analysis for it. Otherwise it is deemed to reject the utterance. The three tables present this data for Phases I, II and III respectively.

### 1.3.4 Efficiency

An indication of the efficiency of the parser is given in Tables X.4.1, X.4.2 and X.4.3. Each is in the same format and shows for each of the three sets of utterances (Dynix, Lotus and Trados) the total time taken to attempt an analysis of all utterances, together with the average time taken to accept, or to reject, an utterance. Once again the three tables correspond to Phases I, II and III.

The type of machine used for testing is also specified in each chapter. While these tables only constitute a guide to performance, it is still worth noting that parse times for different systems vary from fractions of a second on a slow machine up to hours on a very fast one. The reason for including both average time to accept and average time to reject is that many systems are much slower at rejecting utterances than at accepting them. This is because a parser can accept an utterance as soon as it finds an interpretation of it, whereas to reject it, all possible interpretations must first be tried.
1.3.5 Accuracy of Analysis

Tables X.5.1, X.5.2 and X.5.3 provide an analysis of the ability of the parsing system to perform the syntactic analyses A to G which were discussed earlier. Once again the tables correspond to the three phases of the study.

The way in which the percentages in Tables X.5.1, X.5.2 and X.5.3 are computed is now defined. If an utterance can be recognised, then we compute its scores as follows: First, we determine how many of the particular construction it has, calling the answer \( I \). Second, we determine of those, how many are correctly recognised by the parser, calling the answer \( J \).

If an utterance cannot be recognised then we determine its scores as follows: First, we determine how many of the particular construction it has, calling the answer \( I \). Second, \( J \) is considered to have the value zero, because by definition the parser did not find any instances of the construction.

We now determine the figure in the table for each column as follows: First, we compute the sum of the \( I \)s for each utterance \( u \), \( \sum_u I \) and the sum of the \( J \)s for each utterance \( u \), \( \sum_u J \). Second, we compute the value:

\[
\frac{\sum_u J}{\sum_u I} \times 100
\]

In considering these tables, and indeed X.1 and X.2 also, it is important to bear in mind that the interpretation of each analysis can differ from system to system, depending on the grammatical formalism on which it is based. Even relative to a particular formalism, the results of the analysis can vary depending on what is considered the ‘correct’ interpretation of particular utterances. However, these tables do give an indication of how well the various systems perform on different types of syntactic analysis. In addition, problems of their interpretation relative to particular systems are fully discussed in the accompanying texts.

1.4 Structure of the Book

1.4.1 Introductory Chapters

Two introductory chapters preface those relating to specific parsers. The first, by Dekang Lin, justifies the use of a dependency notation as a basis for parser evaluation, and assesses the extent to which it is applicable to the output of the eight parsers described in this book. The second introductory chapter, by Eric Atwell, is a comparative analysis of the output data produced by the IPSM parsers, relating this both to dependency notation and other forms which have been proposed as standards. The
In introduction, Lin and Atwell's chapters together are an attempt to move forward the debate relating to the standardisation of parse data to facilitate both the evaluation of parsers and their integration into language engineering systems.

1.4.2 Parsing Chapters

Each parser is discussed in a separate chapter which is organised around a fixed set of headings. The content of each section is outlined below:

- **Introduction**: A brief introduction to the chapter.

- **Description of Parsing System**: An outline of the parsing system, including the algorithms used and the underlying linguistic formalisms involved.

- **Parser Evaluation Criteria**: A discussion of any parser-specific issues which had to be addressed during the process of evaluation. This is an important topic because not all criteria were applicable to all parsers. For example it is not possible to measure the accuracy of prepositional phrase attachment if a system is not designed to identify prepositional phrases.

- **Analysis I: Original Grammar, Original Vocabulary**: the results of Analysis I when applied to the parser. (See Section 1.3.1 above for discussion.)

- **Analysis II: Original Grammar, Additional Vocabulary**: The results of Analysis II when applied to the parser. (See Section 1.3.1 above for discussion.)

- **Analysis III: Altered Grammar, Additional Vocabulary**: The results of Analysis III when applied to the parser. (See Section 1.3.1 above for discussion.)

- **Converting Parse Tree to Dependency Notation**: a discussion of the problems incurred when an attempt was made to translate parse trees into a dependency form.

- **Summary of Findings**: a general summary of the findings relating to the parser study as a whole.

- **References**: a list of bibliographic references. These are also collated at the end of the volume.
1.4.3 Appendices

Appendix I lists the 60 test sentences which were used for the analysis described in this book. Appendix II gives a sample of parse trees as produced by the eight parsers. Finally, Appendix III is a collated list of all bibliographic references which occur within the book.

1.5 Discussion

In this section we make some concluding remarks relating to the project as a whole. Firstly, carrying out the work within a single text domain has proved useful in a number of respects. One of the most interesting findings of the analysis of utterance type in the original IPSM Corpus (Table 1.1) is that 43% of utterances are not in fact sentences at all. Nevertheless we wish to be able to analyse them accurately. This implies that an effective robust parser must not be tied to traditional notions of grammaticality.

While much of the corpus is regular, constructs occasionally occur which can not reasonably be analysed by any parser. The ability to return partial analyses in such cases is extremely valuable.

Secondly, progress has been made towards our original goal of providing a direct comparison between different parsers. For example the parse trees of Appendix II provide much useful information regarding the characteristics of the different systems which goes beyond what is discussed in the text. On the other hand, the range of parsing algorithms presented here is extremely wide which means that there are very few linguistic assumptions common to all systems. For example, when we talk about a ‘noun phrase’ each participant conjures up a different concept. Direct comparisons between systems are therefore difficult.

Tables X.5.1, X.5.2 and X.5.3 provide useful and interesting data regarding the efficacy of the different parsers. However, each participant has had to make a different set of linguistic assumptions in order to provide this information. Ideally we would like to have constrained the process more and to have based the results on a larger set of utterances. This might be accomplished in future by focusing on a task such as predicate-argument extraction which is closely related to parsing and can also be assessed automatically.

In conclusion, IPSM has proved to be an interesting and constructive exercise.
1.6 Acknowledgements

This introduction would not be complete without acknowledging the help of many people. The most important of these are:

- The copyright holders of the Dynix, Lotus and Trados software manuals, for allowing extracts from their documents to be used in the research;

- Helen J. Wybrants of Dynix Library Systems, Michael C. Ferris of Lotus Development Ireland and Matthias Heyn of Trados GmbH for making the manuals available in machine-readable form;

- The National Software Directorate of Ireland (Director Barry Murphy, PAT Director Seamus Gallen) for funding some of the work under the project ‘Analysing Free Text with Link Grammars’;

- DGXIIE5 of the European Commission who provided travel assistance to some of the participants under the project SIFT (LRE-62030) (Programme Manager Roberto Cencioni, Project Officers Chris Garland and Lidia Pola);

- Denis Hickey, Tony Molloy and Redmond O’Brien who solved innumerable technical problems at Limerick relating to organisation of the IPSM workshop;

- The contributors to this volume, all of whom carried out two analyses and wrote two completely different articles describing their results.

1.7 References


Dependency-Based Parser Evaluation: A Study with a Software Manual Corpus

Dekang Lin

University of Manitoba

2.1 Introduction

With the emergence of broad-coverage parsers, quantitative evaluation of parsers becomes increasingly more important. It is generally accepted that parser evaluation should be conducted by comparing the parser-generated parse trees (we call them answers) with manually constructed parse trees (we call them keys). However, how such comparison should be performed is still subject to debate. Several proposals have been put forward (Black, Abney, Flickenger, Gdaniec, Grishman, Harrison, Hindle, Ingrino, Klavans, Liberman, Marcus, Roukos, Santorini & Strzalkowski, 1991; Black, Lafferty & Roukos, 1992; Magnerman, 1994), all of which are based on comparison of phrase boundaries between answers and keys.

There are several serious problems with the phrase boundary evaluations. First, the ultimate goal of syntactic analysis is to facilitate semantic interpretation. However, phrase boundaries do not have much to do with the meaning of a sentence. Consider the two parse trees for a sentence in the software manual corpus shown at the top of Figure 2.1. There are four phrases in the answer and three in the key, as shown at the bottom of the figure.

According to the phrase boundary method proposed in Black, Abney et al. (1991), the answer has no crossing brackets, 100% recall and 75%
Answer:

\[
\begin{align*}
&\text{CP} \\
&\quad\text{Chbar} \\
&\quad\text{IP} \\
&\quad\text{NP} \\
&\quad\text{Det A} \\
&\quad\text{Nbar} \\
&\quad\text{N BIB} \\
&\quad\text{CP} \\
&\quad\text{Op}[1] \\
&\quad\text{Chbar} \\
&\quad\text{IP} \\
&\quad\text{NP} \\
&\quad\text{Nbar} \\
&\quad\text{N summary} \\
&\quad\text{Ibar} \\
&\quad\text{VP} \\
&\quad\text{Vbar} \\
&\quad\text{V} \\
&\quad\text{V NP} \\
&\quad\text{V NP screen} \\
&\quad\text{t[I]} \quad \text{[V appears]} \quad \text{]} \quad \text{]\text{[V appears]}}} \\
&\end{align*}
\]

Key:

\[
\begin{align*}
&\text{CP} \\
&\quad\text{Chbar} \\
&\quad\text{IP} \\
&\quad\text{NP} \\
&\quad\text{Det A} \\
&\quad\text{Nbar} \\
&\quad\text{N BIB summary screen} \\
&\quad\text{IP} \\
&\quad\text{Ibar} \\
&\quad\text{VP} \\
&\quad\text{Vbar} \\
&\quad\text{V appears} \\
&\end{align*}
\]

Phrases in Answer:  Phrases in Key:

- summary screen
- BIB summary screen
- A BIB summary screen
- A BIB summary screen appears
- A BIB summary screen appears

Figure 2.1: Two parse trees of “A BIB summary screen appears”.

precision, which are considered to be very good scores. However, the answer treats “screen” as a verb and “summary screen” as a relative clause modifying the noun “BIB.” This is obviously a very poor analysis and unlikely leads to a correct interpretation of the sentence. Therefore, parse trees should be evaluated according to more semantically relevant features than phrase boundaries.

Another problem with phrase boundary evaluation is that many differences in phrase boundaries are caused by systematic differences between different parsing schemes or theories. For example, Figure 2.2 shows two parse trees for the same sentence. The first one is from the SUSANNE corpus (Sampson, 1995), the second one is the output by PRINCIPAR (Lin, 1994). Although the two parse trees look very different, both of them are correct according to their own theory. An evalua-
### 2.2 Dependency-Based Evaluation

In Lin (1995), we proposed a dependency-based evaluation method. Since semantic dependencies are embedded in syntactic dependencies, the results of the dependency-based evaluation are much more meaningful than those of phrase boundary methods. Furthermore, it was shown in Lin (1995) that many systematic differences among different theories can be eliminated by rule-based transformation on dependency trees.

In the dependency-based method, the parser outputs and treebank parses are first converted into dependency trees (Mel'čuk, 1987), where every word is a modifier of exactly one other word (called its head or modifiee), unless the word is the head of the sentence or a fragment of the sentence in case the parser failed to find a complete parse of the sentence. Figures 2.2a and 2.2b depict the dependency trees corresponding to Figures 2.2a and 2.2b respectively. An algorithm for transforming constituency trees into dependency trees was presented in Lin (1995).

A dependency tree is made up of a set of dependency relationships. A dependency relationship consists of a modifier, a modifiee and (optionally) a label that specifies the type of the dependency relationship. Since a word may participate as the modifier in at most one dependency relationship, the dependency scheme should not arbitrarily prefer one and penalize the other.

---

**Figure 2.2:** Two different phrase structure analysis of the same sentence

<table>
<thead>
<tr>
<th>a. SUSANNE parse tree</th>
<th>b. PRINCIPAR parse tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S</td>
<td>(CP (Ch (IP</td>
</tr>
<tr>
<td>(Ns:n</td>
<td>(Det The)</td>
</tr>
<tr>
<td>(AT The)</td>
<td>(M Maguire)</td>
</tr>
<tr>
<td>(Mn:np Maguire))</td>
<td>(M family))</td>
</tr>
<tr>
<td>(NN:np family))</td>
<td>(N:np</td>
</tr>
<tr>
<td>(V:np was)</td>
<td>(Be was)</td>
</tr>
<tr>
<td>(VP:np setting))</td>
<td>(NP:np</td>
</tr>
<tr>
<td>(R:n (NP=n))</td>
<td>(V (V [NP</td>
</tr>
<tr>
<td>(Ns:o)</td>
<td>(V (V _ [NP</td>
</tr>
<tr>
<td>(ATl a)</td>
<td>setting up)</td>
</tr>
<tr>
<td>(JJ separate)</td>
<td>(Det a)</td>
</tr>
<tr>
<td>(NNl:np camp))</td>
<td>(Mbar</td>
</tr>
<tr>
<td>(R:p (RL nearby)))</td>
<td>(AP (Abar (A separate))</td>
</tr>
<tr>
<td></td>
<td>(M camp))))</td>
</tr>
</tbody>
</table>
|                       |     (AP (Abar (A nearby))))))))

---
relationship, we may treat the modifiee in a dependency relationship as the tag assigned to the modifier. Parser outputs can then be scored on a word-by-word basis similar to the evaluation of the part-of-speech tagging results.

For each word in the answer, we can classify it into one of the four categories:

- if it modifies the same word in the answer and in the key or it modifies no other word in both the answer and the key, it is considered to be **correct**.
- if it modifies a different word in the answer than in the key, it is considered to be **incorrect**.
- if the word does not modify any word in the answer, but modifies
<table>
<thead>
<tr>
<th>Parser</th>
<th>Dependency</th>
<th>Constituency</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALICE</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENGCG</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LPARSER</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRINCIPAR</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>RANLT</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEXTANT</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DESPAR</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOSCA</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 2.1: Output format of IPSM’95 parsers*

- if a word in the key, then it is **missing** a modifie.
- if the word does not modify any word in the key, but modifies a word in the answer, then it has a **spurious** modifie.

For example, if we compare the two dependency trees in Figures 2.3a and 2.3b, all the words are correct, except the word **nearby** which has different modifiees in the key and in the answer (was vs. setting).

### 2.3 Manual Normalization of Parser Outputs

Table 2.1 shows the output formats of the parsers that participated in the IPSM workshop.

Given that the parsing literature is dominated by constituency-based parsers and all the large tree banks used constituency grammars, it is surprising to find that there are more dependency-based parsers in the workshop than constituency-based ones.

In order to apply the dependency-based method to evaluate the participating parsers, the workshop participants conducted an experiment in which each participant manually translated their own parser outputs for a selected sentence into a dependency format similar to what was used in PRINCIPAR. For dependency-based parsers, this is quite straightforward. Essentially the same kind of information is encoded in the outputs of these parsers. The distinctions are mostly superficial. For example, in both SEXTANT and DESPAR, words are assigned indices. Dependency relationships are denoted by pairs word indices. SEXTANT uses an integer (0 or 1) to specify the direction of the dependency relationship, whereas DESPAR uses an arrow to indicate the direction. In PRINCI-
Figure 2.4: The dependency trees for the sentence “The contents of the Clipboard appear in the desired location.”

PAR, the dependency relationships are specified using relative positions instead of absolute indices of words.
Besides the superficial distinctions between the dependency formats, significant differences do exist among the representations in SEXTANT, DESPAR and PRINCIPAR. For example, in SEXTANT, the preposition "of" in "of the clipboard" is a modifier of "clipboard," whereas in DESPAR and PRINCIPAR, "clipboard" is a modifier of "of." These differences, however, can be eliminated by transformations in dependency trees proposed in Lin (1995).

Experiments on manual translation of constituency trees into dependency trees were also conducted. The main concern was that some important information gets lost when the trees are translated into dependency trees. For example, the positions of traces and the feature values of the nodes in the parse trees. Some of the participants felt that the parsers cannot be fairly compared when this information is thrown away. Since the feature values, like category symbols, varies a lot from one parser/grammar to another, the loss of this information is not due to the transformation into dependency trees, but a necessary consequence of comparing different parsers. The loss of information about the positions of traces, on the other hand, is a legitimate concern that still needs to be addressed.

2.4 Automated Transformation from Constituency to Dependency

An algorithm for converting constituency trees into dependency trees is presented in Lin (1995). The conversion algorithm makes use of a conversion table that is similar to Magerman's Tree Head Table for determining heads (lexical representatives) in CFG parse trees (Magerman, 1994, pp. 64-66).

An entry in the conversion table is a tuple:

\[(\text{<mother> [<direction>] \{<child-1> <child-2> ... <child-k>\}})\]

where <mother>, and <child-1>, <child-2>, ..., <child-k> are conditions on category symbols and <direction> is either 'l' (default) or 'r'. If a node \(a\) in the constituency tree satisfies the condition <mother>, then its head child is determined in the following two steps:

1. Find the first condition <child-i> in <child-1>, <child-2>, ..., and <child-k> such that one of the child nodes of \(a\) satisfies <child-i>;

2. If <direction> is \(l\), then the head child is the first child of \(a\) that satisfies the condition <child-i>; or if <direction> is \(r\) or is absent, it is the last child of \(a\) that satisfies the condition <child-i>.
Figure 2.5: A RANLT parse tree

The condition \( \text{reg-match} \langle \text{reg-exp} \rangle \ldots \langle \text{reg-exp} \rangle \) is satisfied if a prefix of the category symbol matches one of the regular expressions. For example:

\[
\text{reg-match} \ "[0-9]+\$" \ "[A-Z]\"
\]

returns true if its argument is an integer or a capitalized word. If the condition is an atom, then it is a shorthand for the condition \( \text{reg-match} \langle \text{atom} \rangle \), e.g., \( N \) is a shorthand for \( \text{reg-match} \langle N \rangle \).

For example, the following table was used to convert RANLT parse trees into dependency trees.

\[
\begin{align*}
(N (N)) \\
(V l ((\text{lexical}) V)) \\
(S l (V S)) \\
(A (A)) \\
(P l (P)) \\
((t) l ((t))))
\end{align*}
\]

The entry \( (N (N)) \) means that the head child of a node whose label begins with \( N \), such as \( N/1/- \) and \( N/2+/\text{DET}1a \), is the right most child whose label begins with \( N \). The entry \( (V l ((\text{lexical}) V)) \) means that the head child of a node beginning with \( V \) is the left most lexical child (i.e., a word in the sentence), or the left most child beginning with \( V \) if it does not have a lexical child. The condition \( (t) \) is satisfied by any node. Therefore, the last entry \( ((t) l ((t)))) \) means that if a node’s label does not match any of the above entries, then its head child is its left most child.

Consider the parse tree in Figure 2.5. The node \( N/2+/\text{DET}1a \) has two children: a lexical node “a” and \( N/2- \). The entry \( (N (N)) \) in the conversion table dictates that \( N/2- \) is the head child. Similarly, the head of \( N/2- \) is \( N/1/\text{APMOD}1 \) and the head child of \( N/1/\text{APMOD}1 \) is \( N/1/N \). The head child of \( N/1/N \) is its only child: the lexical node screen. Once the head child of each node is determined, the lexical head of the node is the
lexical head of its head child and the dependency tree can be constructed as follows: for each pair of a head child and a non-head child of a node, there is a dependency relationship between the lexical head of the head child and the lexical head of the non-head child.

The algorithm was also applied to TOSCA parses. An example of a TOSCA parse tree is shown as follows:

```
<tparn fun=UTT cat=S att=(act,decl,indic,intr,pres,unm)>
    <tparn fun=SU cat=NP>
        <tparn fun=DT cat=DTP>
            <tparn fun=DTCE cat=ART att=(indef)>A</tparn>
        </tparn>
        <tparn fun=NPHD cat=N att=(com,sing)>BIB summary screen</tparn>
    </tparn>
    <tparn fun=V cat=VP att=(act,indic,intr,pres)>
        <tparn fun=MVB cat=LV att=(indic,intr,pres)>appears</tparn>
    </tparn>
</tparn>
```

The nodes in the parse are annotated with functional categories (fun), syntactic categories (cat) and attribute values (att). We first used the following UNIX sed script to transform the representation into LISP-like list structures and remove functional categories and attribute values:

```
sed -e '    ///(//s/\*LRB\*/\g
    ///\s/\*RRB\*/\g
    ///\/(\<tparn \.* \)cat=\([^-]*\)\[^>\]*\)\s//\(\\2\)/\g
    ///\</tparn\>\s//\)/g
'
```

The output of the sed script for the above example parse is the following:

```
(S
    (NP
        (DTP
            (ART A
                ))
        (N BIB summary screen
            ))
    (VP
        (LV appears
            ))
)
```

The transformed parse tree is then converted to dependency structure with the following conversion table:
The resulting dependency tree is:

```
(A ~ ART < screen)
(BIB ~ U < screen)
(summary ~ U < screen)
(screen ~ N < appears)
(appears ~ LV *)
```

### 2.5 Conclusion

Parser evaluation is a very important issue for broad-coverage parsers. We pointed out several serious problems with the phrase boundary based evaluation methods and proposed a dependency based alternative. The dependency-based evaluation not only produces more meaningful scores, but also allows both dependency and constituency based parsers to be evaluated. We used the RANLT and TOSCA outputs as examples to show that constituency based parses can be automatically translated into dependency trees with a simple conversion table. This provides further evidence that the dependency-based evaluation method is able to accommodate a wide range of parsers.

### 2.6 References


Comparative Evaluation of Grammatical Annotation Models

Eric Steven Atwell
University of Leeds

3.1 Introduction

The objective of the IPSM Workshop was to empirically evaluate a number of robust parsers of English, in essence by giving each parser a common test-set of sentences, and counting how many of these sentences each parser could parse correctly. Unfortunately, what counts as a 'correct' parse is different for each parser, as the output of each is very different in both format and content: they each assume a different grammar model or parsing scheme for English. This chapter explores these differences in parsing schemes, and discusses how these differences should be taken into account in comparative evaluation of parsers. Chapter 2 suggests that one way to compare parser outputs is to convert them to a dependency structure. Others, (e.g. Atwell, 1988; Black, Garside & Leech, 1993) have advocated mapping parses onto simple context-free constituency structure trees. Unfortunately, in mapping some parsing schemes onto
this kind of 'lowest common factor', a lot of syntactic information is lost; this information is vital to some applications.

The differences between parsing schemes is a central issue in the project AMALGAM: Automatic Mapping Among Lexico-Grammatical Annotation Models. The AMALGAM project at Leeds University is investigating the problem of comparative assessment of rival syntactic analysis schemes. The focus of research is the variety of lexico-grammatical annotation models used in syntactically-analysed Corpora, principally those distributed by ICAME, the International Computer Archive of Modern English based at Bergen University. For more details, see Atwell, Hughes and Souter (1994a, 1994b), Hughes and Atwell (1994), Hughes, Souter and Atwell (1995), Atwell (1996), AMALGAM (1996) and ICAME (1996).

Standardisation of parsing schemes is also an issue for the European Union-funded project EAGLES: Expert Advisory Group on Language Engineering Standards (EAGLES, 1996). Particularly relevant is the 'Final Report and Guidelines for the Syntactic Annotation of Corpora' (Leech, Barnett & Kahrel, 1995); this proposes several layers of recommended and optional annotations, in a hierarchy of importance.

3.2 Diversity in Grammars

The parsers in this book are diverse, in that they use very different algorithms to find parse-trees. However, to a linguist, the differences in underlying grammars or parsing schemes are more important. The differences are not simply matters of representation or notation (although these alone cause significant problems in evaluation, e.g. in alignment). A crucial notion is delicacy or level of detail in grammatical classification. This chapter explores some possible metrics of delicacy, applied to comparative evaluation of the parsing schemes used in this book.

Delicacy of parsing scheme clearly impinges on the accuracy of a parser. A simple evaluation metric used for parsers in this book is to count how often the parse-tree found is 'correct', or how often the 'correct' parse-tree is among the set or forest of trees found by the parser. However, this metric is unfairly biased against more sophisticated grammars, which attempt to capture more fine-grained grammatical distinctions. On the other hand, this metric would favour an approach to syntax modelling which lacks this delicacy. Arguably it is not sensible to seek a scale of accuracy applicable across all applications, as different applications require different levels of parsing; see, for example, Souter

---

2DISCLAIMER: My description of the EAGLES guidelines for the syntactic annotation of corpora is based on the PRE-RELEASE FINAL DRAFT version of this Report, dated July 31st 1995; the final version, due for publication in 1996, may include some changes.
and Atwell (1994). For some applications, a skeletal parser is sufficient, so we can demand high accuracy: for example, n-gram grammar modelling for speech or script recognition systems (see next section); parsing corpus texts prior to input to a lexicographer’s KWIC workbench; or error-detection in Word Processor text. For these applications, parsing is simply an extra factor or guide towards an improved ‘hit rate’ - all could still work without syntactic analysis and annotation, but perform better with it. Other applications require detailed syntactic analysis, and cannot function without this; for example, SOME (but by no means all!) NLP systems assume that the parse-tree is to be passed on to a semantic component for knowledge extraction, a process requiring a richer syntactic annotation.

### 3.3 An Extreme Case: the ‘Perfect Parser’ from Speech Recognition

The variability of delicacy is exemplified by one approach to parsing which is widely used in Speech And Language Technology (SALT). Most large-vocabulary English speech recognition systems use a word N-gram language model of English grammar: syntactic knowledge is captured in a large table of word bigrams (pairs), trigrams (triples), ... N-grams (see surveys of large-vocabulary speech recognition systems, e.g. HLT Survey, 1995; comp.speech, 1996). This table is extracted or learnt from a training corpus, a representative set of texts in the domain of the speech recogniser; training involves making a record of every N-gram which appears in the training text, along with its frequency (e.g. in this Chapter the bigram recognition systems occurs 4 times). The ‘grammar’ does not make use of phrase-structure boundaries, or even word-classes such as Noun or Verb. The job of the ‘parser’ is not to compute a parse-tree for an input sentence, but to estimate a syntactic probability for the input word-sequence. The ‘parser’ is guaranteed to come up with SOME analysis (i.e. syntactic probability estimate) for ANY input sentence; in this sense it is a ‘perfect’ parser, outperforming all the other parsers in this book.

However, this sort of ‘parsing’ is inappropriate for many IPSM applications, where the assumption is that some sort of parse-tree is to be passed on to a semantic component for knowledge extraction. In linguistic terms, the Speech Recognition grammar model has insufficient delicacy (or no delicacy at all!).
3.4 The Corpus as Empirical Definition of Parsing Scheme

A major problem in comparative evaluation of parsing schemes is pinning down the DEFINITIONS of the parsing schemes in question. Generally the parser is a computer program which can at least in theory be directly examined and tested; we can evaluate the algorithm as well as the output. Parsing schemes tend to be more intangible and ephemeral: generally the parsing scheme exists principally in the mind of the expert human linguist, who decides on issues of delicacy and correctness of parser output. For most of the syntactically analysed corpora covered by the AMALGAM project, we have some ‘manual annotation handbook’ with general notes for guidance on definitions of categories; but these are not rigorously formal or definitive, nor are they all to the same standard or level of detail. For the AMALGAM project, we were forced to the pragmatic decision to accept the tagged/parsed Corpus itself as definitive of the tagging/parsing scheme for that Corpus. For example, for Tagged LOB, (Johansson, Atwell, Garside & Leech, 1986) constitutes a detailed manual, but for the SEC parsing scheme we have to rely on a list of categories and some examples of how to apply them; so we took the LOB and SEC annotated corpora themselves as definitive examples of respective syntactic analysis schemes.

Another reason for relying on the example data rather than explanatory manuals is the limitation of the human mind. Each lexicogrammatical annotation model for English is so complex that it takes an expert human linguist a long time, months or even years, to master it. For example, the definition of the SUSANNE parsing scheme is over 500 pages long (Sampson, 1995). To compare a variety of parsing schemes via such manuals, I would have to read, digest and comprehensively cross-reference several such tomes. Perhaps a couple of dozen linguists in the world could realistically claim to be experts in two rival Corpus parsing schemes, but I know of none who are masters of several. I have been forced to the conclusion that it is unreasonable to ask anyone to take on such a task (and I am not about to volunteer myself!).

This pragmatic approach is also necessary with the parsing schemes used in this book. Not all the parsing schemes in use have detailed definition handbooks, as far as I am aware; at the very least, I do not have access to all of them. So, comparative evaluation of parsing schemes must be based on the small corpus of test parse-trees presented at the IPSM workshop. Admittedly this only constitutes a small sample of each parsing scheme, but hopefully the samples are comparable subsets of complete grammars, covering the same set of phrase-types for each parsing scheme. This should be sufficient to at least give a relative
indicator of delicacy of the parsing schemes.

3.5 Towards a MultiTreebank

One advantage of the IPSM exercise is that all parsers were given the same sentences to parse, so we have directly-comparable parses for given sentences; the same is not true for ICAME parsed corpora, also called treebanks. Even if we assume that, for example, the Spoken English Corpus (SEC) treebank (Taylor & Knowles, 1988) embodies the definition of the SEC parsing scheme, the Polytechnic of Wales (POW) treebank (Souter, 1989) defines the POW parsing scheme, etc, there is still a problem in comparing delicacy across parsing schemes. The texts parsed in each treebank are different, which complicates comparison. For any phrase-type or construct in the SEC parsing scheme, it is not straightforward to see its equivalent in POW: this involves trawling through the POW treebank for similar word-sequences. It would be much more straightforward to have a single text sample parsed according to all the different schemes under investigation, a MultiTreebank. This would allow for direct comparisons of rival parses of the same phrase or sentence. However, creation of such a resource is very difficult, requiring the cooperation and time of the research teams responsible for each parsed corpus and/or robust parser.

A first step towards a prototype MultiTreebank was achieved in the Proceedings of the IPSM workshop, which contained the output of several parsers’ attempts to parse half a dozen example sentences taken from software manuals. Unfortunately each sentence caused problems for one or more of the parsers, so this mini-MultiTreebank has a lot of ‘holes’ or gaps. As an example for further investigation, I selected one of the shortest sentences (hence, hopefully, most grammatically straightforward and uncontroversial), which most parsers had managed to parse:

Select the text you want to protect.

To the example parses produced by IPSM participants, I have been able to add parses conformant to the parsing schemes of several large-scale English treebanks, with the assistance of experts in several of these parsing schemes; see AMALGAM (1996).

3.6 Vertical Strip Grammar: a Standard Representation for Parses

Before we can compare delicacy in the way two rival parsing-schemes annotate a sentence, we have to devise a parsing-scheme-neutral way
of representing rival parse-trees, or at least of mapping between the
schemes. I predict that most readers will be surprised by the wide
diversity of notation used by the parsers taking part in the IPSM workshop;
I certainly was. This can only confuse attempts to compare underlying
grammatical classification distinctions.

This is a major problem for the AMALGAM project. Even Corpora
which are merely wordtagged (without higher syntactic phrase bound-
daries marked) such as BNC, Brown etc, are formatted in a bewildering
variety of ways. As a ‘lowest common factor’, or rather, a ‘lowest com-
mon anchor-point’, each corpus could be visualised as a sequence of word
+ wordtag pairs. Even this simplification raises problems of incompatible
alignment and segmentation. Some lexico-grammatical annotation
schemes treat various idiomatic phrases, proper-name-sequences, etc as
a single token or ‘word’; whereas others split these into a sequence of
words to be assigned separate tags. Some parsing schemes split off cer-
tain affixes as separate lexemes or tokens requiring separate tags; while
others insist that a ‘word’ is any character-sequence delimited by spaces
or punctuation.

However, putting this tokenisation problem to one side, it is useful to
model any wordtagged Corpus as a simple sequence of word + wordtag
pairs. This can be used to build N-gram models of tag-combination
syntax. For full parses, the words in the sentence still constitute a ‘lowest
common anchor point’, so we have considered N-gram-like models of
parse-structures. For example, take the EAGLES basic parse-tree:

[S[VP select [NP the text [CL[NP you NP] [VP want [VP to
protect VP] [VP] CL[NP] [VP] . S]]

Words are ‘anchors’, with hypertags between them showing opening
and/or closing phrase boundaries. These hypertags are inter-word gram-
matical tokens alternating with the words, with a special NULL hypertag
to represent absence of inter-word phrase boundary:

[S[VP
select
[NP
the
NULL
text
[CL[NP
you
NP] [VP
want
[VP
to

When comparing rival parses for the same sentence, we can ‘cancel out’ the words as a common factor, leaving only the grammatical information assigned according to the parsing scheme. So, one way to normalise parse-structures would be to represent them as an alternating sequence of wordtags and inter-word structural information; this would render transparent the amount and delicacy of structural classificatory information. This would allow us to try quantitative comparison metrics, e.g. the length of the hypertext-string.

However, this way of building an N-gram like model is heavily reliant on phrase structure bracketing information, and so is not appropriate for some IPSM parsing schemes, those with few or no explicit phrase boundaries. The problem is that all the parses do have WORDS in common, but not all have inter-word bracketing information. An N-gram-like model which has states for words (but not inter-word states) may be more general. A variant N-gram-like model which meets this requirement is a Vertical Strip Grammar (VSG), as used in the Vertical Strip Parser (O’Donoghue, 1993). In this, a parse-tree is represented as a series of Vertical Strips from root to leaves. For example, given the syntax tree:

```
S--------------
|             |
VP--
|     |
|     NP----
|     |     CL--
|     |     |
|     |     NP VP--
|     |     |     |
|     |     |     VP--
|     |     |     |
select the text you want to protect.
```

This can be chopped into a series of Vertical Strips, one for each path from root S to each leaf:
This Vertical Strip representation is highly redundant, as the top of each strip shares its path from the root with its predecessor. So, the VSG representation only records the path to each leaf from the point of divergence from the previous Strip:

Sentence:
select the text you want to protect.

ALICE:
SENT VP-INF
AUX SENT INF-MARK VP-INF
? NP NP VP-ACT to protect
select DET NOUN you want
the text

ENGCG:
@+FMAINV @DN> @OBJ @SUBJ @+FMAINV @INFMARK> @-FMAINV.
V DET N PRON V INFMARK V
select the text you want to protect.
The ENGCG output is unusual in that it provides very detailed word-category labelling for each word, but only minimal structural information. In the above I have omitted the wordclass subcategory information, e.g.

select: &gt; &lt;SVO&gt; &lt;SV&gt; &lt;P/for&gt; V IMP VFNM

LPARSER:

```plaintext
  Q   B
  W   D   C   S   TO   I
  v    the    n    you    want    to
  select    text    protect
```

PRINCIPAR:

```plaintext
  VP
  Vbar
  V
  V_NP
  V_NP   NP
  select    Det    Nbar
  the    N    CP
  text    Op[1]
  Char
  IP
  NP    Ibar
  Nbar   VP
  N    Vbar
  you    V
  V_CP
  V_CP   CP
  want    Char
  IP
  PRO
  Ibar
  Aux    VP
  to    Vbar
  V
  V_NP
  V_NP
  protect
  t[1]
```
PLAIN:

ILLOC
command
PROPOS
*   DIR_OBJ1
imperat DETER *   ATTR_ANY
select definit singula rel_clause
text   PRED
   SUBJECT *   DIR_OBJ2
   you   present clause
   want   PROPOS
to   protect

RANLT:

VP/NP
select N2+/DET
   the
   N2-
   N1/INFM
   N1/RELM
   VP/TO
   N1/N
   S/THATL
   to
   VP/NP
text   Sia
   protect
   N2+/PRO
   VP/NP
   TRACE1
   you   want
   E
   TRACE1
   E

SEXTANT:

VP   NP   NP   VP   --
INF   3   *   *   INF   TO   4 : .
select DET   1   PRON   want   to   SUBJ : .
   DET   DOBJ   you
   the
   NOUN
   protect
text

DESPAR:

8   3   1   5   3   7   5   0
VB   DT   NN   PP   VBP   TO   VB : .
select the
   text
   you
   want
to   protect : .
TOSCA:

Unfortunately this was one of only a couple of IPSM test sentences that the TOSCA parser could not parse, due to the syntactic phenomenon known as ‘raising’: according to the TOSCA grammar, both the verbs ‘select’ and ‘protect’ require an object, and although in some deep sense ‘the text’ is the object of both, the TOSCA grammar does not allow for this construct. However, the TOSCA research team have kindly constructed a ‘correct’ parse for our example sentence, to compare with others, by parsing a similar sentence and then ‘hand-editing’ the similar parse-tree. This includes very detailed subclassification information with each label (see Section 3.7.5, which includes the TOSCA ‘correct’ parse-tree). For my VSG normalisation I have omitted this:

```
MVB, LV DT, DTP NPHD, N NPPO, CL
Select DT CE, ART text SU, NP V, VP OD, CL
you want to MVB, LV

3.7 EAGLES: A Multi-Layer Standard for Syntactic Annotation

This standard representation is still crude and appears unfair to some schemes, particularly dependency grammar which has no grammatical classes! Also, it assumes the parser produces a single correct parse-tree – is it fair to parsers (e.g. RANLT) which produce a forest of possible parses? It at least allows us to compare parser outputs more directly, and potentially to combine or merge syntactic information from different parsers.

Mapping onto a standard format allows us to focus on the substantive differences between parsing schemes. It turns out that delicacy is not a simple issue, as different parsers output very different kinds or levels of grammatical information. This brings us back to our earlier point: parsing schemes should be evaluated with respect to a given application, as different applications call for different levels of analysis.

To categorise these levels of grammatical analysis, we need a taxonomy of possible grammatical annotations. The EAGLES Draft Report on parsing schemes (Leech, Barnett & Kalred, 1995) suggests that these layers of annotation form a hierarchy of importance, summarised in Table 3.1 at the end of this section.
The Report does not attempt formal definitions or stipulate standardised labels to be used for all these levels, but it does give some illustrative examples. From these I have attempted to construct the layers of analysis for our standard example sentence.

3.7.1 (a) Bracketing of Segments

The Report advocates two formats for representing phrase structure, which it calls Horizontal Format and Vertical Format; see Atwell (1983). In both, opening and closing phrase boundaries are shown by square brackets between words; in horizontal format the text reads horizontally down the page, one word per line, while in vertical format the text reads left-to-right across the page, interspersed with phrase boundary brackets:

```
[[ select [ the text [[ you ][ want [ to protect ]] ] ] ] ]
```

3.7.2 (b) Labelling of Segments

This can also be represented compactly in vertical format:

```
[S[VP select [NP the text [CL[NP you NP][VP want [VP to protect VP][CL[NP][VP]]]][VP]]][S]
```

The EAGLES report recommends the use of the categories S (Sentence), CL (Clause), NP (Noun Phrase), VP (Verb Phrase), PP (Prepositional Phrase), ADVP (Adverb Phrase), ADJP (Adjective Phrase). Although the EAGLES standard does not stipulate any obligatory syntactic annotations, these phrase structure categories are recommended, while the remaining layers of annotation are optional. Thus the above EAGLES parse-tree can be viewed as a baseline ‘lowest common factor’ target for parsers to aim for.

3.7.3 (c) Showing Dependency Relations

The Report notes that: “as far as we know, the ENGCG parser is the only system of corpus annotation that uses dependency syntax”, which makes the ENGCG analysis a candidate for the de-facto EAGLES standard for this layer. However, the dependency analysis is only partial - the symbol > denotes that a word’s head follows, and only two such dependencies are indicated for our example sentence:

```
> >
select the text you want to protect .
```
The report cites three traditional ways of representing dependency analyses graphically; however, the first cited traditional method, using curved arrows drawn to link dependent words, is equivalent to the DESPAR method using word-reference numbers:

\[
\begin{array}{cccccccc}
8 & 3 & 1 & 5 & 3 & 7 & 5 & 0 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8
\end{array}
\]

select the text you want to protect.

3.7.4 (d) Indicating Functional Labels

The report cites SUSANNE, TOSCA and ENGCG as examples of parsing schemes which include syntactic function labels such as Subject, Object, Adjunct. In TOSCA output, every node-label is a pair of Function, Category, for example, SU,NP labels a Noun Phrase functioning as a Subject. In the ENGCG analysis, function is marked by @:

@+FMAINV OD @OBJ @SUB @+FMAINV @INFMARK @-FMAINV.

select the text you want to protect.

3.7.5 (e) Marking Subclassification of Syntactic Segments

Example subclassification features include marking a Noun Phrase as singular, or a verb Phrase as past tense. The TOSCA parser has one of the richest systems of subclassification, with several subcategory features attached to most nodes, lowercase features in brackets:

NOFU,TXTU()
UTT,S(-su,act,imper,motr,pres,unn)
V,VP(act,imper,motr,pres)
MVB,LV(imper,motr,pres){Select}
OD,NP()
DT,DTP()
DTCE,ART(def){the}
NPHD,N(com,sing){text}
NPPO,CL(+raised,act,indic,motr,pres,unn,zrel)
SU,NP()
NPHD,PN(pers){you}
V,VP(act,indic,motr,pres)
MVB,LV(indic,motr,pres){want}
OD,CL(-raised,-su,act,indic,infin,motr,unn,zsub)
TO,PRTCL(to){to}
V,VP(act,indic,infin,motr)
MVB,LV(indic,infin,motr){protect}
PUNC,PM(perm){.}
The ENGCG parsing scheme also includes subclassification features at the word-class level:

"select" <SVO> <SV> <P/for> V IMP VFIN
"the" <Def> DET CENTRAL ART SG/PL
"text" N NOM SG
"you" <NonMod> PRON PERS NOM SG/PL
"want" <SVOC/A> <SVO> <P/for> V PRES -SG3 VFIN
"to" INFMARK
"protect" <SVO> V INF

3.7.6 (f) Deep or ‘Logical’ Information

This includes traces or markers for extraposed or moved phrases, such as capturing the information that ‘the text’ is not just the Object of ‘select’ but also the (raised) Object of ‘protect’. This is captured by the features +raised and -raised in the above TOSCA parse-tree; by cross-indexing of Op[/1/] and t[/1/] in the PRINCIPAR parse; and by (TRACE1E) in the RANLT parse.

3.7.7 (g) Information about the Rank of a Syntactic Unit

The Report suggests that ‘the concept of rank is applied to general categories of constituents, words being of lower rank than phrases, phrases being of lower rank than clauses, and clauses being of lower rank than sentences’. This is not explicitly shown in most of the parser outputs, beyond the common convention that words are in lowercase while higher-rank units are in UPPERCASE or begin with an uppercase letter. However, I believe that the underlying grammar models used in PRINCIPAR and RANLT do include a rank hierarchy of nominal units: NP-Nbar-N in PRINCIPAR, NP-N2-n1-N in RANLT.

3.7.8 (h) Special Syntactic Characteristics of Spoken Language

This layer includes special syntactic annotations for “a range of phenomena that do not normally occur in written language corpora, such as blends, false starts, reiterations, and filled pauses”. As the IPSM test sentences were written rather than spoken texts, this layer does not apply to us. However, we have successfully applied the TOSCA and ENGCG parsers to spoken text transcripts at Leeds in the AMALGAM research project.
### Layer Explanation

<table>
<thead>
<tr>
<th>Layer</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>Bracketing of segments</td>
</tr>
<tr>
<td>(b)</td>
<td>Labelling of segments</td>
</tr>
<tr>
<td>(c)</td>
<td>Showing dependency relations</td>
</tr>
<tr>
<td>(d)</td>
<td>Indicating functional labels</td>
</tr>
<tr>
<td>(e)</td>
<td>Marking subclassification of syntactic segments</td>
</tr>
<tr>
<td>(f)</td>
<td>Deep or 'logical' information</td>
</tr>
<tr>
<td>(g)</td>
<td>Information about the rank of a syntactic unit</td>
</tr>
<tr>
<td>(h)</td>
<td>Special syntactic characteristics of spoken language</td>
</tr>
</tbody>
</table>

**Table 3.1:** EAGLES layers of syntactic annotation, forming a hierarchy of importance.

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Verbs recognised</td>
</tr>
<tr>
<td>B</td>
<td>Nouns recognised</td>
</tr>
<tr>
<td>C</td>
<td>Compounds recognised</td>
</tr>
<tr>
<td>D</td>
<td>Phrase Boundaries recognised</td>
</tr>
<tr>
<td>E</td>
<td>Predicate-Argument Relations identified</td>
</tr>
<tr>
<td>F</td>
<td>Prepositional Phrases attached</td>
</tr>
<tr>
<td>G</td>
<td>Coordination/Gapping analysed</td>
</tr>
</tbody>
</table>

**Table 3.2:** Characteristics used in IPSM parser evaluation.

### 3.7.9 Summary: a Hierarchy of Importance

Table 3.1 summarises the EAGLES layers of syntactic annotation, which form a hierarchy of importance. No parsing scheme includes all the layers (a)-(g) shown in the table; different IPSM parsers annotate with different subsets of the hierarchy.

### 3.8 Evaluating the IPSM Parsing Schemes against EAGLES

For the IPSM Workshop, each parsing scheme was evaluated in terms of "what kinds of structure the parser can in principle recognise". Each of the chapters after this one includes a table showing which of the characteristics in Table 3.2 are handled by the parser.

These characteristics are different from the layers of annotation in the EAGLES hierarchy, Table 3.1. They do not so much characterise the parsing scheme, but rather the degree to which the parser can apply it successfully. For example, criterion F does not ask whether the parsing
<table>
<thead>
<tr>
<th>Layer</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALICE</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>2</td>
</tr>
<tr>
<td>ENGCG</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>3</td>
</tr>
<tr>
<td>LPARSER</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>2</td>
</tr>
<tr>
<td>PLAIN</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>3</td>
</tr>
<tr>
<td>PRINCIPAR</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>5</td>
</tr>
<tr>
<td>RANLT</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>4</td>
</tr>
<tr>
<td>SEXTANT</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>4</td>
</tr>
<tr>
<td>DESPAR</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>TOSCA</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 3.3: Summary Comparative Evaluation of IPSM Grammatical Annotation Models, in terms of EAGLES layers of syntactic annotation. Each cell in the table is labelled yes or no to indicate whether an IPSM parsing scheme includes an EAGLES layer (at least partially). score is an indication of how many layers a parser covers.

scheme includes the notion of Prepositional Phrase (all except DESPAR do, although only PRINCIPAR and TOSCA explicitly use the label PP); rather it asks whether the parser is ‘in principle’ able to recognise and attach Prepositional Phrases correctly. Furthermore, most of the characteristics relate to broad categories at the ‘top’ layers of the EAGLES hierarchy.

Table 3.3 is my alternative attempt to characterise the rival parsing schemes, in terms of EAGLES layers of syntactic annotation. Each IPSM parsing scheme is evaluated according to each EAGLES criterion; and each parsing scheme gets a very crude overall ‘score’ showing how many EAGLES layers are handled, at least partially.

Note that this is based on my own analysis of output from the IPSM parsers, and I may have misunderstood some capabilities of the parsers. PRINCIPAR is unusual in being able to output two parses, to give both Dependency and Constituency analysis; I have included both in my analysis, hence its high ‘score’. The TOSCA analysis is based on the ‘handcrafted’ parse supplied by the TOSCA team, given that their parser failed with the example sentence; I am not clear whether the automatic parser can label deep or ‘logical’ information such as the raised Object of protect.
3.9 Summary and Conclusions

In this chapter, I have attempted the comparative evaluation of IPSM grammatical annotation models or parsing schemes. The first problem is that the great variety of output formats hides the underlying substantive similarities and differences. Others have proposed mapping all parser outputs onto a Phrase-Structure tree notation, but this is arguably inappropriate to the IPSM evaluation exercise, for at least two reasons:

1. several of the parsers (ENGCG, LPARSER, DESPAR) do not output traditional constituency structures, and

2. most of the parsers output other grammatical information which does not ‘fit’ and would be lost in a transformation to a simple phrase-structure tree.

The chapter by Lin proposes the alternative of mapping all parser outputs to a Dependency structure, but this is also inappropriate, for similar reasons:

1. most of the parsers do not output Dependency structures, so to force them into this minority representation would seem counter-intuitive; and

2. more importantly, most of the grammatical information output by the parsers would be lost in the transformation; dependency is only one of the layers of syntactic annotation identified by EAGLES.

In other words, mapping onto either constituency or dependency structure would constitute ‘degrading’ parser output to a lowest common factor, which is a particularly unfair evaluation procedure for parsers which produce ‘delicate’ analyses, covering several layers in the EAGLES hierarchy.

As an alternative, I have transformed IPSM parser outputs for a simple example sentence onto a compromise Vertical Strip Grammar format, which captures the grammatical information tied to each word, in a compact normalised form. The VSG format is derived from a constituent-structure tree, but it can accommodate partial structural information as output by the ENGCG and LPARSER systems. The VSG format is NOT intended for use in automatic parser evaluation experiments, as clearly the VSG forms of rival parser outputs are still clearly different, not straightforwardly comparable. The VSG format is intended as a tool to enable linguists to compare grammatical annotation models, by factoring out notational from substantive differences.

The EAGLES report on European standards for syntactic annotation identifies a hierarchy of levels of annotation. Transforming IPSM parser
<table>
<thead>
<tr>
<th>Layer</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALICE</td>
<td>7</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>ENCG</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>LPARSER</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>09</td>
</tr>
<tr>
<td>PLAIN</td>
<td>7</td>
<td>6</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>17</td>
</tr>
<tr>
<td>PRINCIPAR</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td>RANLT</td>
<td>7</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>SEXTANT</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>22</td>
</tr>
<tr>
<td>DESPAR</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>05</td>
</tr>
<tr>
<td>TOSCA</td>
<td>7</td>
<td>6</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>22</td>
</tr>
</tbody>
</table>

Table 3.4: Summary Comparative Evaluation of IPSM Grammatical Annotation Models, weighted in terms of EAGLES hierarchy of importance. Each cell in the table is given a weighted score if the IPSM parsing scheme includes an EAGLES layer (at least partially). score is a weighted overall measure of how many layers a parser covers.

outputs to a common notation is a useful exercise, in that it highlights the differences between IPSM parsing schemes. These differences can be categorised according to the EAGLES hierarchy of layers of importance. Table 3.3 in turn highlights the fact that no IPSM parser produces a ‘complete’ syntactic analysis, and that different parsers output different (overlapping) subsets of the complete picture.

One conclusion is to cast doubt on the value of parser evaluations based purely on success rates, speeds, etc without reference to the complexity of the underlying parsing scheme. At the very least, whatever score each IPSM parser achieves should be modified by a ‘parsing scheme coverage’ factor. Table 3.3 suggests that, for example, the PRINCIPAR and TOSCA teams should be given due allowance for the richer annotations they attempt to produce. A crude yet topical formula for weighting scores for success rate could be:

\[
\text{overall-score} = \text{success-rate} \times (\text{parsing-scheme-score} - 1)
\]

However, I assume this formula would not please everyone, particularly the DESPAR team! This weighting formula can be made even more controversial by taking the description hierarchy of importance at face value, and re-assigning each yes cell in Table 3.3 a numerical value on

\(^{3}\)At the time of writing, UK university researchers are all busy preparing for the HEFCS’ Research Assessment Exercise: all UK university departments are to have their research graded on a scale from 5 down to 1. RAE will determine future HEFCs funding for research; a possible formula is: Funding-per-researcher = N\(^{\text{Grade-1}}\), where N is a (quasi-)constant.
a sliding scale from 7 (a) down to 1 (g), as in Table 3.4. The TOSCA, SEXTANT and PRINCIPAR parsing schemes appear to be “best” as they cover more of the “important” layers of syntactic annotation.

A more useful conclusion is that prospective users of parsers should not take the IPSM parser success rates at face value. Rather, to repeat the point made in Section 3.2, it is not sensible to seek a scale of accuracy applicable across all applications. Different applications require different levels of parsing. Prospective users seeking a parser should first decide what they want from the parser. If they can frame their requirements in terms of the layers of annotation in Table 3.1, then they can eliminate parsers which cannot meet their requirements from Table 3.3. For example, the TOSCA parser was designed for use by researchers in Applied Linguistics and English Language Teaching, who require a complex parse with labelling similar to grammar conventions used in ELT textbooks. In practice, of the IPSM participants only the TOSCA parser produces output suitable for this application, so its users will probably continue to use it regardless of its comparative ‘score’ in terms of accuracy and speed.

To end on a positive note, this comparative evaluation of grammatical annotation schemes would not have been possible without the IPSM exercise, which generated output from a range of parsers for a common test corpus of sentences. It is high time for more linguists to take up this practical, empirical approach to comparing parsing schemes!

3.10 References


Research Into Language (pp. 143-158). Amsterdam, The Netherlands: Rodopi.
Using ALICE to Analyse a Software Manual Corpus

William J. Black
Philip Neal
UMIST

4.1 Introduction

The ALICE parser (Analysis of Linguistic Input to Computers in English) was developed for use in the CRISTAL project concerned with multilingual information retrieval (LRE project P 62-059). It is designed eventually to be used in open-ended situations without restrictions on user vocabulary, but where fragmentary analysis will be acceptable when complete parses turn out not to be possible. At all stages of its development, the emphasis has been on robustness at the expense of full grammatical coverage, and on a small lexicon augmented by morphological analysis of input words.

4.2 Description of Parsing System

The grammatical framework is Categorial Grammar (CG) (Wood, 1993), in which the words of a sentence are analysed as functions and arguments of each other by analogy with the structure of a mathematical equation. In the same way that the minus sign in “-2” is a functor forming a numeral out of a numeral and the equals sign in “2 + 3 = 5” one forming an equation from two numerals, categorial grammar analyses the adjective in “poor Jemima” as a functor forming a noun (phrase) out of a noun and the transitive verb in “France invaded Mexico” as a functor forming a sentence out of two nouns. The category of a function from a noun to...
a noun is symbolised as $n/n$ and a transitive verb as $s/n\langle n$, where the
direction of the slash represents the direction in which a functor seeks its
arguments. Arguments are given feature structures to force agreement
of number and gender, and to handle phenomena such as attachment
and gapping.

It is a property of Categorial Grammar that the syntactic potential
of words is represented entirely in the lexicon. The current parsing unit
only employs the two standard rules of left and right function application.
Rules of composition have been tried out in some versions of the parser
and may well be restored at a later stage.

ALICE operates in three phases: preprocessing, parsing and post-
parsing.

### 4.2.1 Preprocessing

Preprocessing is concerned with tokenisation and morphological analy-
sis. The purpose of the tokeniser is to recognise the distinction between
punctuation and decimal points, construct single tokens for compound
proper names, attempt to recognise dates, and other similar tasks; its
effect is to partition a text into proto-sentences.

The morphological analyser finds the likely parts of speech, and where
appropriate the inflected form of each token in the current sentence
string. An attempt is made to guess the part of speech of all content
words from their morphological and orthographic characteristics. A very
limited use is made of the position of words in the sentence. No use is
currently made of syntactic collocations (n-grams).

A final sub-phase of preprocessing inserts NP gaps into the well-
formed substring table at places where they could be used in relative
clauses and questions. Between stage I and stage III, some attempt was
made to extend the use of gaps to the analysis of complex conjunctions.
In future development of ALICE it is intended to make much more exten-
sive use of local constraints on the same principle as Constraint Grammar
(Voutilainen & Järvinen, 1996).

### 4.2.2 Parsing

The parsing system has the following characteristics:

- It is based on a well-formed substring table (a chart but not an
  active chart).

- Parsing proceeds bottom up and left to right, with edges added
to the data structure as rules are completed (rather than when
  “fired” as in the Active Chart parsing algorithm).
- Rules are in Chomsky Normal Form; that is, each rule has exactly two daughters.

- Term Unification is supported as the means of expressing feature constraints and the construction of phrasal representations.

- The parser uses the predicate-argument analysis to construct a semantics for an input string in which lexemes, and some purely grammatical features, are treated as quasi-logical predicates with instantiated variables. This was originally regarded as the principal output, to be used in conjunction with an inference system able to construct from it a disambiguated scoped logical representation. None of this has been included in the delivered parses.

The lexicon when used in stage I of the tests had about 800 word entries, and about 1200 word entries by stage III of the tests. The content vocabulary was mostly drawn from the financial domain, and the modifications between the stages of the tests were motivated by the demands of the CRISTAL project rather than the tests themselves. The number of types of CG sign represented (roughly comparable to rules in a phrase structure formalism) was 117 at stage I and 139 at stage III.

### 4.2.3 Postprocessing

In addition to the default output based on predicate-argument structure, the postprocessing phase is able to extract a surface syntactic tree, which is useful for debugging the grammar. For the convenience of those who prefer to read trees rather than predicate-argument structure, the surface tree nodes bear labels corresponding to the CG signs actually used. This post-parsing translation is supported by a set of rules putting CG into correspondence with Phrase Structure Grammar (PSG), but there are about half as many such rules as there are category types in the lexicon, so some categories or category-feature combinations are conflated.

Generally, a sentence will have multiple parses or no complete parse. In the latter case the postprocessor extracts from the chart a set of well-formed fragments. Currently, this is done from right to left, picking out the longest fragment and recursively extracting from that point leftwards. This heuristic, and minor variations on it which have been tried, is not all that it might be (it characteristically fails to identify an entire verb group where the main complement has not been identified). The introduction of rules of composition should alter the operation of this rule.
<table>
<thead>
<tr>
<th>Characteristic</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALICE</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 4.1: Linguistic characteristics which can be detected by ALICE. See Table 4.2 for an explanation of the letter codes.

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Verbs recognised</td>
</tr>
<tr>
<td>B</td>
<td>Nouns recognised</td>
</tr>
<tr>
<td>C</td>
<td>Compounds recognised</td>
</tr>
<tr>
<td>D</td>
<td>Phrase Boundaries recognised</td>
</tr>
<tr>
<td>E</td>
<td>Predicate-Argument Relations identified</td>
</tr>
<tr>
<td>F</td>
<td>Prepositional Phrases attached</td>
</tr>
<tr>
<td>G</td>
<td>Coordination/Gapping analysed</td>
</tr>
</tbody>
</table>

Table 4.2: Letter codes used in Tables 4.1, 4.5.1 and 4.5.3

4.3 Parser Evaluation Criteria

We have evaluated the parser at two stages of its development: firstly its state shortly after the IPSM meeting in March 1995 (certain changes were made after the meeting, but linguistic coverage was not modified), and secondly its state in December 1995. The suggestion that improvements in the lexicon should be tested separately against the original grammar and against an improved grammar is not appropriate to the categorial formalism, in which almost all the syntactic information is incorporated into the lexicon. We have thus prepared two analyses but numbered them I and III for ease of comparison with the other results presented in this book.

ALICE recognises all seven characteristics prescribed in Table 4.1, in the sense that it attempts to recognise them and often succeeds.

We have claimed that ALICE accepted all 60 sentences of the test set: this means that for each sentence it produced either a full parse or a set of fragmentary parses spanning the whole sentence. Since it is not anticipated that ALICE will ever produce complete, accurate parses for all sentences of unrestricted input, we have concentrated on improvements which will produce accurate fragments and some accurate full parses rather than aiming for full parses at all costs.

The parsing times were obtained on a Sun workstation (ALICE is also portable to PC). They do not include the time taken to convert default output to tree notation.

Output in the form of tree notation was used to evaluate the success