Practical Unification-based Parsing of Natural Language

John Andrew Carroll

Computer Laboratory
University of Cambridge

September 1993

PhD Thesis

Acknowledgements[1]

I would like to thank Ted Briscoe, my supervisor, who prompted me to write this thesis. The ideas and suggestions he has come up with have had a large influence on the direction in which the research has developed. His comments on drafts of the thesis have been invaluable. Claire Grover was the first person to use the implementations of several of the algorithms described in the thesis; her prompt notification of problems has been a great help. I have appreciated the ease with which we are able to communicate, for long periods by electronic mail alone.

Discussions with Ann Copestake and Dick Crouch have been of great value. Thanks also to Antonio Sanfilippo for his help with training the parsing system described in chapter 6. I am grateful to all of the above, and in addition Bernie Jones, Arturo Trujillo, and Eirik Hektoen, for moral support. I would like to thank other members of the Alvey Natural Language Tools (ANLT) projects for help with practical issues over the years, users of the ANLT whose comments on early versions spurred me on to think about optimising parser throughput, Karen Sparck Jones for her continued support and interest, and Steve Pulman and Hiyan Alshawi for help and advice with a practical comparison with the CLARE system. I also appreciate helpful comments made by Mark-Jan Nederhof and David Weir.

Finally, I would like to thank my wife, Clare, for her support and encouragement, and my baby daughter Amy who provided effective diversion from the stresses involved!

[1] The research was supported in part by grant GR/D/05554 from the UK SERC, and SERC / DTI-IED project 4/1/1261 'Extensions to the Alvey Natural Language Tools'.

Summary

The thesis describes novel techniques and algorithms for the practical parsing of realistic Natural Language (NL) texts with a wide-coverage unification-based grammar of English. The thesis tackles two of the major problems in this area: firstly, the fact that parsing realistic inputs with such grammars can be computationally very expensive, and secondly, the observation that many analyses are often assigned to an input, only one of which usually forms the basis of the correct interpretation.

The thesis starts by presenting a new unification algorithm, justifies why it is well-suited to practical NL parsing, and describes a bottom-up active chart parser which employs this unification algorithm together with several other novel processing and optimisation techniques. Empirical results demonstrate that an implementation of this parser has significantly better practical performance than a comparable, state-of-the-art unification-based parser. Next, techniques for computing an LR table for a large unification grammar are described, a context free non-deterministic LR parsing algorithm is presented which has better time complexity than any previously reported using the same approach, and a unification-based version is derived. In experiments, the performance of an implementation of the latter is shown to exceed both the chart parser and also that of another efficient LR-like algorithm recently proposed.

Building on these methods, a system for parsing text taken from a given corpus is described which uses probabilistic techniques to identify the most plausible syntactic analyses for an input from the often large number licensed by the grammar. New techniques implemented include an incremental approach to semi-supervised training, a context-sensitive method of scoring sub-analyses, the accurate manipulation of probabilities during parsing, and the identification of the highest ranked analyses without exhaustive search. The system attains a similar success rate to approaches based on context-free grammar, but produces analyses which are more suitable for semantic processing.

The thesis includes detailed analyses of the worst-case space and time complexities of all the main algorithms described, and discusses the practical impact of the theoretical complexity results.

Contents

1. Introduction **1**

1.1 Goals of this Research 1

1.1.1 Problems in Practical Wide-coverage Natural Language Analysis 1

1.1.2 Primary Frameworks and Methods used in this Research 3

1.2 History of the Research 3

1.3 Background to the Research 5

1.3.1 Grammar Formalisms and Unification Grammar 5

1.3.2 Wide-coverage NL Grammars 7

1.3.3 Software Support for Grammar Writing 8

1.3.4 Phrase Structure Grammar Parsing 11

1.3.5 Parse Selection 15

1.4 Summary of the Thesis 16

1.4.1 Advances Reported in the Thesis 16

1.4.2 Overview of the Thesis 17

2. The ANLT Grammar Formalism and Unification Algorithm **19**

2.1 The ANLT Metagrammatical Formalism 19

2.1.1 Basic Design 19

2.1.2 Brief Description of the Formalism 20

2.2 The ANLT Grammar 24

2.3 The Object Grammar Formalism 26

2.3.1 Metagrammar Compilation 26

2.3.2 Format of the Object Grammar 28

2.4 Unification in the ANLT system 30

2.4.1 Definition of Term Unification 30

2.4.2 The Unification Algorithm 33

2.4.3 Category Copying and Subsumption 34

2.4.4 Extending the ANLT to Perform Graph Unification 38

2.4.5 Comparisons with other Unification Algorithms 39

3. The ANLT Chart Parser **43**

3.1 How the Parser Works 44

3.1.1 An Example 45

3.1.2 The Basic Algorithm 46
3.1.3 Dealing Efficiently with Ambiguity 56
3.2 Complexity of the Parser 60
3.2.1 Packing and Complexity 60
3.2.2 Time Complexity of a CF Variant of the Parser 61
3.2.3 Time Complexity of the Unification-based Parser 63
3.3 Practical Performance of the Parser 65
3.4 Experiments with Top-down Predictive Parsing 68

4. LR Parsing in a Unification-Based Grammar Framework 71
4.1 LR Parsing 71
4.1.1 Shift-Reduce Context-free LR Parsing 72
4.1.2 LR Parse Tables 73
4.1.3 Variants of LR 77
4.1.4 Extended Lookahead LR Parsing 78
4.2 Creating LR Parse Tables for Unification Grammars 79
4.2.1 Previous Approaches 79
4.2.2 Constructing a CF Backbone 81
4.2.3 Building LR Parse Tables for Large NL Grammars 86

5. Non-deterministic LR Parsing with Unification Grammars 91
5.1 Non-deterministic Context-free LR Parsing 91
5.5.1 Lang's Algorithm 91
5.1.2 Tomita's Algorithm 92
5.1.3 Complexity of Tomita's Algorithm 95
5.2 A New, Efficient CF LR Parsing Algorithm 98
5.2.1 The Basic Algorithm 98
5.2.2 Modifications to the Algorithm for Correctness and Efficiency 100
5.2.3 Complexity of the Modified Algorithm 105
5.2.4 Other Efficient Non-deterministic CF LR-like Algorithms 107
5.3 Non-deterministic Unification-based LR Parsing 108
5.3.1 Extending the CF Parsing Algorithm with Unification 108
5.3.2 Schabes' LR-like Parser 111
5.3.3 Complexity of Unification Versions of LR and Schabes' Parsers 113
5.3.4 Performance Comparison of Unification-based Chart and LR Parsers 114

6. Probabilistic LR Parsing 117
6.1 Probabilistic Approaches to Parsing 117
6.1.1 Previous Approaches 117

6.1.2 Problems with Probabilistic CFG 119
6.1.3 Language Models and Decision Procedures 122
6.2 Training using Interactive Incremental LR Parsing 123
6.2.1 Constructing a Disambiguated Training Corpus 123
6.2.2 The Interactive LR Parsing System 125
6.2.3 Deriving Probabilities from Training Data 128
6.3 Probabilistic LR Parsing 132
6.3.1 Computing a Probabilistic Parse Forest 133
6.3.2 Probabilistic Unpacking and Normalisation 134
6.4 Experiments Parsing LDOCE Definitions 138
6.4.1 The Experiments 138
6.4.2 Discussion of Experimental Results 142

7. Conclusions **144**
7.1 Maximising Parser Throughput 144
7.1.1 Parsing Techniques and Algorithms 144
7.1.2 Parsing and Complexity 146
7.2 Parse Selection and Tuning Grammars to Corpora 147
7.2.2 Acquisition of Lexical Entries and Word Sense Frequencies 148
7.2.3 Making Use of Collocational Information 149
7.3 Dealing with Undergeneration 150
7.4 Summary 152

Bibliography **154**

Appendix A **166**

Appendix B **170**

1. Introduction

1.1 Goals of this Research

This thesis describes research into techniques and algorithms for the practical parsing of realistic Natural Language texts with a wide-coverage unification-based grammar of English. The research proposes and implements novel solutions to two of the most important and difficult problems in this area: firstly, the fact that parsing realistic inputs with wide-coverage grammars can be computationally very expensive, and secondly, the observation that such grammars often assign many analyses to a typical input, only one of which usually forms the basis of the correct interpretation.

1.1.1 Problems in Practical Wide-coverage Natural Language Analysis

A substantial amount of recent research in Natural Language (NL) processing has focussed on the task of syntactically analysing substantial corpora of naturally-occurring text and transcribed speech. Analysed corpora would be of great benefit in the gathering of statistical data regarding language use, for example to train speech recognition devices, in more general linguistic research, and as a first step towards robust wide-coverage semantic interpretation.

The corpus analysis literature contains descriptions of a variety of different types of analysis, ranging from part-of-speech tagging to the assignment of a single, detailed syntactic analysis. However, to be able to derive a semantic interpretation from a semantically and pragmatically appropriate syntactic analysis, the analysis must determinately represent the syntactic relations which hold between all constituents in the input. In addition, the analysis must be drawn from a pre-defined, well-formed set of possible syntactic analyses, such as that defined by a generative grammar. If these two requirements are not met, then semantic interpretation of the resultant analyses cannot be guaranteed to be unambiguous and the semantic operations defined—over syntactic configurations—cannot be guaranteed always to apply when they are appropriate to yield a complete interpretation. Therefore, approaches which compute only lexical tags (e.g. de Rose, 1988) or a syntactic analysis which is the 'closest fit' to some previously defined set of possible analyses (e.g. Sampson, Haigh & Atwell, 1989), are inadequate in the absence of additional apparatus.

Until recently, most approaches to corpus analysis were based on the assumption that

computationally-tractable generative grammars of sufficiently general coverage could not be developed (see, for example, papers in Garside, Leech & Sampson, 1987). However, the recent development of wide-coverage declarative grammars written in computationally tractable and linguistically adequate formalisms makes this assumption questionable. For example, the Alvey Natural Language Tools (ANLT) system is a wide-coverage lexical, morphological and syntactic analysis system for English (Briscoe *et al.*, 1987a). Taylor, Grover & Briscoe (1989) demonstrate that a version of the sentence grammar in this system is capable of assigning the correct analysis to 96.8% of a corpus of 10,000 noun phrases extracted (without regard for their internal form) from a variety of corpora. However, although Taylor *et al.* show that the ANLT grammar has very wide coverage, they abstract away from issues of lexical idiosyncrasy by forming equivalence classes of noun phrases and then parse a single token of each class. They also do not address the practical issues of:

(1) ensuring good parser throughput in the face of ambiguity and long input strings,

(2) selecting the correct analysis for a particular input from the set licensed by the grammar,

(3) tuning a grammar to a particular corpus or sublanguage, and

(4) providing reliable analyses of input outside the coverage of the grammar.

Church & Patil (1982) demonstrate that for a broad coverage grammar parsing realistic input, there can be thousands of possible analyses. To be useful in practice, a parsing system must employ well-designed processing techniques to be able to cope with this number of analyses. The system must also incorporate automatic methods of filtering analyses to return only a small set comprising the semantically and pragmatically most plausible ones. Vocabulary, idiom and conventionalised constructions used in, say, legal language and word definitions in a dictionary will differ both in terms of the range and frequency of words and constructions employed, so a practical system must be capable of being tuned to the particular corpus or sublanguage under consideration. However, even when so tuned, it seems unlikely that any generative grammar will ever be capable of correctly analysing all naturally occurring input, so a system should be able to detect when an input falls outside the coverage of the grammar and take action to return an appropriate analysis for it.

This thesis describes an implemented computer system which tackles the first, second and third issues above, and makes some preliminary proposals concerning the fourth problem.

1.1.2 Primary Frameworks and Methods used in this Research

The previous section argued that a NL text analysis system must necessarily use a computationally tractable and linguistically adequate formalism in order to support semantic interpretation. Some recently proposed formalisms based on unification (Shieber, 1986a) satisfy these conditions, although few have had wide-coverage grammars developed in them, notable exceptions being the CLARE grammar in the Core Language Engine (CLE; Alshawi, 1992; Alshawi *et al.*, 1992) and the Alvey Natural Language Tools (ANLT), both for English. The research reported in this thesis is based on the use of the latter system; the grammar formalism and relevant aspects of the system are discussed fully.

The thesis describes a novel unification algorithm suitable for breadth-first parsing, and two unification-based parsers in detail, one a bottom-up chart parser, and the other, a non-deterministic LR parser. LR parser construction techniques are surveyed in detail, with particular reference to their utility for large, ambiguous grammars, and a methodology for deriving an LR parser from a unification grammar is presented. The parsing algorithms are derived from efficient non-deterministic algorithms for context-free grammar, on the premise that unification-based versions of such algorithms form a good basis for the implementation of parsers with good throughput. However, optimisations of the two unification parsers are described which considerably enhance the performance of the implementations. The thesis analyses the worst-case space and time complexities of the main algorithms described, and on the basis of the results of experiments, comments on the practical impact of the theoretical complexity results.

The approach to tuning a grammar to a particular corpus or sublanguage is based on a semi-automatic training phase during which additions to the grammar are made manually and statistical information concerning the frequency of use of grammar rules is acquired (within a context defined by the LR parsing technique). Using this statistical information and modified grammar, a probabilistic LR parser is constructed. The latter is capable of ranking the possible parses identified by the grammar in a useful and efficient manner. However, unseen sentences whose correct analysis is outside the coverage of the grammar remain a problem. Results from preliminary experiments suggest that the system has a success rate in the same region as other approaches based on context-free (CF) grammar, but produces much more detailed analyses than these other systems, ones which are much more suitable for subsequent semantic processing.

1.2 History of the Research

The research described in this thesis grew out of the initial implementation, subsequent enhancement, and observed shortcomings of the ANLT. The UK Alvey Programme originally

funded three parallel, collaborating projects at the Universities of Cambridge, Edinburgh and Lancaster to provide tools for use in NL processing research. The end product was the ANLT (release 1). One project (Phillips & Thompson, 1987) implemented a unification-based parser, and a second, a dictionary and morphological analyser system. The third project developed a large grammar of English (Grover, Carroll & Briscoe, 1993), and to support this effort, the Grammar Development Environment (GDE; Carroll, Briscoe & Grover, 1991). The grammar is written in a unification-based metagrammatical formalism whose notation is based on Generalized Phrase Structure Grammar (GPSG; Gazdar *et al.*, 1985). Subsequently, a fourth project, ending in September 1992, was funded by the UK Department of Trade and Industry and the Science and Engineering Research Council to support and enhance the ANLT, and add a semantic component—consisting of 'rule-to-rule' compositional translations—to the grammar and corresponding extensions to the GDE software. Three further releases of the ANLT were made between the end of the first three projects and the end of this fourth project[1].

By the end of 1991, the ANLT had been developed into a system for the morphological, syntactic and semantic analysis of a large subset of English[2]. The morphological component of the software integrates a finite-state grapheme-to-morpheme transducer (Koskenniemi, 1983) with a word grammar written in a unification-based formalism, and a lexicon access module, resulting in a general-purpose morphological analyser; Ritchie *et al.* (1992) give a detailed description. The lexicon used in the ANLT with this analyser contains about 64,000 entries for 40,000 lexemes, includes detailed subcategorisation information appropriate for the grammar, and was built semi-automatically from the *Longman Dictionary of Contemporary English* (LDOCE; Procter, 1978). Carroll & Grover (1989) describe this lexicon and the techniques used in its creation. The sentence grammar (Grover, Carroll & Briscoe, 1993) consists of a 'compiled' term unification grammar containing around 750 phrase structure rules, and a semantic component comprising rule-to-rule translations expressed in the lambda calculus; the corresponding software support is described by Carroll, Briscoe & Grover (1991). The ANLT contains two syntactic parsers: a bottom-up highly optimised chart parser, and a non-deterministic LR parser, described in this thesis in chapters 3 and 5 respectively.

Part of chapter 6 reports joint research with Ted Briscoe, and some of the text for this chapter and also for chapter 4 is based on papers by Carroll & Briscoe (1992) and Briscoe & Carroll (1991; 1993). Where joint work is described, this is stated in the text.

[1] The Alvey Natural Language Tools system, comprising the Common Lisp source code of the software, a wide-coverage grammar and lexicon for English, and comprehensive documentation, is available for a modest fee for use for research purposes and for a more substantial fee for commercial exploitation. Enquiries about obtaining the ANLT should be directed to: Lynxvale WCIU Programs, 20 Trumpington Street, Cambridge, CB2 1QA, UK (Fax: +44 223 332797). To date, around one hundred academic and industrial research sites worldwide have purchased a copy.

[2] In this thesis, the version of grammar and lexicon assumed is the third release (of December 1991) unless stated otherwise.

1.3 Background to the Research

1.3.1 Grammar Formalisms and Unification Grammar

Grammar Formalisms

A number of syntactic formalisms have been used to develop grammars for practical NL processing systems. The first of these to become widely used was Woods' (1970) Augmented Transition Network (ATN) formalism. In this formalism, a grammar is expressed as a set of directed graphs. When parsing, the arcs in the graphs are traversed and for each arc a set of associated actions is performed, for example to consume the next word in the input, build a segment of syntactic structure, or jump to a subsidiary network. An arc may also contain arbitrary Lisp expressions which are evaluated on traversal. The ATN formalism is very powerful since it can draw on the full power of a general-purpose programming language, but this makes grammars difficult to debug (since side-effects can proliferate) and also difficult to maintain and extend. Moreover, due to the procedural, order-dependent nature of the formalism, efficient parsing techniques cannot be used. For example, Carroll (1983) demonstrates that the automated analysis of dependencies is very complex, and shows that any parsing strategy apart from strictly left-to-right is impractical for large, general-purpose ATN grammars.

CF grammar (Chomsky, 1959) falls at the opposite extreme to ATNs. Efficient parsing techniques for it exist, but the formalism is not at all expressive, and for this reason in an unaugmented form it has been little used in practical NL processing systems. However, several systems have used variants of CF grammar (e.g. Robinson, 1982; Jensen *et al.*, 1986; Tomita, 1987) in which rules are augmented with tests and actions; these actions in some cases can be arbitrary procedures, having similar power to actions in an ATN grammar. Although these systems are much more flexible than CF grammar, any realistic grammar with substantial coverage would probably end up being just as difficult to debug, maintain and extend as an ATN grammar, due to the unconstrained nature of the actions (Shieber, 1986a).

More recently, several declarative theories have emerged, for example LFG (Kaplan & Bresnan, 1982), GPSG (Gazdar *et al.*, 1985), UCG (Zeevat, Klein & Calder, 1987) and HPSG (Pollard & Sag, 1987), whose detailed designs have been motivated more by theoretical linguistic than computational considerations. This generation of theories mostly uses unification of feature-based representations as a primitive operation (Shieber, 1986a). Although linguistically elegant and theoretically appealing, such theories tend not to be well suited to practical parsing with large grammars, since they often assume only minimal phrase structure components and instead rely on the (sometimes conceptually simultaneous) application of sets of principles or constraints to function. The worst case complexity of parsing with grammars expressed in these formalisms can be intractable (Barton, Berwick & Ristad, 1987). Inspired by

these theories, and as vehicles for further exploration of their properties and of the relationships between theories, flexible formalisms such as PATR (Shieber, 1984), FUG (Kay, 1985), TFS (Zajac, 1992), the LKB (Copestake, 1992) and ALE (Carpenter, 1992) have been developed which do not implement a single specific theory. However, these formalisms can often be used to simulate linguistic theories (e.g. Shieber, 1986b).

Recent projects to build wide-coverage unification-based grammars have tended to adopt relatively simple formalisms with straightforward computational interpretations (although the grammars draw on some of the linguistic insights made in the theories mentioned above). A reason for using a simpler formalism stems from the fact that (aside from their theoretical worst case complexity) implementing a complex theory and at the same time achieving reasonable average-case parsing performance with a large grammar is often extremely difficult. Another reason is that a direct implementation of a particular theory would be likely to become obsolete (or at least incapable of incorporating new linguistic analyses) relatively quickly (Boguraev *et al.*, 1988). However, although a very flexible formalism such as PATR might be able to model a wide range of theoretical frameworks, it has been considered to be unsuitable "for any major attempt at building natural-language grammars" (Shieber, 1984:364), due to reasons such as efficiency of encoding, and verbosity and redundancy of the formal notation.

As a consequence of the reasons outlined above, efforts directed towards supporting the development of wide-coverage unification grammars have invented new formalisms that express grammars at a higher level than, for example, PATR, but which nonetheless remain uncommitted to a particular linguistic framework. For example, the Core Language Engine (CLE; Alshawi, 1992) employs a phrase structure rule-based formalism employing a compact high level notation which is compiled into Prolog terms, thence using the built-in Prolog mechanisms to perform unification. The ANLT uses an expressively flexible and powerful *meta-grammatical* formalism which is automatically compiled into a strongly equivalent (in the sense that it describes exactly the same language and set of structural descriptions) *object grammar* that is a syntactic variant of the Definite Clause Grammar formalism (Pereira & Warren, 1980); the design of the object grammar allows a fast phrase structure parser using term unification to be employed.

Unification

The unification operation takes two feature-based representations, each consisting of a collection of attributes (features) and associated values (a value possibly being a further such collection), and combines the information contained in them to produce a representation that includes all the information of both. If they contain incompatible information, the unification operation *fails*, returning no result. When parsing a string, a grammar uses unification to assemble pieces of

linguistic information derived from various parts of the string into an analysis for the string, and to control the way in which the information is put together.

There are two main types of unification: *graph unification* and *term unification*. In the former, the sets of features appearing in the representations being unified, and order in which the features occur may differ; the features in the result of a successful unification are the union of those in the inputs. In the latter, for unification to succeed, the input representations, or *categories*, must contain exactly the same same set of features appearing in the same order; the result will contain just this set, and the order of the features will be the same. The unification operation that is built in to Prolog (Clocksin & Mellish, 1984) is term unification.

1.3.2 Wide-coverage NL Grammars

The DIAGRAM grammar (Robinson, 1982) forming part of SRI's TEAM natural language interface (Grosz *et al.*, 1987), and the PEG grammar from the Critique project at IBM (Jensen *et al.*, 1986) are widely recognised to be two of the widest coverage computational grammars that have been written to date, although they were developed in the nineteen-seventies and early eighties before unification-based formalisms started to be proposed for NL grammars. Both involved considerable amounts of human effort to build, and are based on formalisms comprising a rule-based skeleton augmented by procedural programming facilities associated with the rules. Robinson (1982) indicates that DIAGRAM at that time contained about one hundred constituent structure rules, and Jensen *et al.* (1986) state that PEG contained 235 decoding rules representing the "central [...] grammatical structures of English".

As part of the TOSCA project at Nijmegen, Oostdijk (1991) describes a grammar designed to analyse a one million word corpus of English written texts, drawn from a wide variety of sources. The grammar therefore covers a large subset of English, and is written in a formalism based on a restricted form of affix grammar (Koster, 1991); Briscoe (1993) conjectures that this variant of affix grammar is "probably similar in generative and expressive capacity to unification-based formalisms, such as PATR-II or the ANLT formalism [...]". The associated parsing system comprises two stages, the first assigning to each word of the input sentence an unambiguous lexical category that is compatible with the grammar, and the second, assigning analyses to this unambiguous sequence of lexical categories and allowing the correct analysis to be selected manually from the typically large set that is possible.

Hindle's (1989) Fidditch system is another corpus-oriented wide-coverage parsing system. Fidditch is *deterministic* (based on the approach of Marcus (1980)), pursuing a single path in analysing a sentence, and returning a single analysis or fragments of one if no complete analysis could be found. Fidditch distinguishes fifty atomic lexical categories, and contains pattern-action

rules (which match a single incomplete node and up to three completed constituents held in a buffer) to build analyses and to disambiguate lexical categories. De Marcken (1990) describes a system which has been used to analyse large portions of the LOB corpus (Garside, Leech & Sampson, 1987). It consists of a lexical tagger returning one (or occasionally more than one) category for each word, and a parser which takes the tags and builds analyses for phrases, not seeking to analyse sentences completely. The parser uses a relatively small set of CF-like rules, and can additionally combine neighbouring phrases into a binary structure, and deterministically discard certain phrases.

The CLARE grammar (Alshawi *et al.*, 1992) is a general-purpose wide-coverage unification-based grammar developed within the CLE framework (see section 1.3.1 above). In its general approach to obtaining wide coverage, CLARE has much in common with the ANLT grammar (section 2.2). The size and the syntactic and semantic coverage of the two grammars are broadly comparable, although CLARE is capable of deeper semantic and contextually determined interpretation.

1.3.3 Software Support for Grammar Writing

Grammar Formalisms and Software Systems

In Natural Language research, software systems implementing a particular linguistic theory or formalism have been used for three distinct tasks:

- during the development of a linguistic theory, to ensure that the theory and analyses based on it remain consistent when they are modified, and during investigation of the theory, to help elucidate the manner in which the various aspects of the theory interact with one another, and

- to provide a formalism for encoding linguistic information in a uniform way, in order to be able to compare and evaluate alternative linguistic theories, and

- to support the development of large, wide-coverage NL grammars.

Examples of the first of these types of system include the Grammar-Writer's Workbench (Kiparsky, 1985) for LFG, and ProGram (Evans, 1985) and GPSGP (Phillips & Thompson, 1985) for versions of GPSG. These systems are direct computational implementations of particular versions of the respective linguistic theories, and thus cannot be used to compare alternative theories since, in general, a grammar expressed in one theory cannot be encoded in

exactly the same way in another. Boguraev *et al.* (1988) argue that this lack of flexibility also makes this type of system unsuitable for the task of grammar development on a large scale. Although the applicability of these systems is limited outside their theoretical domain, they are invaluable for monitoring changes and maintaining consistency within the single linguistic theory which they implement.

Shieber (1984) developed the PATR formalism to be able to model a wide range of theoretical devices and analytical frameworks, in order to experiment with, and compare a variety of unification-based syntactic theories (Shieber, 1987); D-PATR (Karttunen, 1986) and CL-PATR (Shieber, 1988) are implementations of the formalism and examples of the second type of system. However, PATR is not a 'universal' formalism for encoding linguistic theories: in particular, it is not well-suited to constraint-based or typed feature structure formalisms. ELU (Estival, 1990) and UD (Johnson & Rupp, 1992) implement a more expressive variant of the PATR formalism, incorporating a language for specifying constraints (but not supporting typed feature structures). ALE (Carpenter, 1992) and TFS (Zajac, 1992) are more recent systems which directly implement types and also inheritance, and thus are perhaps better suited for experimenting with theories which use these devices, such as HPSG. However, as discussed above in section 1.3.1, the formalisms used in this type of system have so far not been demonstrated to be suitable for supporting the development of large grammars.

Wide coverage grammars typically require many person-years to build. They contain large numbers of rules which can interact in unpredictable ways, and it is generally acknowledged that software support is invaluable for their development. The third type of system listed above, often known as a *grammar development environment*, is designed to support such efforts. Examples of such systems include PLNLP (Jensen *et al.*, 1986), the METAL-SHOP module of the Machine Translation system METAL (White, 1987), the GDE (Carroll, Briscoe & Grover, 1991) and the GWB (Nederhof *et al.*, 1992). By analogy with the tools used to manage large computer programs in software engineering and those provided in interactive programming environments (Barstow, Shrobe & Sandewall, 1984), environments for grammar development support such tasks as: incremental input and editing of the grammar with on-line syntax checking, browsing and display of selected parts of the grammar based on particular combinations of attributes, static analysis to monitor interactions between different parts of the grammar, testing using special-purpose tools (such as user-driven generators to check for overgeneration), and tracing of dynamic rule behaviour during parsing to assist debugging.

The ANLT Grammar Development Environment

The grammar development environment—the GDE—that forms part of the ANLT is a 'second

generation' system, drawing on several ideas first implemented in systems such as ProGram and D-PATR. It was designed to be easy to use, to be portable and machine independent, and to enable a grammarian to achieve high productivity. The ANLT uses a meta-grammatical formalism which is compiled into an object grammar before parsing or generation. The GDE consists of a 'kernel' which performs the compilation of meta- into object grammar, and a number of integrated modules, the primary ones managing the user interface, grammar input and modification, generation, and interfacing to parsers.

The GDE imposes constraints on the way in which a parser operates so that only certain types are suitable for use during grammar development. The design requirement for high productivity means that the GDE should support a rapid, incremental style of grammar development. The implementation of metagrammar compilation is therefore optimised, and small modifications to the grammar usually result in only a minimal amount of processing before a new object grammar is available to the parser[3]. The parser must be able to accept the new object grammar and be ready to parse with it almost immediately. This constraint means that the parser must be able to operate without extensive postprocessing of the object grammar (e.g. constructing tables or automata). High productivity also implies that the parser should provide extensive support for interactive grammar debugging, for example the ability to display partial constituents built during the course of a parse, and to indicate which sub-analyses are shared between more than one complete analysis. Facilities based on interactive traces, for instance, are likely to be less useful, since, when parsing realistic inputs, large grammars typically lead to many thousands of steps being performed.

Even though only a limited amount of postprocessing is likely to be possible, a parser that is used in the development of a large grammar must have good practical performance, otherwise the grammarian will either be limited by the speed of the parser, or have to restrict the size of the grammar by specifying that only certain portions of it are to be used at any one time. Either alternative will slow down grammar development. The parser must also be *all-paths* (or *non-deterministic*), that is produce all possible parses of an input, otherwise the grammarian will again be burdened with deciding which parts of the parser's search space should not be explored during each parse.

The GDE allows more than one parser to be available for use at any given time (the ANLT contains two parsers: a bottom-up chart parser intended for grammar development, and an LR parser). The user may switch between parsers at any time. The parser interface contains a module which extracts semantic representations from complete syntactic analyses. When an input has been successfully analysed, the user can display the syntactic analyses in any of five formats (differing in the amount of detail they show), the sub-analyses that are shared between them, the syntactic category and semantics at individual nodes within a syntactic analysis, and the

[3] Metagrammar compilation in the GDE is incremental and automatic (see Boguraev *et al.*, 1988).

semantics (before or after beta-reduction) associated with each complete analysis. Additionally (with the chart parser), whether or not the input has been successfully parsed, the user can display selected subsets of edges in the chart. The GDE also contains facilities for automatically creating corpora of sentences and systematic batch testing of a grammar after changes have been made to it.

1.3.4 Phrase Structure Grammar Parsing

A *phrase structure grammar* consists of a set of *rules* (or *productions*), each of which contains a *mother* and a (possibly empty) ordered sequence of *daughters*; a rule is notated as `mother --> daughter`$_1$ `...` `daughter`$_n$. A *context-free* grammar is a phrase-structure grammar which consists of four elements: a set of (atomic) symbols, known as terminals; a set of non-terminal symbols; a set of phrase structure rules in which each mother is a non-terminal; and a designation of one of the non-terminals as the *start* symbol. Only strings (sequences of terminals) which may be derived from the start symbol by successive expansion of non-terminals are part of the language defined by the grammar.

Parsing Algorithms

Many phrase structure grammar parsing and recognition algorithms[4] that are suitable for use in NL processing have been proposed over the years; the first algorithm that is still of relevance nowadays is the Cocke-Kasami-Younger (CKY) algorithm (Kasami, 1965; Younger, 1967). This is a purely *bottom-up* method, that is, it always analyses fully all the daughters of a rule before constructing a constituent from its mother. The algorithm uses a tabular data-structure to record the state of the parse (in this case, all constituents found so far). The algorithm requires a CF grammar expressed in *Chomsky Normal Form* (CNF)[5], and has a worst-case time complexity of $O(n^3)$, where n is the length of the input[6]. Earley (1970) describes a CF parsing algorithm which is not restricted to CNF grammars but nevertheless also has cubic complexity. Leermakers (1989) shows that it achieves this by implicitly using a 2-form version of the grammar at parse time (i.e. each rule having at most two daughters), *dotted rules* (section 4.1.2) being new non-

[4] A parser returns an analysis (or set of analyses) for a grammatical input, whereas a recogniser only indicates whether an input is grammatical or not. The two types of algorithm are often related, in the sense that it may be possible to convert one into the other (see chapter 5 for an example of this).

[5] In CNF, each rule must contain either a single terminal daughter, or exactly two daughters both of which are non-terminals.

[6] Valiant (1975) shows that the computation performed by a CKY recogniser can be reduced to a boolean matrix multiplication problem, giving an algorithm that is still polynomial but with an order of less than three. Although Valiant's recogniser has better worst-case complexity than any other that is currently known, the constant factors are too large to make it of more than theoretical interest for the lengths of input encountered in NL processing.

terminals. To reduce the number of operations performed (although not in general the complexity), information is extracted from the grammar in order to *predict* in a top-down manner which rules could be applied at any given point during a parse, and eliminate from consideration others which could not possibly form part of a complete analysis at that point, given what has already been analysed.

Graham, Harrison & Ruzzo (1980) present a predictive recogniser that is derived from Earley's—in fact they state that "the CKY and Earley methods are "almost" identical [to it]"—but claim their recogniser to be conceptually simpler. Graham *et al.* describe versions of the recogniser which theoretically run in less than $O(n^3)$ steps, but state that these versions might not be useful in practice since the overheads associated with them may be too large. LR parsing (Knuth, 1965) is another important predictive technique, and is the most powerful (in the sense of precluding the need for non-deterministic search for the largest set of grammars) for which an automatic construction procedure is known. This, and some of the other predictive techniques, can be configured to look ahead at the next word (or words) in the input to further constrain prediction. Lang (1974) shows how Earley's technique can be adapted to any non-deterministic push-down automaton (NPDA) while still retaining cubic complexity. NPDAs can be constructed by powerful predictive methods, such as LR generalised to apply to ambiguous CF grammars. Tomita (1985) describes a more intuitive method for creating a non-deterministic LR parser, but the time complexity of this parser for certain kinds of grammar is worse than proportional to n^3. Improvements to Tomita's algorithm have subsequently been proposed; chapter 5 surveys this area in detail.

Predictive parsers are often claimed to have better average-case performance than non-predictive parsers (e.g. Schabes, 1991:106) as a result of the top-down filtering of applicable rules performed by the predictive component, but practical experiments with implementations of parsers—at least unification-based ones—do not bear out this claim in all cases (for further discussion see sections 3.4, 5.3 and Moore & Dowding, 1991). Also, *precompilation* to construct a predictive component from the grammar before parse time is liable to be CPU-intensive (e.g. Wright, Wrigley & Sharman, 1991; Briscoe & Carroll, 1991), potentially rendering the particular technique impractical for use in conjunction with grammar development, or for large, ambiguous grammars. Heering, Klint & Rekers (1989) note this problem and describe a system which, during parsing, incrementally compiles an LR-based predictive component. However, their system does not use lookahead and assumes the grammar incorporates an explicit CF component, so it is not clear that the techniques would be as successful or even usable on-line for large unification-based grammars, or if lookahead is required.

The predictive methods mentioned above (Earley, Graham *et al.*, LR) assume a CF grammar. However, they can still be applied where the grammar has a CF component but additionally contains augmentations to the rules; for instance Shieber (1985) uses *restriction*

sets to parse with a grammar containing feature-based categories that are subject to unification. Indeed, a method such as predictive left-corner parsing, which merely requires the computation of a *FIRST* table (Aho, Sethi & Ullman, 1986), can be applied to grammars which are not context-free, such as ones written in the CLE formalism, in which categories are feature-based and rules may have (non-initial) sequences of daughters specified by a single variable whose value is a list of categories. In a non-predictive, purely bottom-up parser, precompilation can still be of use, to index strategic parts of the grammar so that they can be accessed in an efficient manner at parse time (see section 3.1).

Augmented CF grammars are usually parsed with *on-line* algorithms, in which the augmentations associated with a rule are applied as soon as the rule is used, constraining the parsing process at the earliest possible opportunity. However, in situations where interacting constraints may be widely distributed across the input, an *off-line* algorithm (a version of which is used in the ELU parser; Estival, 1990) may be more appropriate. Such algorithms consist of two phases: parsing with a pure CF algorithm, followed by application of the constraints to all the resulting analyses. Maxwell & Kaplan (in press) evaluate these two strategies, and offer an empirical comparison with respect to an LFG grammar.

Some parsing algorithms are necessarily uni-directional, due to their control structure or the data-structures they use to represent the state of the parse: these algorithms must either process the input from left to right or (very unusually) from right to left, and construct all sub-analyses that will ever be needed for all of the input encountered so far before moving on to process the next word. For example, Tomita's (1985) LR parsing algorithm is uni-directional, since the (shift and reduce) operations which modify the stack holding the parse state must be synchronised with respect to the input, otherwise the search is not guaranteed to be *complete* (i.e. find all possible analyses). The CLE parser is based on the same shift-reduce procedure, and so must be uni-directional while pursuing a single derivation possibility, but the parser uses backtracking and thus may process the same words again (but still in a left-to-right order) when it backtracks.

A parser which uses a *depth-first* control strategy pursues a single analysis derivation at a time, recording possible alternatives so that processing can return (or *backtrack*) to them later. A depth-first parser must store in a well-formed substring table (Kuno, 1965) all constituents that it finds in order to prevent them from being re-analysed several times in exactly the same way, with a consequent loss in efficiency. *Breadth-first* parsers conceptually pursue all possible derivations simultaneously, also storing every constituent found in a data-structure; however, this data-structure also often contains (in an analogue of the internal system state information required to support backtracking) further information about the state of the parse at the point that the constituents were found. For example, in an active chart parser (Kay, 1986), the additional information consists of categories which are sought in the input at a certain point (active edges), so that sub-analyses can be combined in a consistent manner; in Earley Deduction (Pereira &

Warren, 1983) the information consists of non-unit definite clauses that can be further instantiated; in Tomita's LR algorithm, it consists of LR table state numbers which allow the parser to continue processing from any point at which it left off. Breadth-first algorithms that use tabular data-structures can in general be more flexible in their processing strategies than other types of algorithm. Thus, a CKY parser may process words in the input and fill in the table in any order as long as all of the row to the left and the column below in the table has already been filled in. An active chart parser can use a mixed bottom-up and top-down strategy (i.e. with rules triggered both on their mothers and daughters; Steel & De Roeck, 1987) as long as certain constraints are satisfied to ensure completeness. An agenda-based system, for instance MCHART (Thompson, 1983), can be parameterised to obtain either top-down or bottom-up behaviour; a system based on chart parsing or Earley Deduction can implement non-uni-directional strategies such as processing constituents in a head-driven order (Kay, 1989), or constructing analyses starting from lexical items that have been identified with relatively the greatest certainty (Stock, Falcone & Insinnamo, 1988).

Practical Parsing

As part of a complete NL analysis system, which will probably also contain a morphological analyser and semantic and pragmatic processing modules, a parser will typically contribute upwards of 25% to the total processing time (Alshawi, personal communication). When using wide-coverage grammars and analysing realistic text, the time taken for a complete analysis of a single sentence on current general-purpose computer hardware is sometimes of the order of tens of seconds. Processing times of this magnitude obviously place constraints on the tasks that the system can be called on to provide, and render impractical some types of on-line application such as enquiry systems, and those analysing complex and uncertain speech data or requiring the analysis of large volumes of machine-readable text, such as automatic summarisation and knowledge acquisition from documents. Therefore, the practical throughput of a parser can be important.

A parsing algorithm is termed *efficient* if its time complexity is polynomial. It is not the case, though, that a straightforward implementation of an efficient algorithm will result in a practical parsing system. Attention must be paid to the computational techniques used (Earley, 1970:97), and also to how augmentations to rules are dealt with, since operations such as unification can easily dominate parse times. However, most practical, wide-coverage NL analysis systems have indeed been based on formalisms for which computationally tractable parsing algorithms are known. Thus, no system in this area has employed an unmodified implementation of GPSG (shown by Barton, Berwick & Ristad (1987) to have EXP-POLY complexity for

recognition), LFG (which Barton *et al.* show to be NP-hard), or ID/LP format (Gazdar *et al.*, 1985; Sharman, 1989; shown by Barton *et al.* to be NP-complete). Even TAG, for which the best parsing algorithm currently known has worse than cubic complexity (in fact $O(n^6)$; Vijay-Shanker & Joshi, 1985), has not been used in such systems.

1.3.5 Parse Selection

A parser forming part of a complete NL analysis system must incorporate a mechanism for identifying the most plausible analysis (or analyses) for a particular input from the (often large) set licensed by the grammar. There are essentially three approaches to this problem:

(1) applying preference heuristics which encode observed tendencies for certain types of syntactic structure to predominate in corpora,

(2) scoring alternative semantic analyses on the basis of numerical weights assigned to particular semantic configurations, and

(3) ranking syntactic analyses based on probabilities associated with syntactic rules, derived from the rules' frequency of occurrence in a parsed (training) corpus.

The first approach has been investigated particularly with respect to the resolution of prepositional phrase (PP) attachment ambiguities. For example, Kimball (1973) observes that PPs often attach low (i.e. to the adjacent constituent to the left); Wilks, Huang & Fass (1985) indicate that temporal PPs may have a preference for attaching to events that themselves have temporal qualities; previous entities in the discourse may in certain cases also have an effect (Hirst, 1987). Using data from naturally occurring dialogues, Whittemore & Ferrara (1990) empirically compare several such heuristics that have been put forward in the literature and propose an improved deterministic procedure for parse selection. The procedure, in common with other approaches of this type, is only applicable to a single, specific source of ambiguity (i.e. PP attachment), and thus is not a general-purpose technique.

In the second approach, alternative semantic analyses are ranked by combining numerical weights previously assigned to semantic configurations. Newman (1988) presents a principled search technique for identifying the highest-scoring analysis from a set of complete analyses using weightings assigned to associations of word senses and semantic relations. In a variant of this approach (Schubert, 1986), competing analysis hypotheses are ranked during the course of a parse, based on weights assigned to considerations including attachment preference, sentential position, salience and so on. Only a small number of sub-analyses are retained at any stage to

control the combinatorial problem. The CLARE system (Alshawi *et al*., 1992) assigns preference measures to structural properties, semantic head collocations, particular resolutions of anaphora, and scoping permutations in 'quasi-logical form' representations during intermediate stages of semantic processing. Collecting data on semantic collocations in this type of approach is an open-ended, and usually subjective and labour-intensive manual process, although it may be possible to derive limited types of relevant data (semi-) automatically, for example from machine-readable dictionaries (e.g. Amsler, 1981; Alshawi, 1989; Copestake, 1990).

Recent theoretically-motivated approaches to the parse selection problem (e.g. Fujisaki *et al.*, 1989; Sharman, Jelinek & Mercer, 1990; Pereira & Schabes, 1992) have mostly used corpus-based probabilistic techniques, based on those pioneered in the field of speech recognition. The techniques associate probabilistic information with syntactic rules or parse configurations, on the assumption that the probabilities will model semantic and pragmatic constraints in the particular corpus being analysed. This area is surveyed in chapter 6.

1.4 Summary of the Thesis

1.4.1 Advances Reported in the Thesis

The research reported in this thesis includes new results and techniques in the areas of:

- *Unification for practical NL parsing:* a new unification algorithm is presented which, for certain types of system, is argued to be better-suited to practical NL parsing than existing alternatives.

- *Optimising bottom-up chart parsing:* optimisations to a bottom-up chart parser are described which considerably enhance its performance. Experimental results are given, showing that the parser implementation has significantly better practical performance than a comparable, state-of-the-art unification-based system.

- *Constructing LR parse tables for a large unification-based grammars:* a novel methodology for constructing LR parsers for unification grammars is described, various approaches to LR parse table construction are compared with respect to their suitability for large (NL) grammars, and a compact computational representation of tables for ambiguous grammars is described.

- *Non-deterministic LR parsing:* a context-free non-deterministic LR parsing algorithm is presented which has better worst-case time complexity than any previously reported in the

literature using the same approach, and a unification-based version of this algorithm is derived. In experiments, the performance of an implementation of this latter unification-based parser is shown to exceed both the chart parser and also the implementation of another LR-like $O(n^3)$ algorithm that has recently been proposed.

• *Probabilistic parsing:* a probabilistic system is described for parsing text taken from a specific corpus. New techniques implemented in the system include an incremental approach to semi-supervised training, a method of probabilistically scoring sub-analyses that is sensitive to context, the accurate manipulation of probabilities during the course of an LR parse, and the identification of the highest ranked analyses in a unification-based parse forest without a full exploration of the search space.

1.4.2 Overview of the Thesis

Chapter 2 starts with a summary of the ANLT metagrammatical formalism, a brief description of the ANLT grammar, a specification of the object grammar formalism, and of the unification operation defined on it. An algorithm that implements this operation is described and its suitability for NL parsing is compared with other unification algorithms in the literature.

Chapter 3 is about the ANLT chart parser. The algorithm is described, but an analysis of the theoretical worst-case time and space complexities of the parser gives what appear to be unpromising results. However, in an experiment comparing its practical performance with the ANLT grammar against the CLE and the CLARE grammar, the parser actually performs very well. Further experiments to augment the parser with left corner information to direct its search indicate that no performance improvements are possible in this area.

In chapter 4, the LR parsing technique is introduced, and the main types of parse table that are used to drive LR parsers are described. A technique for computing an LR table to drive an LR parser for a unification grammar is presented, and the application of this to the ANLT grammar is discussed. The practicality of various approaches to computing parse tables for large grammars are compared by applying them to the ANLT grammar.

Chapter 5 is concerned with non-deterministic LR parsing algorithms and their complexity. The best-known context free non-deterministic LR parsing algorithm (due to Tomita) is presented by means of an example, and results giving (exponential) lower bounds on its worst-case complexity are summarised. A parser which implements the algorithm is specified in detail, and then improved to give one with a time complexity of only $O(n^3)$. This algorithm is extended to parse unification grammars, and an implementation of it is compared with that of another recently proposed (also cubic complexity) LR-like parser, using the ANLT grammar.

Chapter 6 builds on the results and techniques of the previous chapters, describing a probabilistic LR-based parsing system. The system consists of an interactive incremental LR parser for use in training on a specific corpus, and a probabilistic LR parser which ranks in order of plausibility the typically large number of analyses assigned by the ANLT grammar, based on the statistics gathered during the training phase. Experiments parsing a corpus of noun definitions drawn from LDOCE are reported; the results are comparable to other systems described in the literature, but obtained on the basis of relatively little manual effort and a very much smaller training corpus.

The research described in the thesis is summarised in chapter 7, and directions for further work are discussed.

2. The ANLT Grammar Formalism and Unification Algorithm

The Alvey Natural Language Tools (ANLT) is a computer system for the morphological, syntactic and semantic analysis of a large subset of English. Chapter 1 gives an outline of the system. The components of the ANLT concerned with syntactic analysis are a wide-coverage sentence grammar of English, a grammar compiler, and a unification-based chart parser. These form the starting point for most of the research reported in chapters 4–6; some of the original aspects of the design and implementation of the latter two components are described later in this chapter and in the next.

2.1 The ANLT Metagrammatical Formalism

2.1.1 Basic Design

The ANLT was designed to be a computationally tractable implementation of GPSG. GPSG is a metagrammatical system in which several different types of rule interact to license local trees (i.e. trees with only a mother and daughters), the categories in the trees containing fully instantiated sets of feature name-value pairs. In the 'projection' from immediate dominance (ID) rules to fully instantiated local trees, features are subject to 'free' instantiation and the various rule types are assumed to act simultaneously. This process is conceptually complex, presents many implementational difficulties (Kilbury, 1986; Busemann & Hauenschild, 1988), and leads to time and space complexity results that suggest that the process is potentially impractical (Barton, Berwick & Ristad, 1987). Shieber (1986b) suggests that a less complex description of the grammatical theory embodied in the GPSG formalism can be obtained by a procedural mapping from GPSG grammars into a much simpler unification-based formalism. The ANLT follows this approach, and instead of generating fully instantiated local trees, the metagrammar is compiled into a set of partially instantiated phrase structure rules. This 'object grammar' is a syntactic variant of the Definite Clause Grammar formalism (Pereira & Warren, 1980), designed for parsing using term unification. This approach is computationally tractable, in theory and also in practice (sections 2.3.1, 3.2 and 3.3).

The ANLT metagrammatical formalism was also designed to be expressively powerful. GPSG contains a formally precise and prescriptive theory of natural language syntax which puts

strict constraints on the techniques a linguist can use when writing a grammar. Developing a wide-coverage grammar in GPSG (or indeed any other grammatical theory of this type) would be very difficult, since such theories are intended to make substantive claims about the behaviour of Natural Language, whereas wide-coverage grammar development requires a formalism that is maximally expressive (while remaining declarative and computationally tractable). Thus, in the ANLT, some of the restrictions GPSG places on grammars are relaxed, and principles (for example, those stating how feature values propagate between mothers and daughters) that are built into the theory are replaced by more flexible mechanisms that the grammarian can modify as required. This leads to a more expressive variant of the GPSG formalism which can be thought of more as a programming language specialised for writing grammars than as a syntactic theory (c.f. Shieber, 1984).

The dual attributes of expressive power and computational tractability offered by the ANLT derive from the separation of the metagrammar and the object grammar. This separation allows the latter to be designed so as to allow the implementation of a no-compromise fast parser, at the same time as the former is made into a formalism attractive to a grammarian. The complexity of the metagrammar formalism 'disappears' during compilation into the object grammar, leaving a well-defined and computationally tractable object grammar to be deployed at parse time. Although the interpretation of the metagrammatical rules is given by the procedure for their expansion and the semantics of the resulting object grammar, this interpretation can still be thought of in declarative terms.

The Grammar Development Environment (GDE; Carroll, Briscoe & Grover, 1991) is a software system which supports a grammarian in the task of writing a grammar in the ANLT metagrammatical formalism. The GDE is a large, mature, system containing facilities for the input of metagrammar rules and declarations, checking their well-formedness, interactive grammar browsing and editing, the monitoring of consistency between the various components of the metagrammar, and a (user-driven) generator for helping the grammarian to identify over-generation in the grammar under development. The GDE also contains a metagrammar compiler, taking the rules and declarations input, producing an object grammar that a parser (and the GDE generator) can directly use. The compiler is incremental, in the sense that only affected parts of the metagrammar are recompiled after any changes are made, and also 'lazy', in that recompilation takes place automatically and only when it is strictly required by subsequent operations. Boguraev *et al.* (1988) describe these aspects of the GDE more fully.

2.1.2 Brief Description of the Formalism

There are thirteen rule and declaration types in the ANLT metagrammatical formalism: feature, set, alias, category, extension and top declarations, immediate dominance (ID) and phrase

structure rules, propagation and default rules, metarules, linear precedence rules, and lexical entries.

Feature Declarations define the feature system used in the grammar. Each such declaration enumerates the values a specified feature may have. The feature system supported by the GDE is very similar to that assumed by several contemporary grammatical theories, but extends some of them, such as GPSG, in allowing features to take a variable value. The variable value ranges over the set of actual values as declared; a variable is notated as a name which starts with '`@`'. A feature may be defined as having a category as its value; such features may then have a variable as their value, but never an ordinary atomic value. Features are used to form categories, a category being an unordered collection of features, each feature in the category having a value. Feature / value pairs are separated by commas, and the whole category is enclosed by square brackets, e.g. `[N +, V −, BAR 2]`.

Set Declarations define groups of features which behave in the same manner with respect to feature value defaulting, feature value propagation and so forth. In rules in the metagrammar which perform these functions the name of the set may be used as a more readable way of referring to the whole collection of features, e.g. `NOMINALHEAD = {PLU, PER, CASE,...}`.

Alias Declarations define names that are considered equivalent to whole categories and feature complexes. These names can then be used in rules and other declarations to avoid having to write out in full all the feature / value pairs in a category. For example, the alias declarations `N2 = [N +, V −, BAR 2]` and `+PLU = [PLU +]` would allow the category `[N +, V −, BAR 2, PLU +, PER 3]` to be written as `N2[+PLU, PER 3]`.

Category Declarations define a category that conforms to a given pattern as consisting of a particular set of features. Category declarations are used to flesh out into more fully specified categories the partially specified categories which typically appear in ID rules and lexical entries. When a category declaration is applicable to a category forming part of an ID rule or lexical entry, those features in the category declaration which are not present in the rule or entry are added to it with a variable value. For example, the category declaration `Nominal: N2 => NOMINALHEAD` would ensure that all `N2` categories contained the features in the `NOMINALHEAD set`.

Extension Declarations specify that certain features, such as `SLASH` in GPSG, are not part of the 'basic' make-up of a category (in the way that, for instance, the feature `PER` might be in nominal categories). These 'extension' features, will therefore not appear in any category declaration; extension declarations do not affect the form of the object grammar, but act mainly as a

convenient reminder of feature usage for the grammar writer.

Top Declarations are used to indicate that when the parser returns the set of parses for a sentence or phrase, only those parses whose top node is an extension of one of the categories in the declarations should be retained, and that the rest should be ignored; thus the top declaration `S, N2` specifies that extensions of `S` and `N2` (i.e. sentences and noun phrases) are the only types of constituent that are recognised as being valid parses.

Immediate Dominance Rules encode permissible dominance relations in phrase structure rules. The orderings of the rule daughters that are possible are determined by the set of Linear Precedence Rules (see below) in the metagrammar. As an example, the first ID rule below states that a verb phrase may consist of a verb subcategorised for `NP`, and a noun phrase. Associated with this rule is an (optional) semantic translation: the GDE uses this to construct a semantic representation of a successfully parsed input. The representation is a (beta-reduced) composition of the semantics of the rule at each non-terminal node and of the lexical entry at each terminal node, in direct correspondence to the structure of the syntactic parse tree. The second rule contains an optional prepositional phrase, and the third, two daughters each marked as 'kleene plus' and able to dominate a contiguous sequence of one or more constituents of the appropriate category.

```
VP/TAKES_NP: VP --> H[SUBCAT NP], N2 :
    (lambda (x) (2 (lambda (y) (1 x y)))).
VP/SSR: VP --> H[SUBCAT SSR], ( P2[to] ), VP[TO].
N2/COORD: N2[PLU +, COORD +] --> ( N2[CONJ NULL] )+, ( N2[CONJ AND] )+
```

In a similar way to kleene plus, daughters marked with a kleene star ('`*`') may dominate a contiguous sequence of zero or more constituents. The head constituent of a rule is assumed to contain the feature `H` with value `+`. A daughter category containing the feature `NULL` (with any value) is treated during parsing as an empty, or phonologically null, category, hereafter referred to as a *gap*.

Phrase Structure (PS) Rules are similar to ID rules, but typically would be employed where it was wished to bypass linear precedence rules, for example in the following rule encoding 'heavy NP movement' in English:

```
Heavy_NP_Shift: VP --> H[SUBCAT NP_PP] P2 N2[+Heavy].
```

If the grammar contains linear precedence rules encoding the general rule that `N2`s precede `P2`s, then the rule in this example has to be a PS rule rather than an ID rule.

Propagation Rules define how features propagate between mother and daughter categories and between two or more daughter categories in ID and PS rules. The effect of propagation rules is to bind variables, instantiate values of features, or add new features with variable values to rules in the object grammar. Propagation rules can be used to encode particular feature propagation principles, such as the various versions of the Head Feature Convention proposed for GPSG. Propagation rules are stated in terms of ID or PS rule patterns which may contain variables over categories (`W`); for example, the following rule states that the mother inherits features from the head (noun) daughter in any ID rule which is nominal, contains a head, and which may optionally contain other daughters of any type:

```
Nominal: [+N, -V] --> [+H], W. F(0) = F(1), F in NOMINALHEAD.
```

Default Rules allow the grammar writer to assign default values for specified features in a particular ID (or PS) rule environment. A default rule will, however, have no effect if the specified features already have values (assigned perhaps initially in the original ID rule definition, or subsequently as a result of the application of a propagation rule or another default rule). These default rules replace Feature Specification Defaults in GPSG; because the rules assign values to features in the context of an ID or PS rule, their application can be accurately controlled, and thus the need for Feature Cooccurrence Restrictions to prevent the construction of 'illegal' categories is diminished. The `Accusative` default below states that an `N2` which is a sister to a verbal or prepositional lexical head will be accusative (if unspecified for any other `CASE` value):

```
Accusative: [] --> H[-N], N2. CASE(2) = ACC.
```

Metarules are a principled way of automatically and systematically enlarging the object grammar on the basis of the set of ID and PS rules initially written by the grammarian. A metarule consists of, on the LHS a pattern, and on the RHS the skeleton of a new rule; for every existing ID or PS rule that matches the metarule pattern, a new rule based on the skeleton is added to the object grammar. (In general, when a metarule is applied to an ID or PS rule, the semantics of the rule is changed. The GDE allows formulae to be associated with metarules, to specify the relationship between the semantics of the input ID rule and the semantics of the new rule that is created as the result of applying the metarule). The `Passive` metarule below adds a new VP rule without a noun phrase direct object (and with an optional 'by' prepositional phrase) for every basic VP rule with a non-predicative N2 daughter:

```
Passive: VP --> N2[-Prd], W. ==> VP[Pas] --> W, (P2[by]).
```

Linear Precedence (LP) Rules specify permissible precedence relations among daughter categories in ID rules. Thus an LP rule saying that an ID rule category that is specified for the feature SUBCAT will always occur before one that is not so specified could be written as:

```
LP1: [SUBCAT] < [~SUBCAT].
```

Lexical Entries may be defined as a quick and convenient way of adding to the set of words for which definitions are returned by the morphological analyser. A lexical entry declaration consists of the word itself, followed by one or more syntactic categories to be associated with the word. Semantic formulae may be paired with the syntactic categories. For example, a lexical entry for *he* might be defined as:

```
he: N[+PRO, PER 3, -PLU, CASE NOM, SUBCAT NULL] :
    (pro (the (x) (and (sg x) (male x)))).
```

2.2 The ANLT Grammar

The ANLT contains a wide-coverage sentence grammar of English (Grover, Carroll & Briscoe, 1993), written in the ANLT metagrammatical formalism. An earlier version of this grammar has been shown by Taylor, Grover & Briscoe (1989) to be capable of assigning the correct analysis to 96.8% of a corpus of 10,000 noun phrases extracted randomly from manually analysed corpora. The grammar contains of the order of 75 features, 450 ID and PS rules, 150 default and propagation rules, 15 metarules, and also set, alias, top and LP rule declarations.

The basic feature system of the ANLT grammar is very similar to GPSG, having four major categories: nominal, verbal, adjectival and prepositional. Each of these categories is specified for the features N and V, nominal categories being extensions of [N +, V −], verbal [N −, V +], adjectival [N +, V +], and prepositional [N −, V −]. Phrasal categories are distinguished from non-phrasal ones by the value of the feature BAR; phrasal categories being assigned the value 2, categories which are intermediate between lexical and phrasal having the value 1, and lexical the value 0. The feature SUBJ is used to distinguish sentential categories (which have value +) from verb phrases (having value −) within the set of [N −, V +, BAR 2] categories. None of the other (minor) categories, for instance determiners, particles and conjunctions, have the features N, V or BAR. For each major category there is a set of head features which must appear on all instances of that category, regardless of their BAR feature value. Further features must (or may) be associated only with certain instances of a major category, depending on the value of their BAR feature (or, on occasions, some other feature).

All lexical entries are specified for the feature SUBCAT. This feature is atomic-valued, and

has approximately 80 possible values. Around a quarter of these values are used to identify lexical items belonging to a minor category (for example, the complementiser *that* is defined simply as [SUBCAT THAT]). The remainder of the values are used to index the complementation patterns of major class lexical items, verbs having by far the largest number of possible patterns. The value NULL is used to indicate that a lexical item takes no complements, otherwise the general convention is that the value is made up from the category names of each of the complements, separated by the underscore character. In specific cases the value may also contain information about the semantic nature (e.g. the logical type) of, especially, verbs.

The feature AGR is used to enforce agreement, where applicable, between two or more constituents in a phrase structure rule. AGR is declared as being category-valued, but in the grammar the features in the categories that it can have as its value come from a small set which does not include any category-valued features.

Unbounded dependency constructions (UBCs) are analysed by propagating the preposed constituent through the parse tree as the value of the SLASH feature (the only category-valued feature apart from AGR), to link it with the gap that appears in the constituent's normal position. All non-lexical major categories contain the feature, rules in the grammar propagating it between mother and a single daughter; other daughters are marked [SLASH [NOSLASH +]] indicating that the daughter is not 'gapped'. Apart from the value [NOSLASH +], the grammar only allows as the value of SLASH restricted versions of noun, adjective and prepositional phrase categories, in which only a subset of the features that may appear in these categories actually do[7]. Figure 2.1 shows how the SLASH feature is used by the grammar in the analysis of the sentence *which abbot does he help?*. In this example, a subset of the features of the N2 constituent *which abbot* are percolated into the SLASH category value in the other VP daughter. An alternative treatment

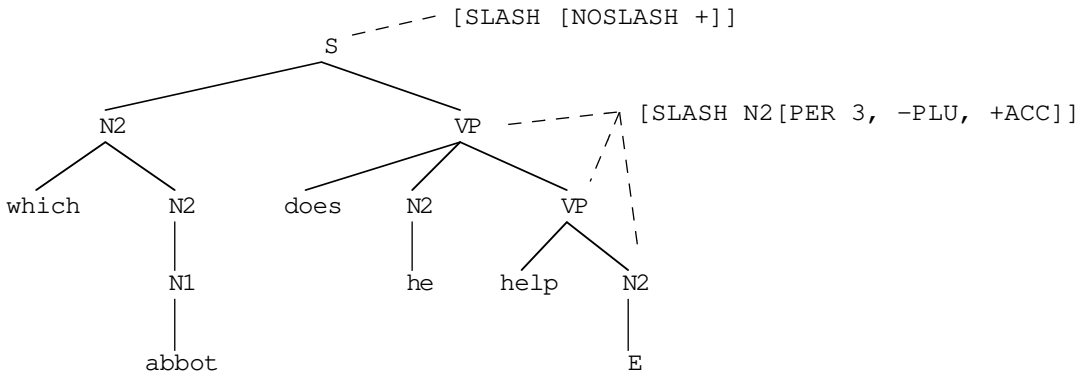

Figure 2.1. ANLT Analysis of the Sentence *which abbot does he help?*.

[7] In particular, the feature system is not fully recursive, in that neither SLASH nor AGR ever appear in a category value. This scheme has the advantage that the category set is finite (section 3.2.2 discusses the implications for parsing complexity), at the cost of ruling out some natural analyses for certain types of English constructions (Briscoe *et al.*, 1987b).

of UBCs involving 'gap threading' (e.g. Pereira & Shieber, 1987) would be more economical, but would make the statement of the semantics associated with ID rules more cumbersome.

The ANLT grammar employs different feature sets for each of the four major categories, so co-ordination rules have to be stated separately for each category. The analysis of co-ordination is broadly the same for each case, though, and is illustrated in figure 2.2. The feature `CONJ` appears on all conjuncts and encodes the type of conjunction that occurs with the conjunct. For example, in *the dog and the cat* the first conjunct (*the dog*) is `[CONJ NULL]` (contains no conjunction) whilst the second conjunct (*and the cat*) is `[CONJ AND]` (contains the conjunction *and*). Conjuncts that can iterate (as in *the mouse the dog and the cat*) are specified in the rule that licenses them as kleene plus categories, enabling them to occur from one up to an indefinite number of times.

2.3 The Object Grammar Formalism

2.3.1 Metagrammar Compilation

An ANLT metagrammar specifies a set of partially instantiated phrase structure rules, rather than fully instantiated local trees, as in GPSG. That is, rules are allowed to contain variable values for features and these variables are instantiated at parse time by unification. Restrictions on the instantiation of these variables can be made directly in the statement of ID (and PS) rules, or can be introduced indirectly when propagation rules apply to these ID rules. For example in the ANLT grammar, propagation rules applying to the ID rule for noun phrases which introduces a determiner and nominal phrase force agreement between the latter two categories, by binding together the values for the `PLU` feature in all of the categories:

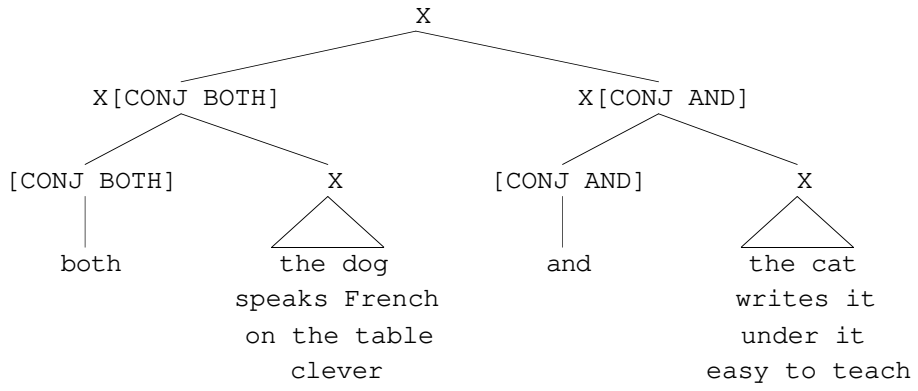

Figure 2.2. ANLT Analysis of Co-ordination.

```
N2+/DET:
   N2[+SPEC, PLU @2] --> DetN[AGR N2[PLU @2]], N2[-SPEC, PLU @2].
```

(Features irrelevant to this example are not shown). The AGR value in the lexical entry for the determiner *a* contains [PLU -], whilst the AGR value in the entry for *the* is unspecified for PLU. Therefore, the result of unifying the entry for *the* to the DetN category will not instantiate the variable @2, whereas the entry for *a* will instantiate it to -. However, unifying an N2 analysis of *boys* with the N2 daughter will instantiate all the instances of @2 in the rule to +. Therefore, the noun phrase * *a boys* will not be accepted because @2 cannot simultaneously be instantiated as both + and -.

The object grammar is produced from a metagrammar by expanding out aliases in ID (and PS) rules, applying propagation rules, default rules and category declarations (in that order), followed by metarules specified as applying to unlinearised rules, one-by-one in a predefined order. After each metarule has applied, propagation and default rules and category declarations are applied to any new ID rules and these are added to the original set before the application of the next metarule. Next, the resulting expanded set of rules is linearised according to the LP rules and any linear metarules are applied to the complete pool of PS rules. Lexical entries compatible with the object grammar are produced from metagrammar lexical entries (and also from the output of the ANLT morphological analyser) by deleting from the entries feature / value pairs in which the feature is a member of a distinguished set (the set MORPHOLOGYONLY), and then applying category declarations. The features H and NULL are removed from object grammar categories, and rules containing optional or kleene star categories are split into two or more having only normal or kleene plus categories.

Since the metagrammar must be finite in size (having been written down explicitly by a grammarian), and given the metagrammar compilation procedure described above, the object grammar must also be finite. If we assume that the object grammar contains $|R|$ rules, then the worst-case time complexity for the creation of a single object grammar rule is attained when the rule is created via the application of every single metarule. The compilation procedure effectively limits each metarule to apply only once during the derivation of a given rule, so the derivation is bounded above by

$$al \,.\, pdc \,.\, (pdc \,.\, m)^{|M|}$$

where al is the time to expand aliases and apply LP rules, pdc the time to apply propagation and default rules and category declarations, m the time to apply a single metarule, and $|M|$ is the number of metarules. Therefore, the time complexity of metagrammar compilation is

$$|R| \,.\, al \,.\, pdc \,.\, (pdc \,.\, m)^{|M|}$$

Expressing this complexity in terms of the number of object rules is justified if we consider the metagrammatical formalism to be essentially an abbreviatory device allowing a desired object grammar to be stated concisely in a form that is congenial to a grammarian, and which can be manipulated easily. In practice, compiling the ANLT grammar from scratch takes approximately 300 seconds of CPU time on a Sun Sparc ELC workstation, but after it has been compiled once, subsequent recompilations after changes have been made usually take very little time due to the caching that the GDE performs.

2.3.2 Format of the Object Grammar

The object grammar consists of a set of phrase structure rules, categories in the rules being unification-based, possibly containing variable values for features; these variables are instantiated at parse time by unification. Each category in the grammar is 'typed' based on the features that comprise it. The type is represented as a small integer, and in what follows is called the category *identifier* . One of the motivations behind the typing of categories is that since parsing with the object grammar uses term unification, if a pair of categories to be unified have different features then failure of the unification can be detected very quickly, since the category identifiers (types) will not be equal. Another motivation is that since identifiers are small integers, they can be used in a static index of the grammar and also to dynamically index the current parsing state (section 3.1); such indexing can result in a dramatic improvement in practical parsing performance.

Recent research into linguistic representations has proposed using typed feature structures with multiple inheritance (e.g. the HPSG grammar formalism; Pollard & Sag (1987)). These schemes assign a type to each feature structure, in a similar way to the ANLT object grammar, but additionally define a partial order on types. In contrast to the ANLT, in which two categories may unify only if their types are identical, in these theories, feature structures may unify only if their types are related by subsumption (Aït Kaci, 1984; Carpenter, Pollard & Franz, 1991).

Given a canonical ordering for features, feature / value pairs in object grammar categories can be arranged so that the features follow this order, and then if the identifiers of two categories are the same, unification of the categories can traverse each category, one pair at a time, the features at each step being guaranteed to be the same. Indeed, object grammar categories need contain only the feature values, not the features themselves, as long as the mapping from identifiers to features is retained so that categories in analyses from a parser (which will be expressed in terms of the object grammar) may be translated back into the feature / value pair format of the metagrammar. The ANLT adopts this approach, using the order in which features

are declared in the metagrammar as the canonical ordering[8]. Category identifiers are allocated starting at zero, and a discrimination tree of features is used in the translation of metagrammar categories to object grammar ones, a new branch being added to the tree when a category with a new feature make-up is encountered. A table pairing each identifier allocated to a list of features is also maintained to allow lookup in the reverse direction. A category is implemented as a one-dimensional array (vector) with the zeroth element the identifier, and the remainder of the elements the feature values. A rule daughter consists of a category paired with a flag to record whether the daughter was marked in the metagrammar as `[NULL +]` (i.e. a gap), or is kleene plus. As an example, the metagrammar ID rule `N2+/DET` quoted in section 2.3.1 above might be compiled into the object grammar rule in figure 2.3. The figure also sketches the elements of the identifier / feature lookup table that are relevant to this example.

The categories forming lexical entries are compiled into object grammar categories in a similar way to rule categories. Variables in object grammar categories which are unique within the lexical entry or rule being compiled are replaced by instances of the 'unnamed' variable, @. Such variables can be instantiated to any value without restriction, and behave the same as the variable written as '_' in Prolog (Clocksin & Mellish, 1984). All other variables, although they appear as @*name*, are represented internally as structured entities so that the implementation can quickly distinguish them from categories and atomic values, and also so they can be tagged efficiently with other information by storing the tag in a field inside the entity.

```
[[<category 5 + - \2 ... + ... @2 ...> N2+/DET]
 [<category 6 DETN + ... <category 7 + - \2 ... @2 ...> ...>]
 [<category 5 + - \2 ... - ... @2 ...>]]

<...
 [N V BAR ... SPEC ... PLU ...] ; 5
 [SUBCAT QUA ... AGR ...]       ; 6
 [N V BAR ... PLU ...]          ; 7
 ...>
```

Figure 2.3. An Object Grammar Rule and part of Identifier / Feature Lookup Table.

[8] The ordering can affect the efficiency of unification (depending on whether a feature whose value often discriminates between categories is one of the first or one of the last in the ordering), and also the efficacy of grammar indexing in the chart parser (chapter 3).

2.4 Unification in the ANLT system

2.4.1 Definition of Term Unification

The previous section described the object grammar formalism, but glossed over how (object grammar) categories behave under the operations performed on them by a parser. The most important of these operations is unification. Section 1.3 described the two basic types of unification: *term* and *graph*. The ANLT is designed to work efficiently only with term unification (although, since it is a general-purpose NL toolkit, it contains an option to use graph unification instead, described below in section 2.4.4). In the design of the ANLT object grammar formalism, term unification was chosen over graph unification since it enforces good linguistic and grammar writing discipline (which is particularly important during the development of a large grammar), and allows categories to be typed, giving the benefits discussed in the previous section.

Unification in the ANLT uses an approach based on *binding lists*[9]. A binding list (also called a *variable table*, or just *table* where the context is unambiguous) maps a set of variables to their current values, and consists of a list of *binding*s, each of which pairs a variable with a value. A variable is said to be *bound* if the binding list currently under consideration contains a binding for it, otherwise it is *unbound*. Given a binding list, the value of a variable is obtained by *dereferencing* it. The dereferencing operation looks for the first binding of the variable in the table, and if:

(1) there is no binding then the variable is returned, otherwise if

(2) the value in the binding is itself a variable, then the result of dereferencing this variable is returned, otherwise

(3) the value in the binding is returned.

Assigning a new value to a variable is accomplished by adding a new binding for the variable to the beginning of the binding list, effectively making any bindings for it that may already be present invisible. Each variable is tagged to indicate whether it is part of an object grammar category, or alternatively has been newly created at parse time.

Unification takes two categories as input, each with an associated binding list, and, if the operation succeeds, the two binding lists are updated to reflect any new values that variables in the two categories have been assigned by the operation. Unification always fails if the category identifiers of the two categories are different; if they are the same, the unification succeeds if

[9] For the first release of the ANLT, Phillips & Thompson (1987) implemented a unification module based on storing in tables values for variables; however these tables were interpreted in a different manner to that described here, and the unification algorithm was completely different. The algorithm described here can be implemented considerably more efficiently (and corrects several bugs).

every value in one category unifies with the corresponding value in the other category, given the binding lists. Before two values are unified, if they are variables, they are dereferenced with respect to the binding list of the category they are in. The result of unifying the two dereferenced values is defined in figure 2.4. Only the first category may be part of the object grammar, and so is allowed to contain the 'unnamed' variable, @. The result of unification is guaranteed to be correct only in the case where no variables in the first category occur in the second category, and *vice-versa*; a parser must ensure this by copying categories when necessary (section 2.4.3).

At the points marked [1] in the figure, a binding pairing the variable (*value2*) in the second category to the non-variable (*value1*) in the first is added to the second category's variable table, and additionally if the variable is part of the object grammar, the same binding is added to the first category's variable table. Conversely, at [2], a binding pairing *value1* to *value2* is added to the first category's table, and additionally if the variable *value1* has been newly created at parse time (as opposed to when the object grammar was created), the same binding is added to the second category's table. The final distinct case involving the creation of new bindings is at [3], where both dereferenced values are variables. In this situation, a binding for *value1* to *value2* is added to the first category's table, and a binding for *value2* to *value1* is added to the second category's table, either not being added if a binding with the values the other way around already exists in the table being added to (in order to avoid creating a circularity).

value 1	*value 2*	*result*
@	anything	success
category	category	result of unifying categories
	variable	success [1] (see text)
	other	failure
variable	category	success [2]
	variable	success [3]
	other	success [2]
otherwise	category	failure
	variable	success [1]
	other	success if value1 = value2 else failure

Figure 2.4. Result of Unifying two (Dereferenced) Values.

The following example should make it clear how the bindings created at points [1] and [3] in the definition of unification interact to ensure consistency between values for variables. (Bindings created at [2] work in a similar way to those created at [1], but apply in exactly the reverse situation). If the first and second categories are, respectively

```
<category ... @15 + @15>  <category ... @4000 @4000 ->
```

where `@15` is an object grammar variable and `@4000` is not, and the variable tables for each category are initially empty, then the tables after unifying the first pair of values (`@15` and `@4000`) in the two categories will be

```
[<binding @15 @4000>]      [<binding @4000 @15>]
```

following the actions specified for case [3]. (A binding is denoted by an object of the form `<binding variable value>`). When unifying the second pair of values (`+` and `@4000`), dereferencing `@4000` with respect to the second category's table gives `@15`, and the new tables that result are

```
[<binding @15 +>           [<binding @15 +>
 <binding @15 @4000>]       <binding @4000 @15>]
```

following case [1]. Unification of the third pair of values fails because the dereferenced value of the third (`@15`) value in the first category, +, is not equal to the third value in the second category, −. Unification of the two categories therefore fails.

A bottom-up parser would use the unification algorithm by passing as first category a rule daughter from the object grammar, and as second category, a constituent that had already been analysed (which could simply be a lexical entry). The variable table associated with the first category is termed the *rule variable table*, and that associated with the constituent, the *node variable table*. The former will contain values for variables appearing in the rule for which values have been assigned by virtue of appearing in a constituent, and thus are to be propagated up the analysis tree. Similarly, the latter will contain values that are to be propagated down the tree. Section 3.1 expands on this topic, and shows the tables in action by means of an example.

The result of the term unification of two categories C' and C'' can be defined formally as the most general category C, such that $C' \subseteq C$ and $C'' \subseteq C$, and where $\subseteq$ is the subsumption relationship. If $C(f)$ denotes the value associated with the feature f in the category C, and the domain of category C—$dom(C)$—is the set of features it contains, then $C \subseteq C'$ if and only if (Shieber, 1986a; Carpenter, Pollard & Franz, 1991):

- $type(C) = type(C')$,

- $C(l) \subseteq C'(l)$ for all $l \in dom(C)$, and
- $C'(p) = C'(q)$ for all paths p and q such that $C(p) = C(q)$.

An atomic value neither subsumes nor is subsumed by a distinct non-variable value, and a variable value subsumes all other values. The definition of unification in the ANLT system can be shown to be consistent with this formal definition. Firstly, the result of unification in the ANLT must have the same category identifier as the two inputs, thus satisfying the stipulation on types in the first clause above. The second clause ensures that the value of each feature in the result is subsumed by the corresponding values in both of the inputs; this is dealt with by the behaviour set out in figure 2.4 for unification of two deferenced values. The third clause says that all values which are identical in the inputs must be identical in the result; the case marked [3] in the figure deals with this.

2.4.2 The Unification Algorithm

This section describes the ANLT term unification algorithm which implements the behaviour specified above. The algorithm takes as input two categories and two variable tables, the first table associated with the first category, and the second table the second category. The result of a unification is either the flag `false` if the unification failed, or the flag `true` together with possibly updated versions of the two input variable tables.

When it encounters a variable, the unification algorithm has to dereference it. Figure 2.5 gives the definition of a procedure, `DEREFERENCE` , which does this. It takes a variable and a table and returns the dereferenced value of the variable with respect to the table, and also a flag indicating whether the variable is bound or not. When searching the table for bindings for a variable, at [1] in the procedure, the value is wanted from just the first binding found in the table for the variable; `IS-VARIABLE` is then called on such a value if one is found, and returns true if

```
DEREFERENCE(variable, table)
   if ∃ <binding variable, value> ∈ table then                    [1]
      if IS-VARIABLE(value) then return DEREFERENCE(value, table), true
      else return value, true
   else return variable, false

ADD-BINDING(variable, value, table)
   return [<binding variable, value> | table]
```

Figure 2.5. Variable Table Procedures used by the Unification Algorithm.

the value itself is a variable. Figure 2.5 also defines the procedure `ADD-BINDING` , used in the unification algorithm to add a binding to a binding list. In the algorithm, a list is denoted by being enclosed in square brackets, and the item following a '|' in a list itself being a list containing all the elements except the first.

The actual unification algorithm is given in figure 2.6. `UNIFY` first checks that the identifiers of the two categories being unified are equal (a category is assumed to have the format `<category identifier values>` where the *values* field is a one-dimensional array containing the values of the features in the category). If they are equal, successive pairs of values in the two categories are unified by `UNIFY-VALUES`, which returns either the flag `false` indicating unification failure, or `true` and new versions of the tables. The procedure `IS-NODE-VARIABLE` tests whether a variable is tagged as having been created during the course of a parse (rather than being part of the object grammar). Objects which are not relevant to processing are denoted by '_'.

2.4.3 Category Copying and Subsumption

A category can be viewed as a *feature structure*, which is a rooted, directed, acyclic graph[10] (DAG), whose arcs are labelled with feature names. Each arc points to another DAG, or to an atomic value. For example, figure 2.7 shows the category `[N +, V -, SUBCAT NULL, AGR [N +, V -, PLU @]]` expressed in graph notation. A variable in a category appears as an empty DAG. If two arcs point to the same node, there is a *re-entrancy*; re-entrancies can occur at *leaf nodes* (which represent atomic values or variables), or at nodes representing category values. (In a category, re-entrancy is notated by the sharing of variable values).

Category Copying

The definition of the unification operation (section 2.4.1) demands that the set of variables in the first category has a null intersection with the set in the second category. To satisfy this condition, when parsing bottom-up, once analyses have been found for all the daughter nodes of a particular rule and a complete constituent is being built whose top node is the rule mother, any variables in the mother category must be renamed so that they are distinct from any in the object grammar. The rule will probably have been written in such a way that some feature values are shared between the mother category and daughters by means of variables, and so any bindings for these

[10] Cyclic feature structures cannot be represented in the ANLT object grammar formalism (or in GPSG). It is possible to arrange to construct them during the course of a parse, but since they are generally seldom used, and in particular no GPSG-like theory makes use of them, the algorithms described here do not take them into account. This means that an occur-check (i.e. a check that unification is not being attempted between a variable and a category containing the variable) is not required in the unification algorithm.

variables that daughters have picked up through unification with lower-level shared constituents can be incorporated into the new constituent being built. This process of variable renaming and instantiation within the mother category must be performed on a copy of the category

```
UNIFY(<category id1, vals1_0...n>, <category id2, vals2>, vt1, vt2)
   if id1 = id2 then
      let unify := false
      for i from 0 to n
         unify, vt1, vt2 = UNIFY-VALUES(vals1_i, vals2_i, vt1, vt2)
         unless unify then return false
      return true, vt1, vt2
   else return false

UNIFY-VALUES(v1, v2, vt1, vt2)
   if IS-VARIABLE(v1) then v1 := DEREFERENCE(v1, vt1)
   if IS-VARIABLE(v2) then v2 := DEREFERENCE(v2, vt2)
   if v1 = @ or v1 = v2 then return true, vt1, vt2
   if v1 = <category _, _> then
      if v2 = <category _, _> then return UNIFY(v1, v2, vt1, vt2)
      else
         if IS-VARIABLE(v2) then
            unless IS-NODE-VARIABLE(v2) then
               vt1 := ADD-BINDING(v2, v1, vt1)
            return true, vt1, ADD-BINDING(v2, v1, vt2)
         else return false
   if v2 = <category _, _> then
      if IS-VARIABLE(v1) then
         if IS-NODE-VARIABLE(v1) then vt2 := ADD-BINDING(v1, v2, vt2)
         return true, ADD-BINDING(v1, v2, vt1), vt2
      else return false
   if IS-VARIABLE(v1) then
      if IS-VARIABLE(v2) then
         unless DEREFERENCE(v2, vt1) = v1 then
            vt1 := ADD-BINDING(v1, v2, vt1)
         unless DEREFERENCE(v1, vt2) = v2 then
            vt2 := ADD-BINDING(v2, v1, vt2)
         return true, vt1, vt2
      else
         if IS-NODE-VARIABLE(v1) then vt2 := ADD-BINDING(v1, v2, vt2)
         return true, ADD-BINDING(v1, v2, vt1), vt2
   if IS-VARIABLE(v2) then
      unless IS-NODE-VARIABLE(v2) then vt1 := ADD-BINDING(v2, v1, vt1)
      return true, vt1, ADD-BINDING(v2, v1, vt2)
   return false
```

Figure 2.6. The Term Unification Algorithm.

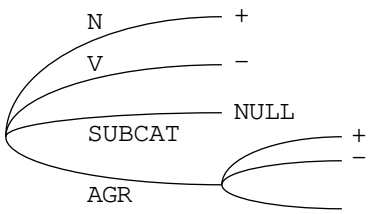

Figure 2.7. A Category Notated as a DAG.

otherwise the object grammar rule would be modified. Figure 2.8 gives the category copying algorithm used in the ANLT. The algorithm calls the procedure NEW-VARIABLE when it needs to obtain a new, unique, variable.

COPY-CATEGORY first creates a new category with the same identifier and number of values as the original category, calls COPY-VALUE on each old value, installs the new value returned into the new category, and finally returns the category. In COPY-VALUE , if the value is @ then the new value is simply a newly created variable, if a category then it is copied, and if it is not a variable then it is just returned. For variables, a local (initially empty) list of bindings is maintained to keep track of (object grammar) variables in the category being copied which are unbound in the main variable table; the test at [1] and updating of the list at [2] in figure 2.4

```
COPY-CATEGORY(<category id, vals0...n>, vt, local)
   let ncat := <category id, nvals0...n>
   for i from 0 to n
      nvalsi, local := COPY-VALUE(valsi, vt, local)
   return ncat, local

COPY-VALUE(val, vt, local)
   if val = @ then return NEW-VARIABLE(), local
   if val = <category _, _> then return COPY-CATEGORY(val, vt, local)
   if IS-VARIABLE(val) then
      if ∃<binding val, v> ∈ local then return v, local                [1]
      let v, bound := DEREFERENCE(val, vt)
      unless bound then
         if IS-NODE-VARIABLE(val) then return val, local
         else
            let nval := NEW-VARIABLE()
            return nval, [<binding val, nval> | local]                 [2]
      if v = <category _, _> then return COPY-CATEGORY(v, vt, local)
      return v, local
   return val, local
```

Figure 2.8. The Category Copying Algorithm.

make sure that multiple occurrences of such variables all end up with the same renamed value in the new category (i.e. that re-entrancies are preserved which involve the sharing of a node in the feature structure graph at which there is no value).

However, this copying algorithm does not preserve re-entrancies in the situation in which a node that *does* have a value in the feature structure graph is shared. When term unification is being used, a re-entrancy that is not preserved only makes a difference to the feature structure graph if the value at the shared node is a category, but even in this case the difference cannot affect the success or failure of subsequent unifications, and precisely the same results will be achieved, modulo non-leaf re-entrancies. (This is because the category value's feature make-up is fixed, and any further instantiation of the value at one of the former re-entrancies will have identical effects on the values at all of the other former re-entrancies). Not taking account of non-leaf re-entrancies minimises the space and time overheads of the algorithm, at the cost of having to copy every shared category once for every path to it. The ANLT grammar contains few such re-entrancies, and in practice this algorithm makes the right trade-off. Grammars with significantly more would probably benefit from the opposite approach, though. In these cases, a copy could be performed in time linear in the number of nodes in the feature structure, at the cost of extra space to record in each node in the source structure its counterpart in the copy.

Category Subsumption

Both the theoretical and the practical performance of a parser can be much improved by 'packing' constituents containing local ambiguity (as described in chapter 3). The situation in which this is possible is when a constituent is found whose top category is more specific than—is *subsumed* by—the top category of a constituent previously found. Roughly, for one category to subsume another, both must have the same identifier, and each value in the first category must subsume the corresponding value in the other; a variable subsumes anything, category value subsumption is checked recursively, and an atomic value subsumes another atomic value if the two are equal. (See section 2.4.1 above for a more formal definition).

The situation is more complicated when re-entrancies exist in the potentially subsuming category. If we consider a case where the first (potentially subsuming) and second categories are, respectively `<category ... @15 @15>` and `<category ... @4000 @4001>`, then the first category is in fact more specific than the second, since if the part shown of the first category becomes instantiated, the two instantiating values will be forced to be the same, whereas (in the absence of further constraints) the values will not have to be in the second. The first category in this example therefore does not subsume the second. Only re-entrancies involving leaves in the feature structure graph are affected by this complication, so the unfolding of (non-leaf) category

value re-entrancies carried out by COPY-CATEGORY cannot affect the situation.

To allow for a fast implementation, the subsumption algorithm that is implemented does not take account of leaf re-entrancies, and the algorithm assumes that the potentially subsuming category has no such re-entrancies (but does not check for this). However, when COPY-CATEGORY constructs a category which contains leaf re-entrancies, it flags it as re-entrant, and a parser must arrange to avoid subsequently testing this category for subsumption. This is a pragmatic approach to the use of a potentially expensive operation. Figure 2.9 gives the subsumption algorithm.

2.4.4 Extending the ANLT to Perform Graph Unification

In addition to providing good support for the use of the ANLT grammar, the ANLT is also intended to be a general-purpose toolkit for experimenting with other types of grammar; it therefore contains an option to use graph unification with the currently loaded metagrammar instead of term unification. However, the object grammar formalism was designed with a strong bias towards being used with term unification. Thus categories are represented as vectors whose size is fixed to be only the number of features actually present in the category. The feature structure resulting from the *graph* unification of two categories, though, must have at least as many features at the top level as either of the input categories, so in general, either every

```
SUBSUME(<category id1, vals1_0...n>, <category id2, vals2>, vt)
   if id1 = id2 then
      for i from 0 to n
         unless SUBSUME-VALUES(vals1_i, vals2_i, vt) then return false
      return true
   else return false

SUBSUME-VALUES(val1, val2, vt)
   if IS-VARIABLE(val1) then
      let v1 := DEREFERENCE(val1, vt)
      if IS-VARIABLE(v1) or v1 = val2 then return true
      else return false
   if val1 = <category _, _> then
      if val2 = <category _, _> then return SUBSUME(val1, val2, vt)
      else return false
   if val1 = val2 then return true
   return false
```

Figure 2.9. The (Approximate) Category Subsumption Algorithm.

unification must return a new category, or all categories must have a completely explicit and non-sparse representation, in which they contain (unused put potentially fillable) slots for any features not actually present in the category, in addition to the slots normally there for values of features that are present. The former solution, involving the creation of a new category as the result of a unification, does not fit in well with the technique of using binding-lists as the means of recording the result of a successful unification. For this reason, the ANLT implements the latter solution.

When the graph unification option is selected, all object grammar categories constructed during compilation are assigned the same identifier and are allocated a slot for every feature declared in the metagrammar. Each slot for a feature not actually present in the category is filled with a unique instance of a variable tagged as being 'optional'. The unification algorithm is augmented so that unifying an optional variable with a non-optional one results in a binding being created, associating the latter value with the former, in effect filling in the slot holding the optional variable and thus extending the category with the value. No action is taken when unifying two optional variables. The only other change required is that during category copying a new optional variable is generated each time one is encountered in the category being copied.

The graph unification technique just described becomes impractical if the metagrammar contains many features, since then category copying becomes very expensive. Traversing large categories containing mostly optional values during unification also degrades performance. However, in the implementation, the graph unification option seems to be quite practical for experimenting with small to medium-sized grammars.

2.4.5 Comparisons with other Unification Algorithms

Representations and Copying

Efficient unification algorithms have become a focus for recent research, most of it concentrated on graph unification. Tomabechi (1991:315) states that parsing systems which use this type of unification in conjunction with large, wide-coverage grammars often spend 85–90% of the total parse time performing unification. There are essentially two main approaches: firstly, representing feature structures as skeletal DAGs and unification results as versions of them together with *environments* binding previously uninstantiated DAG nodes to values (Pereira, 1985; Emele, 1991); and secondly, destructively updating (sometimes incrementally constructed copies of) portions of input feature structures to produce the results (e.g. Karttunen, 1986; Wroblewski, 1987; Kogure, 1990; Tomabechi, 1991). The first approach often results in less copying of DAGs, though Pereira's technique involves practical overheads (see below), and

Emele's requires a parser using the unification module to perform chronological backtracking. The second approach is prone to inefficiency due to unnecessary DAG copying, consuming both time and space (Godden, 1990). Two ways in which this can happen are:

- *early copying*: copies of the argument DAGs are made (at some stage, perhaps even before unification starts), but unification fails, and

- *over copying*: both argument DAGs are copied unnecessarily before a destructive operation.

Tomabechi's and Kogure's algorithms avoid both of these inefficiencies whenever possible, but Emele points out that these approaches are prone to performing *redundant copying* since the result of unification may not share structure with the input DAGs every possible case.

Term unification algorithms which are attuned to the requirements of parsing systems have been comparatively little studied. Emele's and Tomabechi's algorithms form part of implementations using typed feature formalisms: TFS (Zajac, 1992) and an HPSG-based (Pollard & Sag, 1987) analysis system for Japanese, respectively. However, it appears that they do not exploit the fact that in such formalisms (as is the case when term unification is used) every category contains only a fixed set of features. Term unification is one of the basic mechanisms in the Prolog programming language (Clocksin & Mellish, 1984), but parsers written in Prolog which use the built-in implementation of unification must either perform an excessive amount of copying, or restrict their control strategy and still have to copy in cases where it may be strictly unnecessary. In particular, such a parser must copy every category that is input to unification (both early and over copying) unless the parser is guaranteed to backtrack to a previous point, and must take a copy of the result of every unification (redundant copying) if the result may be required later and it is possible that the unification may be backtracked through.

The representation of feature structures that is used by the term unification algorithm described in this chapter has similarities to that of Pereira (1985); that is, a skeleton with updates (which instantiate variables) held in an environment. However, the way in which the representation is used is quite different. In Pereira's algorithm, environments have to be copied when a choice point in a derivation is encountered. (His use of 'virtual arrays' makes explicit copying unnecessary, but it is not clear that the overheads involved in maintaining these arrays are negligible compared to the rest of the work performed). In the algorithm described here, environments are extended non-destructively and so need not be copied, though skeletons and their associated environments are 'consolidated' to form a new skeleton with an empty environment when a complete constituent is found.

The algorithm is well-suited to parsing with traditional GPSG-like formalisms in which

phrase structure trees are the main device for linguistic representation. Categories need only be copied when new constituents are built, and successful unifications between rule daughters and existing constituents merely update tables keeping track of the values for variables. In contrast, most other unification algorithms end up building new structure in every category that results from a successful unification. Using variable tables helps to reduce the parser's space requirements since it is much more common for a unification to succeed than for a new constituent to be successfully analysed. Moreover, the space required to maintain a variable table is often much smaller than that consumed by a copy of a category, particularly when the table's space cost can be amortised over several alternative derivations which share portions of it. The unification technique is somewhat similar to a proposal of Pereira & Warren (1983), who suggest that in Earley Deduction, derived unit clauses could be copied and other derived clauses represented implicitly using variable binding environments in the manner of Boyer & Moore's (1972) approach to sharing term structures in theorem proving.

The algorithm will not work so well with formalisms based on theories such as UCG (Zeevat, Klein & Calder, 1987) and LFG (Kaplan & Bresnan, 1982) which represent linguistic objects as feature structures. In such formalisms, a constituent is typically represented by a single relatively large feature structure (rather than a phrase structure tree with a comparatively small top category), and so the fully recursive copying operation that builds a new constituent will be inefficient. Conversely, to work correctly with GPSG-like formalisms, the two main approaches mentioned at the beginning of this section would have to simulate phrase structure trees by means of feature structures (by explicitly storing the mother and daughter categories at every level). With a wide-coverage grammar, the depth and arity of the feature structures would typically be very much larger than would normally be the case, and so the amount of copying required to instantiate deeply-nested re-entrancies would be likely to be excessive. This dichotomy between types of formalism thus seems to dictate the approach to unification that must be taken in a practical parsing system.

Time Complexity

The complexity of unification algorithms can be linear, or almost so; Martelli & Montanari (1982) and Jaffar (1984) describe algorithms which attain $O(ack^{-1}(n) \cdot n)$ complexity, where n is the combined number of nodes and arcs in the terms being unified, and ack^{-1} is the (very slowly growing) inverse of Ackermann's function. Paterson & Wegman (1978) give a truly linear algorithm, but this entails a large constant overhead due to the use of a doubly-linked list representation. These algorithms destructively update their inputs in the same way as Prolog's built-in unification operation, so, unmodified, they would suffer from the drawbacks discussed

above with regard to Prolog.

The algorithm described in this chapter falls short of optimal complexity. It potentially visits every node in the input feature structures, and for each corresponding pair of nodes in the inputs, it dereferences variables at leaf nodes where applicable and performs a computation whose time is bounded by a constant. A feature structure containing f nodes can contain up to f–1 variables, so the unification of two feature structures whose combined number of nodes is n adds $O(n)$ entries to the variable tables of the two categories. The number of unifications performed in the course of deriving the input feature structures is bounded above by the maximum number of daughters of any rule in the grammar, say ρ, so the size of each of the variable tables is $O(n\rho)$. A variable may head a 'chain' of variable references, requiring multiple traversals of a table to arrive at a value which cannot be dereferenced further; such a chain may be of length equal to the size of the table, and therefore the cost of dereferencing a single variable is $O(n^2\rho^2)$. Summing over n nodes, the worst-case time complexity of the complete algorithm is $O(n^3\rho^2)$.

This sub-optimal complexity stems from a deliberate space-time trade-off whose purpose is to improve the average-case space behaviour of the algorithm by minimising the amount of category copying, as discussed above. An implementation of the algorithm forms part of the chart and LR-based parsing systems described in chapters 3, 5 and 6. Section 3.4 contains a practical comparison of the speed of the chart parser with respect to another parser which employs a different unification technique.

3. The ANLT Chart Parser

Since the ANLT is a general-purpose NL analysis toolkit, it includes as one of its main components an all-paths parser. The parser is unification-based, taking a grammar and lexicon in the ANLT metagrammatical formalism, and for a given input returning all the syntactic analyses licensed by the grammar. A major constraint on the design of the parser was that no expensive postprocessing should be necessary to compile the object grammar into a form suitable for use by the parser, since otherwise the processing needed after any modification to the grammar would slow down grammar development[11]. A further objective was that the parser's space and time requirements should be as small as possible, and that its performance with large wide-coverage grammars (in particular the ANLT grammar) should be good enough in practice to make it a usable tool for the analysis of realistic texts.

The exclusion of expensive postprocessing means that techniques such as LR parsing and related methods that construct tables or automata to control the steps taken by the parser (Aho, Sethi & Ullman, 1986) cannot be used. Even the extraction of unification-based top-down prediction information from a wide-coverage unification grammar (e.g. Moore & Dowding, 1991; Alshawi, 1992) is computationally expensive, requiring a transitive closure (or equivalent) operation using unification at each step to be applied over all the rules in the grammar. A more limited form of prediction based on (atomic) category types proves to be of little use in practice (see section 3.4) with such grammars. Simple top-down parsers can go into infinite loops with grammars that contain left-recursive sets of rules, and require grammar postprocessing to overcome this limitation. Left recursion in a grammar should be allowed: a bottom-up parser (which has no problems with right or left recursion) is therefore required.

The bottom-up active chart parser (see e.g. Kay, 1986) that forms part of (the fourth release of) the ANLT demonstrates that a practical parser can be designed and implemented to satisfy all of the objectives set out above[12]. This chapter describes the parser. The parser requires only a very small amount of object grammar postprocessing, and is fast enough to obviate the need for a second 'production' parser (or mode in which the grammar is subject to postprocessing) for use

[11] Incremental processing of the grammar at parse time might render certain types of compilation practical, but, for grammars written in modern unification-based formalisms, approaches assuming the preexistence of a CF component (e.g. Heering, Klint & Rekers, 1989) would have to be able to construct this component either cheaply or incrementally, perhaps by using restrictor sets (Shieber, 1985; but see section 4.2.1 for a discussion of some drawbacks to this technique).

[12] In the rest of this thesis, grammars are assumed to be acyclic, (i.e. not allowing infinite recursion stemming from sets of productions that can be applied to each other recursively without consuming any input), and not to contain infinite ambiguity (for example caused by rules licencing infinite sequences of gaps). Feature systems *may* be recursive, so the number of possible categories may be unbounded, but it is assumed that circular feature structures will not be created.

when the grammar is stable and parsing performance is important.

Phillips (1986) describes a bottom-up active chart parser that was an early version of the one included in the first release of the ANLT; however, the only techniques in common between Phillips' parser and the one described in this chapter are that both use rule indexing (though Phillips' approach to this is much cruder) and a simple recursive breadth-first control strategy without an agenda. The algorithms used in the (fourth release) ANLT parser described in this chapter are all substantially different, and indeed the parser uses several techniques and algorithms that are believed to be novel. The most important of these techniques are:

- local ambiguity packing and parse forest unpacking within a unification chart-parsing framework,
- efficient (static) rule indexing requiring minimal grammar postprocessing,
- efficient rule invocation, interleaving unification with rule retrieval, and
- avoiding copying (of variable binding lists) when unification fails.

Section 3.1 presents the parsing algorithm used, shows the processing steps it would perform on a simple example, and goes on to describe the ways in which critical parts of the algorithm were refined and optimised to give greatly improved performance. In section 3.2, the theoretical complexity of the algorithm is analysed (possibly the first such analysis of a bottom-up active chart parser to be reported). Section 3.3 gives the results of some experiments indicating that the parser meets the practical performance goals outlined above. Finally, section 3.4 describes an investigation into the possible performance benefits of integrating top-down prediction into a version of the parser, and discusses the conclusions (which run counter to those reported by some other researchers).

3.1 How the Parser Works

The parser takes as input a sequence of words. It identifies positions in the input by means of *vertices*, one of which occurs between every word and at each end, the vertices being numbered starting from zero at the left end of the input and increasing by 1 for each word. Each constituent or well-formed sequence of constituents found in the course of parsing is stored in a data-structure called an *edge*. An edge runs between a start and an end vertex, and may be either *active* or *inactive*. An inactive edge holds a single constituent in its *found* field. An active edge holds a list of constituents (corresponding to the first one or more daughters of a rule that have been successfully analysed) in its *found* field, the rest of the rule's daughters that have not yet been fully analysed in the edge's *needed* field, and the rule mother in the edge's *result* field. The

needed field of an inactive edge is empty and the *result* field is not used. Edges also contain two lists of variable bindings (see section 2.4.1), the first—the *rule variable table*, or *rvt*—containing values for variables being propagated up the analysis tree, the second—the *node variable table*, or *nvt*—values for variables being propagated down the tree. In summary, an edge is of the form `<edge needed found result rvt nvt start-vertex end-vertex>`.

A complete constituent is represented by a *node*, containing the syntactic category of the constituent, and a word if the node is a leaf otherwise a list of daughter nodes: i.e. `<node category node-list-or-word>`. An *analysis* consists of a node and a node variable table: i.e. `<analysis node nvt>`. In the examples and algorithms that follow, an object grammar rule is assumed to have the format `<rule mother-category daughter-category-list>`.

The parser works by taking each word in the input in order from left to right, finding lexical entries for the word, and for each syntactic category creating a node, and then an inactive edge with the node in its found field. As soon as an inactive edge is made, the parser checks to see if it covers all of the input: if so it creates an analysis record for it and adds it to the list of analyses found. The parser then tries to extend any active edge which ends at the same vertex that the inactive edge starts at by unifying the node category in the inactive edge's *found* field with the active edge's first *needed* category. If the unification succeeds, a copy is made of the active edge, the inactive edge category is added to the active edge's *found* list, the first category in the active edge's *needed* list is removed, and the end vertex is updated. If the *needed* list has now become empty, the edge is turned into an inactive one, with the *found* field containing a new node with its category taken from the result field. Whenever an inactive edge is made, the parser also finds any rules whose first daughters unify with the node category in the edge's *found* field, creating an active edge for each one with *found* field being a list containing just the node, *needed* field a list of the rule's daughters with the first removed, and *result* field the rule mother category. When all the words have been processed, the parser returns the list of analyses found.

3.1.1 An Example

This section illustrates the operation of the parser with a small example, parsing the sentence *John likes Mary*. The grammar contains just three rules, shown in figure 3.1, and in order to clarify the presentation, these rules are very much simpler than they would be in a realistic NL grammar. The lexicon has definitions for the words *John*, *Mary* and *likes*, also shown in figure 3.1. *John* and *Mary* are singular noun phrases, and *likes* is both a plural noun (as in *his likes and dislikes*) and a verb which requires a singular subject. Figure 3.2 shows the steps the parser goes through when parsing the example sentence, with categories notated in the way in which they appear in the metagrammar rather than as ANLT object grammar categories as they actually

```
1: S --> NP[PLU @p] VP[AGRPLU @p].
2: NP[PLU @p] --> Det N[PLU @p].
3: VP[AGRPLU @p] --> V[AGRPLU @p] NP[PLU @].

John: NP[PLU -].                    Mary: NP[PLU -].
likes: N[PLU +], V[AGRPLU -].
```

Figure 3.1. A Simple Example Grammar.

would be, again for clarity. In the figure, objects which are not relevant to processing are denoted by '_', and lists are enclosed in square brackets.

Note that when the parser finds the noun definition of *likes* at [1], it is unable to do anything with it: there are no active edges needing a noun as their next constituent, and in this grammar there are no rules with a noun as their first daughter. However, at [2], the verb definition of *likes* causes an active edge to be created by virtue of rule 3's first daughter, `V[AGRPLU @p]`. At [3], the active edge `e7` is found to have an empty *needed* list, so it is turned into an inactive edge, with the *found* field containing a new node whose top category, `VP[AGRPLU -]`, is the result of copying the category formerly in the *result* field (i.e. `VP[AGRPLU @p]`), and substituting variable values from the table `[<binding @p ->]`. At [4] the parser finds the following analysis for the sentence, but still carries on after finding it in case there are further ones.

```
<analysis
   <node S,
      [<node NP[PLU -], John>
       <node VP[AGRPLU -],
         [<node V[AGRPLU -], likes> <node NP[PLU -], Mary>]>]>,
   []>
```

3.1.2 The Basic Algorithm

Although the operation of the parser has already been described informally above, a more precise definition of the basic algorithm is needed at this point, to be able to go on to discuss refinements and optimisations to the algorithm which improve its practical performance. The algorithm is given in figure 3.3. In the algorithm, the operator '$\circ$' denotes append, and the object following a '|' in a list is itself a list which contains all the elements except the first.

The algorithm takes as input a sequence of words (assumed to be non-empty) and returns a list of analyses. Two global data-structures are used: a *chart* which stores active edges created

```
parse 'John likes Mary'
   create inactive edge e1 = <edge [], <node NP[PLU -], John>, _, [], [], 0, 1>)
      no active edges extendible by e1
      first daughter of rule 1 unifies with found of e1
         create active edge e2 = <edge [VP[AGRPLU @p]], [<node NP[PLU -], John>], S, [<binding @p ->], [], 0, 1>
         add e2 to chart
   create inactive edge e3 = <edge [], <node N[PLU +], likes>, _, [], [], 1, 2>                                          [1]
      no active edges extendible by e3
      no rules with first daughter unifying with found of e3
   create inactive edge e4 = <edge [], <node V[AGRPLU -], likes>, _, [], [], 1, 2>
      no active edges extendible by e4
      first daughter of rule 3 unifies with found of e4                                                                  [2]
         create active edge e5 = <edge [NP[PLU @]], [<node V[AGRPLU -], likes>], VP[AGRPLU @p], [<binding @p ->], [], 1, 2>
         add e5 to chart
   create inactive edge e6 = <edge [], <node NP[PLU -], Mary>, _, [], [], 2, 3>
      extend active edge e5 with e6 giving e7 =
           <edge [], [<node V[AGRPLU -], likes> <node NP[PLU -], Mary>], VP[AGRPLU @p], [<binding @p ->], [], 1, 3>
         turn e7 into inactive giving                                                                                   [3]
           <edge [], <node VP[AGRPLU -], [<node V[AGRPLU -], likes> <node NP[PLU -], Mary>]>, _, [], [], 1, 3>
         extend active edge e2 with e7 giving e8 =
              <edge [], [<node NP[PLU -], John>
                         <node VP[AGRPLU -], [<node V[AGRPLU -], likes> <node NP[PLU -], Mary>]>], S, [<binding @p ->], [], 0, 3>
           turn e8 into inactive giving
             <edge [],
               <node S,
                  [<node NP[PLU -], John> <node VP[AGRPLU -], [<node V[AGRPLU -], likes> <node NP[PLU -], Mary>]>]>,
               _, [], [], 0, 3>
           add <analysis <node S, ...>> to list of analyses found                                                       [4]
           no active edges extendible by e8
           no rules with first daughter unifying with found of e8
         no rules with first daughter unifying with found of e7
      first daughter of rule 1 unifies with found of e6
         create active edge e9 = <edge [VP[AGRPLU @p]], [<node NP[PLU -], Mary>], S, [<binding @p ->], [], 2, 3>
         add e9 to chart
```

Figure 3.2. Trace of Operations Performed when Parsing *John likes Mary*.

```
PARSE(w_1 ··· w_n)                                                    [1]
   let Chart_0...n := [], Analyses := []
   INSERT-TRACES(0)
   for i from 1 to n
      if i > 1 and Chart_i-1 = [] then return []
      for ∀c ∈ LEXICAL-ENTRIES(w_i)                                   [2]
         let node := <node COPY-CATEGORY(c, [], []), w_i>
         PROCESS-INACTIVE(<edge [], node, _, _, [], i-1, i>)
      INSERT-TRACES(i)
   return Analyses                                                    [3]

INSERT-TRACES(i)
   for ∀r = <rule m, []> ∈ RULES                                      [4]
      let node := <node COPY-CATEGORY(m, [], []), []>
      PROCESS-INACTIVE(<edge [], node, _, _, [], i, i>)

PROCESS-INACTIVE(e = <edge _, found = <node f, _> , _, _, nvt, start, end>)
   if start = 0 and end = n then
      Analyses := [<analysis found, nvt> | Analyses]
   for ∀a = <edge [n | nrest], _, _, arvt, anvt, _, _> ∈ Chart_start  [5]
      let unify, urvt, unvt := UNIFY(n, f, arvt, anvt ∘ nvt)
      if unify then
         EXTEND-ACTIVE(a, found, urvt, unvt, end)
   for ∀r = <rule m, ds = [d | _]> ∈ RULES                            [6]
      let unify, urvt, unvt := UNIFY(d, f, [], nvt)
      if unify then
         let a := <edge ds, [], m, _, _, start, _>                    [7]
         EXTEND-ACTIVE(a, found, urvt, unvt, end)

EXTEND-ACTIVE(a = <edge needed = [n | nrest], found, res, _, _, start, _>,
              node, rvt, nvt, end)
   let e := <edge nrest, found ∘ [node], res, rvt, nvt, start, end>   [8]
   FINISH-EDGE(e)                                                     [9]

FINISH-EDGE(e = <edge needed, found, res, rvt, _, _, end>)
   if needed = [] then
      found := <node COPY-CATEGORY(res, rvt, []), found>
      PROCESS-INACTIVE(e)
   else
      Chart_end := [e | Chart_end]                                    [10]
```

Figure 3.3. The ANLT Chart Parsing Algorithm—Basic Version.

during the course of parsing (keyed on the end vertex of the edges), and a list of analyses found. The analyses list and each chart element is initialised to empty. Parsing consists of invoking

INSERT-TRACES at each vertex to allow any type of gap to appear there, and for each word w_i, retrieving its lexical entries, for each entry creating a parse tree node holding the entry's syntactic category (copied as described in section 2.4.3), and calling PROCESS-INACTIVE on an edge ending at vertex i, starting at the previous vertex, containing the node. Finally the list of analyses found is returned. If, at any stage after the first word, there are no active edges ending at the previous vertex then there cannot be any analyses so processing can stop immediately.

When the procedure INSERT-TRACES is called at a vertex, for each rule which does not have any daughters it creates a parse tree node containing a copy of the rule's mother category, and no daughters. Then the procedure creates an edge containing the node, starting and ending at the given vertex, and passes the edge to PROCESS-INACTIVE. (If there were many daughter-less rules, the practical performance of the parser might suffer, but of course its theoretical worst-case complexity would not be affected).

PROCESS-INACTIVE takes an inactive edge representing a complete constituent, and if the edge spans the whole input, adds a record of the constituent and the current *node variable table* in the edge to the list of analyses found. Whether or not the edge represents a complete analysis, the procedure then tries at [5] to extend any active edges recorded in the chart as ending at the same vertex that the inactive edge starts at. It does this by attempting to unify (as described in section 2.4.2) the first *needed* category of each such active edge with the category of the constituent held in the inactive edge, taking into account the values for variables stored in the two edges. If the unification succeeds, EXTEND-ACTIVE is called to process the extended version of the edge. Finally, at [6], PROCESS-INACTIVE finds all rules whose first daughter unifies with the category of the inactive edge constituent, and for each one passes to EXTEND-ACTIVE a new active edge with all the rule daughters in the *needed* list and the mother as the *result*.

EXTEND-ACTIVE takes an active edge, a constituent that unifies with the first *needed* category in the edge, tables of variable values produced by the unification, and the end vertex of the constituent. It creates and passes to FINISH-EDGE a new edge containing the *needed* list of the active edge with the first element removed, the *found* list of the active edge with the constituent appended, the start vertex and *result* of the active edge, and the variable tables and end vertex passed to the procedure. FINISH-EDGE checks to see if the edge passed to it has any categories remaining in its *needed* list: if not, a complete constituent has been found. The actual constituent is a parse node with the mother being the category in the edge's *result* field, copied and with variable values substituted from the *rule variable table* (as described in section 2.4.3), and daughters being the edge's *found* list. PROCESS-INACTIVE is then invoked on the edge. On the other hand, if the edge passed to FINISH-EDGE still has *needed* categories, the edge is added to the chart keyed on its end vertex.

Extensions to Cover the Full Object Grammar Formalism

For ease of exposition, the parsing algorithm presented above is a simplified version which does not deal with the complete object grammar formalism, since it does not consider `[NULL +]` (null) or kleene plus categories in rules. However, only minor changes to the algorithm are required to deal with these two additional aspects of the formalism.

To cater for null categories in grammar rules, three cases have to be considered. The first is when all the daughters of a rule are null; such rules are considered as having no daughters at all, picked up at [4] in `INSERT-TRACES`, and processed in a similar way to ones with no daughters. The second case is when a rule contains at least one non-null category, but the first one or more categories are null; for such rules, the rule pattern at [6] in `PROCESS-INACTIVE` actually extracts as *d1* the first non-null daughter rather than strictly the first daughter. If the test succeeds, at [7] when a new edge is created, leading null categories in the rule's list of daughters are omitted from the *needed* list, and have gap nodes made for them in the edge's *found* list. The third case is when a rule contains one or more null categories following a non-null category; these null categories will become apparent at [8] in `EXTEND-ACTIVE`, after extending an active edge with a new constituent. If there is a next category in the *needed* list and it is null, it is removed from the list and a gap node containing the category added to the *found* list of the edge. This process of removal and addition is repeated until the first *needed* category is not null.

Kleene plus categories are dealt with by ensuring that whenever an active edge is extended with a new constituent and the edge category being filled in is kleene plus, an additional new active edge is created with the kleene plus category remaining in place at the beginning of the *needed* list, ready to accept further extending constituents. The following inserted after [9] in `EXTEND-ACTIVE` will do this.

```
if n is a kleene plus category then
   let e := <edge needed, found ∘ [node], res, rvt, nvt, start, end>
   FINISH-EDGE(e)
```

The additional edge created here is very similar to the one created in the existing version of `EXTEND-ACTIVE`, except that in this case the *needed* categories of the original active edge are carried through unchanged.

Object Grammar Indexing

The actual ANLT chart parser contains a number of refinements and optimisations which make it substantially more efficient than a straightforward implementation of the algorithm presented

above would be. This section describes the techniques that have been implemented to improve the efficiency of access to the grammar; however, the techniques are also potentially applicable to a wide class of other parsers.

The algorithm treats rules which have one or more daughters and those which have none in completely different ways. The ANLT implementation therefore partitions the object grammar rules into two sets based on this distinction, in a postprocessing stage before parsing is attempted. At parse time only daughter-less rules are considered in `INSERT-TRACES`, and conversely `PROCESS-INACTIVE` only considers ones which do have daughters.

When the algorithm requires the category of an inactive edge constituent to be unified with the first daughters of rules—at [6] in `PROCESS-INACTIVE` —all the rules in the grammar are potentially applicable, since top-down information is not used at any stage to predict which rules may be applied. For all but the smallest grammars, attempting the unification each time against every rule individually would be time-consuming, and also inefficient since the great majority of the unifications would fail[13]. Failed unifications constitute wasted effort since they contribute nothing to the output of the parser. The process is optimised in the ANLT chart parser by indexing the object grammar in a postprocessing stage, keying the index on each rule's first non-`[NULL +]` daughter. The index itself consists of a vector (one-dimensional array), the *i* th element of the vector giving access to all the rules whose first daughter category's object grammar category identifier is *i*. Each element of the vector is a discrimination tree (Charniak, Riesbeck & McDermott, 1980) keyed on the feature values of the first (non-null) daughter, the leaves of the trees containing rules. (The index is cheap to compute, and in addition is cached, only being recomputed if the grammar is modified.) So, for example, the object grammar rule shown in figure 3.4 is stored in the discrimination tree fragment shown in figure 3.5; some other rules with similar first daughters are also shown stored near it.

```
r1: VP[VFORM @v, FIN +, NEG +, ...] -->
       V[SUBCAT MODAL_BSE, VFORM @v, FIN +, INV -, NEG -, ...],
       [SUBCAT NOT], VP[VFORM BSE, FIN -, ...].

[[<category 1 - + \2 - @7 + + ...> r1]
 [<category 2 - + \0 MODAL_BSE @7 + - - ...>]
 [<category 3 NOT>]
 [<category 1 - + \2 - BSE - @ ...>]]
```

Figure 3.4. A Metagrammar ID Rule and Corresponding Object Grammar Rule.

[13] If a large proportion of unifications were to succeed in this situation, it would imply that there were few constraints on the constituents that could start the RHS of a rule.

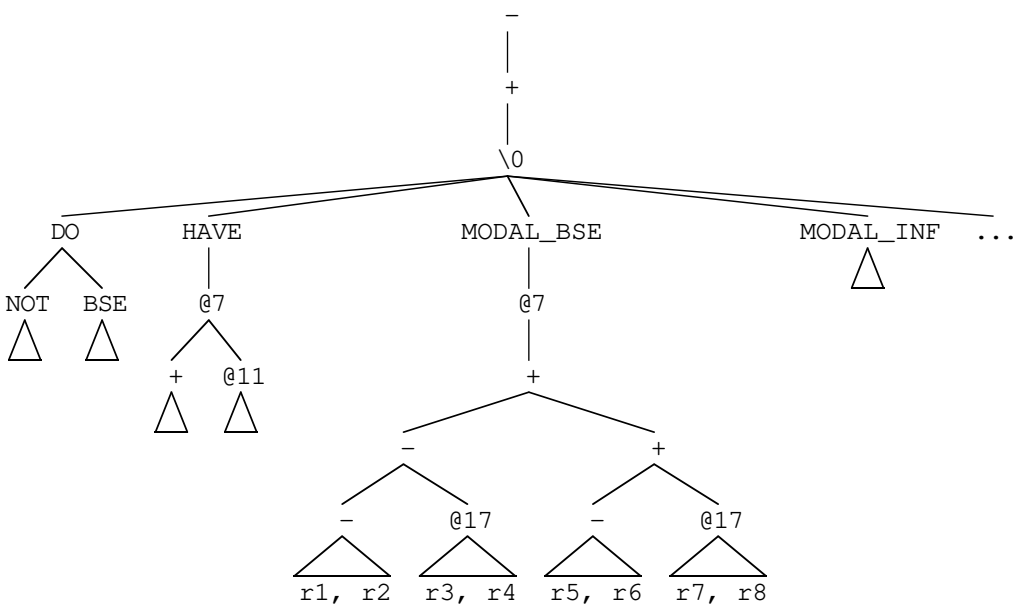

Figure 3.5. Fragment of a Rule Discrimination Tree.

When attempting to retrieve a set of rules whose first daughters unify with an inactive edge category, the first step is to identify the relevant vector element given the edge category's identifier. Then the tree in that element is searched starting from the top node, collecting up the rules in any leaf nodes reached, and traversing sub-branches only where the value at the start node of the branch matches the corresponding value in the category. The matching performed is actually unification of individual feature values, taking into account the values of variables previously bound further up the tree. The result of the process is a set of applicable rules, each rule paired with a set of variable bindings being the result of the unification of the rule's first daughter with the inactive edge category. So, given the rule discrimination tree in figure 3.5, an inactive edge category such as

```
<category 2 - + \0 MODAL_BSE NORM + @1000 - ...>
```

corresponding to a metagrammar category

```
V[SUBCAT MODAL_BSE, VFORM NORM, FIN +, INV @1000, NEG -, ...]
```

would cause the rules r1, r2, r5 and r6 to be retrieved, for each of the rules specifying the variable `@7` to be bound to `NORM` (thus making `NORM` the value of `VFORM` in the rules' first daughters), and the `@1000` value of the `INV` feature in the inactive edge category to be bound to – for rules r1 and r2, and + for rules r5 and r6.

This example illustrates some of the power of the interleaved unification and discrimination tree traversal technique for rule retrieval. Firstly, the grammar is partitioned into equivalence classes of rules, each class having a first daughter with the same category identifier. Only one rule class need be considered for any given inactive edge. Furthermore, whole groups of rules are ruled out on the basis of a single unification failure between two values; in the example the edge category SUBCAT value of MODAL_BSE means that none of the rules indexed under DO, HAVE, MODAL_INF and so forth need be looked at again. Another benefit of the technique is that each retrieved rule which shares a common initial portion of a path in the tree with another rule that is also returned in the result will automatically share the segments of variable binding lists created by the unifications performed while that path was being traversed. In the example above, all the retrieved rules share the path <- + \0 MODAL_BSE @7 +>, r1 and r2 additionally share the subsequent path <- ->, and r5 and r6 share <+ ->. Thus, the segment of the variable binding specification returned stating that the variable @7 is bound to NORM will be shared between all the results. In addition, the binding of @1000 to - will be shared between the r1 and r2 results, and @1000 to + between the r5 and r6 results.

For the technique to work effectively, whenever a given feature in a first daughter category in the object grammar has a variable as a value, the variable must as far as possible always be the same; if this is not the case the discrimination tree will branch unnecessarily. Variables are renamed as appropriate during the construction of the object grammar so as to meet this condition, insofar as is possible. A further efficiency consideration concerning variables in rules is that when a tree is being traversed, consistency of rule variable values need only be checked if one or more of the categories encoded in that particular tree is re-entrant (since otherwise there is no possibility that the variable has been encountered before). The retrieval process therefore inhibits the dereferencing of rule variables when searching trees not encoding any re-entrant categories.

The index for the ANLT object grammar partitions the rules quite finely: out of a total of approximately 750 rules, the maximum number occurring in a single leaf is 17, with more than half of the leaves containing only one or two rules. Practical experiments with parsing sentences using this grammar indicate that retrieving rules by performing full unification on all first rule daughters having the same category identifier as the inactive edge is around 50% slower than the discrimination tree method, and causes more than 35% extra dynamic storage to be allocated. A category discrimination tree of the type described here is cheap to construct, requiring only a single pass over all the rules in the object grammar (taking of the order of just one second of CPU time for the ANLT grammar on a DEC 3100 workstation). It would even be possible to eliminate the need for any grammar postprocessing by interleaving indexing with unification during parsing, but the additional implementational complexity involved would outweigh any benefits. The indexing technique is not restricted in applicability to the ANLT chart parser: it

could be used in any parser[14] which has a component using unification category information to drive or help control the process of rule invocation.

Chart and Parse Tree Indexing Optimisations

The parsing algorithm as described above maintains a chart of active edges in the form of a one-dimensional array, keyed on the end vertex of the edges. The algorithm uses data recorded in the chart only in `PROCESS-INACTIVE` at [5], when it has an inactive edge and wants to find all active edges ending where the inactive edge starts whose first *needed* category unifies with the category of the inactive edge. Evidently the unification will fail if the category identifiers are different, so to avoid even attempting the unification the ANLT chart parser employs a second index into the chart array: the category identifier of the first *needed* category in the active edge[15]. In practice, this results in a negligible increase in space required for the extra array dimension, but gives a speed-up of about 40%. Being able to use indexes dynamically in this way is a direct benefit of the category typing that in turn stems from the use of term unification.

The GDE contains an option to accept as analyses only those whose top categories are extensions of a set of specified categories. When this option is in force, at [1] in addition to the sequence of words, the parser is passed a list of category identifiers. Analyses are only returned (at [3]) if their top category's identifier is a member of the list. The parser also avoids the unnecessary work of unpacking trees (see the section below on local ambiguity packing) whose top category identifier means that the analysis would not be returned once it was unpacked.

Dynamic Storage Optimisations

The algorithm joins lists together in two situations. The first situation is when the algorithm builds the list of parse tree daughter nodes at [8]. Each node is *append*ed individually to the end of the current list; however, this operation is relatively expensive since the list must be traversed and copied every time. The standard alternative method involving the destructive modification of the end of the list is not an option here, since there could well be other daughter lists elsewhere which incorporated the first list, and these would end up being modified unintentionally. The

[14] Low-level programming issues, though, might mean that the technique could not be implementable efficiently in some computer languages. In Prolog, if a discrimination tree were stored in the system database, accessing it would involve an implicit copying operation, and the alternative of passing it between predicates as a logical variable might require explicit *copyterm* calls to be programmed to avoid variables becoming incorrectly bound (Alshawi, personal communication).

[15] The implementation ignores lexical entries (at [2] in the algorithm) which have a category identifier which is not present in the object grammar, since otherwise such entries could cause the parser to attempt to access a chart (or rule vector) element with a subscript greater than the size of the array.

append operation is unnecessary if nodes are added to the beginning rather than end of the list each time. Although analyses then end up reversed at every level, it is much more space- and time-efficient to reverse the lists of nodes only forming part of successful analyses when parsing is finished than to use *append* at every intermediate stage to build up the lists.

The second situation in which an *append* operation is called for is in the unification call just after [5] in `PROCESS-INACTIVE`, where the algorithm joins together two lists of variable bindings. Again, the operation at this point must not be destructive, and so if the unification fails, the space used in the copy of the first list that was created in order to do the append will have been wasted. With a wide-coverage grammar, most unifications performed by a unification-based parser fail (Tomabechi (1991) quotes a figure of over 80%), so a significant amount of space can be involved. The implementation produces results equivalent to the algorithm, but avoids the wasted space. It does this by saving the last pointer of the first list, and destructively concatenating the second list to the end of the first before invoking the unification. After the unification, the last pointer in the first list is restored (undoing the concatenation). If the unification failed no space was wasted in an append operation. If the unification succeeded, then any new bindings produced, a copy of the first list, and the second list are destructively concatenated together[16]. This approach to minimising the space consumed by failed unifications is similar in spirit to Wroblewski's (1987) unification technique involving updating feature structures destructively.

The algorithm can in some cases add active edges to the chart which have no chance of being incorporated into a complete analysis. In particular, ones ending at the last vertex when there are no daughter-less rules in the grammar can be ignored, since no inactive edge could possibly be created to extend them. This *shaper* test (Kuno, 1965) can be added at [10]:

```
if end < n or ∃ <rule _, []> ∈ RULES then
   Chart_end := [e | Chart_end]
```

More complex forms of shaper test taking into account kleene plus daughters and empty productions are possible at vertices near the end of the input string, in order to avoid creating active edges which could not form complete constituents before the end of the input was reached, but experiments show that there is little to be gained in this area with the ANLT implementation. The benefits of cutting down on active edge creation and extension in this way near the end of the sentence are only noticeable when parsing short sentences (of up to around five words), and sentences of this length are generally parsed very quickly anyway.

The refinements and optimisations described above have a significant practical impact on parsing performance. If they are all disabled, and rule access is implemented in a naive but

[16] If the tails of the two original lists share structure, this technique will not work since a circular binding list will be created and passed to the unification procedure. This situation seems to be rare, but is quick and easy to detect, and a simple copying append can be used instead.

nevertheless reasonable manner (by partitioning the object grammar into rules which have one or more daughters and those which have none, and filtering rule application by considering only those rules whose first daughters have the correct category identifier), experiments indicate that the parser is slowed down by a factor of more than three.

3.1.3 Dealing Efficiently with Ambiguity

Very often, if a natural language sentence is syntactically ambiguous, giving rise to multiple analyses, much of the structure in one analysis turns out also to occur in others. Global and local ambiguities result in different types of similarity between analyses.

In a sentence which contains global ambiguity, constituents near the leaves of the syntactic analyses (or indeed some of the lexical entries themselves) may be the same in all of the analyses, but the structure of the analyses at higher levels is differerent. For example, the verb phrase *saw the man in the park* is globally ambiguous, since the prepositional phrase (PP) *in the park* can modify either the man or the event of seeing him. However, the PP and all of the lexical entries are the same in both of the analyses that result; see figure 3.6. It is important that a parser represent such similarities compactly, otherwise its use of memory would increase linearly with the number of analyses, rendering the analysis of realistic inputs impractical, given that wide-coverage grammars and lexicons often assign hundreds or even thousands of analyses to them. A compact way of dealing with global ambiguities is to represent all of the constituents which are the same in different trees by a single object. Tomita (1985) calls this *subtree sharing*. As it stands, the chart parser described in the previous section performs subtree sharing, since each constituent found is represented only once: in a single inactive chart edge.

In the case of local ambiguity, one segment of the sentence is assigned distinct sub-analyses but the remainder of the sentence and also higher levels of the analyses receive the same

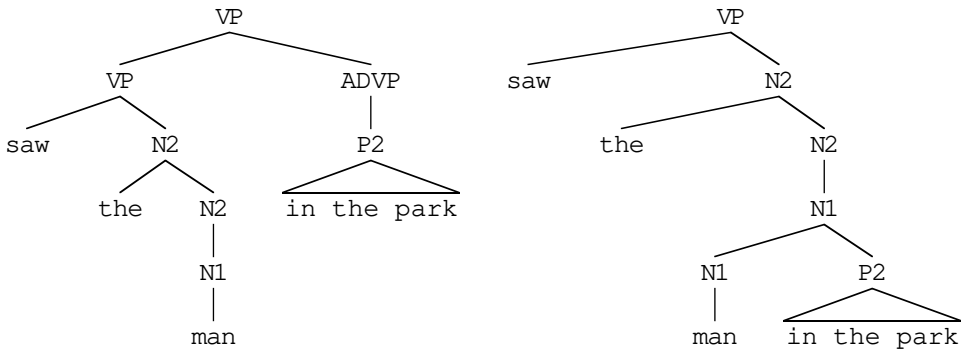

Figure 3.6. Analyses of the Verb Phrase *saw the man in the park*.

structure. For example, a sentence containing a compound nominal of more than two words (e.g. *winter holiday camp*) exhibits a local ambiguity since this phrase could be analysed in more than one way. Subtree sharing will represent compactly the common parts of analyses in the remainder of the sentence, but will not be applicable to any higher levels of the analyses even if their structure is the same, since the sub-analyses which represent the local ambiguity will all be distinct. The representational duplication that might result is absent in Earley's (1970) parsing algorithm, and so is not commented on; however, Tomita (1985) notes the problem and introduces an explicit technique to overcome it, called *local ambiguity packing*, in which subtrees with the same top category are 'packed' into a single tree node. Each packed node can be incorporated as a single entity into an analysis of the whole sentence, in exactly the same way as an ordinary node. With this technique, a single analysis encodes a *set* of (unpacked) analysis trees, and by analogy with 'parse tree', Tomita coins the term *parse forest* for a collection of analyses containing packed nodes and subtree sharing.

Tomita considers only context-free grammars without unification, for which the test for being able to pack nodes is category equality. When categories are feature structures, rather than atoms, the correct test for packing is category subsumption (Alshawi, 1992:142).

Local Ambiguity Packing in the Parser

The ability to detect local ambiguity and create packed nodes when appropriate is easily added to the chart parsing algorithm. When a new constituent (held in a new inactive edge) has just been found, at the beginning of PROCESS-INACTIVE, a check is made to see whether the constituent is subsumed by one previously found which spans the same vertices. If so, the node holding the new constituent is packed into the node holding the old one, and further processing of the edge is immediately halted. Otherwise, the new edge is recorded as having been found, by being stored in an array of inactive edges, and PROCESS-INACTIVE continues. To be able to pack one node inside another, a third field is added to node structures: a list of analysis records, each of which contains a packed node and an associated node variable table. The first part of PROCESS-INACTIVE now becomes:

```
PROCESS-INACTIVE(e = <edge _, found = <node f, _, _>, _, _, nvt,
                  start, end>)
   for ∀f = <edge _, <node ff, _, fp>, _, _, fnvt, _, _> ∈ Inact(start, end)
      if SUBSUME(ff, f, fnvt) then
         fp := [<analysis found, nvt> | fp]
         return
   Inact(start, end) := [e | Inact(start, end)]
```

The array of inactive edges, *Inact*, is initialised to empty before parsing commences. In the implementation, the category identifier of the found constituent is used as the second index into the array rather than the end vertex, since with the ANLT grammar and the type of sentences usually parsed (of up to around 25 words) the identifier turns out to discriminate between edges better than their end vertex does. Consequently, a check that end vertices tally is performed in an additional step before the subsumption test.

The algorithm that is used for testing category subsumption is given in section 2.4.3. As discussed there, for reasons of efficiency this subsumption algorithm is guaranteed to work correctly only in the case where the potentially subsuming category does not contain leaf re-entrancies. Therefore, edges that have a *found* constituent that do are not actually stored in the inactive edge array, in order to make them unavailable for subsumption checking (and thus not available for being packed into). A constituent is efficiently categorised as re-entrant or not (by `COPY-CATEGORY`) when it is created, and this fact is recorded in the otherwise unused *result* field of the inactive edge containing it.

If the parser finds a constituent that subsumes one already found covering the same input, the new constituent cannot be packed into the old one: the subsumption relationship must be the other way around for packing to happen. This situation results in sub-optimal space and time behaviour, in the sense that if the constituents had happened to have been encountered in the reverse order then they would have been packed, saving space, and further processing of the second constituent would have been halted. In this case, however, attempting any sort of packing and halting processing of the second (subsuming) constituent would render the parser incomplete, since the first more specific constituent might rule out analyses that the second less specific one would allow. This ordering problem is due to the fact that unification category subsumption is not a commutative operation; the problem is not encountered in basic context-free parsing since there packing is driven by atomic category equality which is commutative (i.e. if a and b are categories, $a = b$ if and only if $b = a$). However, empirical observations (with the ANLT grammar and test data of section 3.3 below) suggest that about 85% of categories that are candidates for packing are in fact equal, rather than one subsuming the other, and so the order in which they are found is not likely to have any significant practical impact on parser throughput.

Another complication of packing in unification grammar parsing is that some analyses that are implicitly encoded in a parse forest might not actually be legal ones. Each analysis in a parse forest gives rise to a single (unpacked) analysis containing only nodes which were not packed, or the first nodes found at points at which packing did occur. In addition to this analysis, further unpacked analyses can be exhaustively enumerated by choosing all combinations of subsumed nodes at each packed node. However, some of these combinations of packed nodes might not form part of valid analyses, since the values of features at packed nodes might be inconsistent

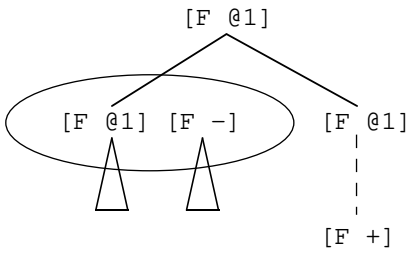

Figure 3.7. A Subtree in a Packed Parse Forest (ellipse encloses packed nodes).

with respect to feature values propagated from elsewhere in the tree. For example, in figure 3.7,which shows a single subtree in a parse forest, if the `[F +]` constituent is incorporated into the forest at the position indicated, the packed node with root category `[F −]` in this subtree may not form part of a complete parse.

Unpacking Packed Parse Forests

Semantic interpretation and further stages of processing after syntactic parsing could take a parse forest as input, as in some early versions of the CLE (Alshawi, 1992). However, to simplify the subsequent processing so that it does not have to be aware that some sub-analyses implicit in the forest may in fact not form part of a complete analysis (due to non-local inconsistencies in feature values as discussed above), the chart parser, by default, always fully unpacks parse forests into individual parse trees as the last stage in the syntactic analysis of a sentence. The parse forest itself cannot merely be pruned to remove sub-analyses affected by such inconsistencies (c.f. Vijay-Shanker & Weir's (1990) pruning of spurious ambiguity in parse forests with Combinatory Categorial Grammar), since a sub-analysis that is a candidate for pruning may form part of a complete analysis in another context.

The unpacking algorithm traverses the parse forest top-down starting at the root node, recursively unpacking packed sub-analyses when it encounters them at a node, and returns a list of the analyses with that node as their root. The root category of each packed sub-analysis is unified with its subsuming category at that node to ensure consistency of non-local feature values. If the unification fails, the sub-analysis is not returned. Feature values percolating from higher up in the parse forest are incorporated into packed analyses simply by adding to the beginning of the variable table of the packed analysis the bindings in the higher level variable table that were created after the analysis was packed.

3.2 Complexity of the Parser

3.2.1 Packing and Complexity

Local ambiguity packing is more than just a technique for saving space when parsing. For an arbitrary context-free grammar, computing all the analyses and returning them in the form of a set of trees not containing packed nodes is exponential in time as well as space with respect to the length of the input. Consider the CF grammar in figure 3.8. With this grammar, the number of analyses for a string of `a`'s of length *n* is given by the *n*th Catalan number (Church & Patil, 1982), so for example a string of length 3 has 2 analyses, 4 has 5, 5 has 14, 9 has 1,430, and so on. The Catalan numbers increase in an exponential manner. All the analyses are different, so at least the top node of every analysis must be distinct and not subject to subtree sharing with any other analysis. Hence, the space required to store all of the analyses is exponential in the length of the input. To be able to return all of the analyses, a parser must at least enumerate them, so the parser's time complexity cannot be less than exponential. Thus, with an arbitrary grammar, the worst-case time and space complexity of a parser which does not perform some form of local ambiguity packing is at least exponential. Computing a parse forest and then unpacking it to give a set of trees results in the same complexity, since there could be an exponential number of parses after unpacking. Therefore, when considering the complexity of parsers the usual measure is the time and space complexity of the computation just up to the point of producing a parse forest, with an arbitrary grammar and input of length n[17].

In a parser which performs subtree sharing, all of the constituents which are the same in different analyses are represented by a single object. With a grammar whose rules are at most binary branching—but not necessarily in Chomsky Normal Form (CNF; see chapter 1)—a parser which uses local ambiguity packing and this form of subtree sharing can represent an arbitrary parse forest in cubic space, since the *grammar of parses* (Billot & Lang, 1989) has at most two symbols on the right-hand side of its rules. However, with an arbitrary grammar (with no *a priori* bound on the maximum number of daughters in a single rule), the space complexity of such a parser is exponential.

```
S --> S S.
S --> a.
```

Figure 3.8. An Ambiguous Grammar.

[17] Constructing a parse forest can be much easier than recognition, and in these cases the complexity of extracting a parse from the forest must also be considered. For CF grammars, extracting a parse from an Earley parse forest is linear, but in the case of Indexed Grammars (Aho, 1968), for example, a parse forest can be constructed in cubic time but recognition is exponential (Lang, 1992).

The chart parser described in this chapter performs subtree sharing, since each constituent found is represented only once: in a single inactive chart edge. However, this is not the only sharing that takes place: in addition, *sequences* of constituents are shared between analyses in cases where a single active edge covering more than one constituent is used in the derivation of two or more analyses. The shared sequences correspond to the leftmost daughters of rules, since the parser works from left-to-right (but a similar type of sharing could take place with rightmost daughters in, for example, a right-to-left chart parser or a non-deterministic LR parser: see chapter 5). In this scheme, even if a grammar contains rules with more than two daughters, Billot & Lang's grammar of parses still has at most two symbols on the right-hand side of its rules. This is because for a constituent with n sub-analyses, the sequence comprising the first $n-1$ sub-analyses can be considered to be a single node since that sequence is a shared entity. Thus, a parser which uses local ambiguity packing, and which shares both subtrees and sequences of subtrees has $O(n^3)$ worst-case space complexity for an arbitrary grammar. The chart parser therefore has cubic space complexity.

For a unification grammar, the size of the corresponding grammar of parses is related to the number of categories implicit in the grammar, $|C|$, of which there could be very many. If a grammar uses a fully recursive feature system (section 2.2), $|C|$ is actually infinite. However, Briscoe *et al.* (1987b) point out that if a grammar restricts the depth of nesting of category-valued features, $|C|$ is finite. The ANLT grammar and also the GPSG grammar of Gazdar *et al.* (1985) obey this restriction, nesting these features to a maximum depth of two[18], but nonetheless their category sets are still extremely large[19].

3.2.2 Time Complexity of a CF Variant of the Parser

To analyse the time complexity of the ANLT chart parser, it is easiest first to consider a variant of the algorithm that only accepts grammars which are context-free with atomic categories, and which do not contain any rules with kleene plus categories. In this variant, an upper bound on the number of inactive edges between any pair of vertices is just the number of categories in the grammar, i.e.

$$|i_{j,k}| = |C| \tag{3.1}$$

since if two nodes span the same segment of the input and have the same top category then they will be held as a packed node in a single inactive edge. To derive a similar upper bound for the

[18] In these grammars, the features `SLASH` and `AGR` are the only category-valued features, and their values never contain embedded occurrences of these features.

[19] Barton, Berwick & Ristad (1987:221) state that GPSG licences more than 10^{775} distinct syntactic categories.

number of active edges, consider a single rule with ρ daughters, which gives rise to a set of active edges ending at vertex k. There are at most

$$(k + 1) + (k + 1)\,(k + 2) / 2 + \ldots + O(k^{\rho-1}) = O(k^{\rho-1}) \tag{3.2}$$

such active edges, equivalent to the number of ways of partitioning a line containing k intervals into a number of segments, from two (corresponding to an initial segment from vertex zero up to the edge's start vertex and a segment from the start to the end vertex covered by the rule's first daughter only) up to ρ (the same initial segment plus a segment for each of the rule's first ρ-1 daughters)[20]. Multiplying by the number of rules in the grammar, an upper bound on the total number of active edges starting at any vertex and ending at vertex k is

$$|a_k| = O(|R|\, k^{\rho-1}) \tag{3.3}$$

where $|R|$ is the number of rules in the grammar, and ρ is the maximum number of daughters in any rule in the grammar.

For an input of length n, the parsing algorithm iterates over each vertex from zero to n. At each vertex k, the time taken by the algorithm can be split up into three parts: that required to extend existing active edges, to pack new constituents found, and to create new edges ending at that vertex having matched rule first daughters with new constituents. At vertex k, the algorithm tries to extend all active edges which end at vertices from zero to k with inactive edges starting where the active edges end and which themselves end at vertex k. Therefore the number of tests required to extend active edges is

$$|a_k|\,|i_{k,k}| + |a_{k\text{-}1}|\,|i_{k\text{-}1,\,k}| + \ldots + |a_0|\,|i_{0,k}| = O((k + 1)\,|C|\,|R|\,k^{\rho-1}) = O(|C|\,|R|\,k^{\rho})$$

(using the bounds on the numbers of active and inactive edges given in expressions 3.1 and 3.3). Each test is just for atomic category equality, taking unit time.

A new constituent can be created from a lexical entry for a word, a rule that just expands to a gap, as a result of filling the final needed category of an active edge, and by the application of a sequence of one or more unary rules to another new constituent. Applying all possible unary rules to a new constituent introduces a factor equal to the number of unary rules in the grammar (in other words up to $|R|$) since each unary rule can be applied at most once in a sequence of only unary rule applications (otherwise an infinite derivation would result). There can only be up to $|i_{k-1,k}|$ constituents created from lexical entries, and $|i_{k,k}|$ from gap rules at vertex k, so extending active edges and applying unary rules to all those forming new constituents is the dominant

[20] The maximum number of segments in the line is ρ since the last daughter can never be assigned a segment, otherwise the edge would be inactive, not active. A segment is allowed to be of zero length, in which case it represents a gap.

bound. This bound is $|R|$ multiplied by the time to extend active edges, giving $O(|C|\ |R|^2\ k^\rho)$. Checking for being able to pack each new constituent takes unit time (given an indexing scheme based on category and start and end vertices), thus the complexity of all the packing operations also has this bound. Creating new edges having matched rule first daughters is bounded by the number of rules in the grammar, $|R|$, multiplied by the number of new constituents found, giving $O(|C|\ |R|^3\ k^\rho)$.

Summing the times for the three parts of the algorithm over an input of length n gives

$$\sum_{k=0}^{n} 2\ O(|C|\ |R|^2\ k^\rho) + O(|C|\ |R|^3\ k^\rho) = O(|C|\ |R|^3\ n^{\rho+1})$$

as the total time complexity. Considering grammars in which each rule has at most two daughters (i.e. grammars for which ρ is 2), the time complexity of the algorithm is thus $O(n^3)$. This bound is as good as that of the CKY algorithm (Kasami, 1965; Younger, 1967), which also uses a bottom-up tabular approach. CKY, though, is more restrictive in that it does more than just limit each rule in a grammar to having at most two daughters: it requires the grammar to be in CNF. CNF forces natural language grammars to be written in an unnatural way, since rules introducing gaps and unary rules expanding non-terminals are often convenient but are not allowed. It is possible to translate an arbitrary CF grammar into CNF with the help of additional, dummy non-terminals (Chomsky, 1959). However, there is no guarantee that for any particular grammar, the CNF version will not contain a great many more rules than the original grammar. When each rule has at most two daughters, the chart parsing algorithm presented here has the same time complexity with respect to input length as CKY but does not have its CNF requirement, so in some cases it might make a more suitable alternative.

Even the restriction of having at most two daughters in any rule is undesirable from the point of view of the grammar writer. Lifting the restriction results in a time bound that is worse than cubic. However, putting theoretical complexity issues aside, it is not clear that under all conditions a grammar containing a few rules with more than two daughters will take longer to parse sentences than the same grammar transformed so that all rules are at most binary branching. (Approximately 90% of the rules in the ANLT grammar are unary or binary branching). Investigating the trade-off between worst-case complexity and increase in transformed grammar size would be a useful area for further research.

3.2.3 Time Complexity of the Unification-based Parser

Most of the above analysis for the complexity of a context-free variant of the parser holds when atomic categories are replaced by unification-based categories. However, packing a new

constituent cannot now be assumed to be possible in unit time; each check to see if a constituent may be packed may require up to $|C|$ tests, since subsumption checking on unification categories cannot be indexed as completely as can equality tests on atomic categories (which need only a single array access and boolean test). Assuming unification can be accomplished in bounded time (the bound is independent of $|C|$, $|R|$, and input length; see section 2.4.5), the unification grammar time complexity is $O(|C|^2 |R|^3 n^{\rho+1})$. In a similar way to the unification-based space result, the dependency on the number of categories makes the parsing algorithm appear to be intractable.

The complexity results for the parser that are derived above take no account of rules which contain kleene plus categories. Any such rule can potentially consume a sequence of sub-analyses limited in number only by the length of the input, and so it could be expected that the number of active edges a rule with kleene plus daughters could give rise to could exceed the bound for non-kleene rules given in expression 3.2. This is indeed the case; however, deriving an upper bound for the number of active edges for these rules is more difficult than for non-kleene rules. Before deriving the bound, observe that no legal grammar can contain a kleene plus category able to consume a constituent containing just a gap, otherwise invocation of a rule containing the category would not terminate since it would be able to dominate an infinite sequence of gaps. Therefore, using a method similar to that used to derive expression 3.2, the number of active edges that a rule, all of whose daughters are kleene plus, could give rise to which end at vertex k is the sum of the number of ways of partitioning a line containing k intervals into from 1 to k+1 segments (such that all segments except the first are of length strictly greater than zero), and where all sub-sequences of segments (excluding the first) are allocated in all possible ways to between 1 and the number of daughters in the rule. After some algebraic manipulation, this sum approximates to

$$(k + (k - 1)k / 2 + ... + O(k^{k-1}))\, 2^{k-1} = O(k^{k-1}).$$

The other intermediate results are derivable in the same way as before, so the total time complexity of the unification parser for the full ANLT formalism is $O(|C|^2 |R|^3 n^n)$.

Thus, allowing a grammar to contain rules with kleene plus daughters has changed the theoretical worst-case behaviour of the algorithm from polynomial to exponential. To retain the polynomial time bound, grammars containing kleene rules could be transformed into ones without any kleene categories with the addition of extra rules set up to produce recursive tree structures instead of the iterated flat tree structure created by the kleene categories. This technique is described in chapter 4, in connection with unification-based LR parsing. However, applying this technique to the ANLT grammar actually slows the chart parser down. It appears that the way the kleene categories in the grammar are used makes little impact on the performance of the parser; in fact, the increased overheads in rule invocation and structure

building associated with the new rules introduced to remove the kleene categories slow it down more.

Although the ANLT grammar is large, with around 750 rules ($|R| = 750$), and the size of its category set ($|C|$) is very large, in practice the performance of the chart parsing implementation is really quite good (see the next section), even for relatively long sentences. It therefore seems likely that grammars of natural languages do not bring the algorithm anywhere near the worst-case complexity.

3.3 Practical Performance of the Parser

As discussed in chapter 1, a complete NL analysis system contains more than just a parser, and the latter may contribute only of the order of 25% to the total processing time. However, with wide-coverage grammars and realistic text, the syntactic analysis of a single sentence of moderate length on current computer hardware can take several seconds (Briscoe & Carroll, 1991:27), and this time necessarily increases at least linearly with the length of the sentence. The ANLT chart parser was designed and implemented with practical performance with large, wide-coverage grammars a high priority; to assess how well the parser meets this goal, and how its performance compares with the state of the art in unification-based parsing, an experiment was carried out to measure the performance of the ANLT parser and of the CLE parser (in the version which forms part of the CLARE2.5 system (Alshawi, 1992; Alshawi *et al.*, 1992)). A corpus of approximately 1500 sentences (produced by the linguist who developed the ANLT grammar) is supplied as part of the ANLT, implicitly defining the types of construction the grammar is intended to cover. The corpus is used to check for any adverse effects on coverage when the grammar is modified during grammar development. For the experiment, an initial sample of 175 representative sentences was extracted from the corpus.

The experiment consisted of parsing each set of sentences in the sample, firstly with the CLARE2.5 parser, grammar and lexicon, secondly with the ANLT parser, grammar and lexicon, and thirdly with the CLE parser, having replaced the built-in grammar and lexicon by versions of the ANLT object grammar and ANLT lexical entries for the words in the sample translated into the CLE formalism. The reverse of the latter configuration, in which the CLARE2.5 grammar is loaded into the ANLT system, is not possible since some central rules in the grammar contain sequences of daughters specified by a single 'list' variable. This aspect of the CLE formalism has no counterpart in the ANLT object grammar formalism, and cannot be simulated.

In the main, the vocabulary of the sample matches that of the CLARE2.5 lexicon fairly well: only a small number of lexical entries needed to be added for the purposes of the experiment. The standard CLARE2.5 parser has a facility for scoring constituents to obtain a final ranking of

analyses; a version of the parser was used that avoids the associated overhead[21]. Timings for this parser were recorded for its 'phrasal' analysis stage in addition to the subsequent main syntactic analysis stage. Since the CLE parser returns analyses in the form of a packed forest, for the purposes of this experiment the ANLT system was modified to do the same, omitting its normal unpacking stage. When running the CLE with the ANLT object grammar, the grammar was compiled, and the features `SLASH`, `INV` and `COMP` were declared as 'prediction filter' features; without the declarations, pure bottom-up parsing was approximately twice as slow. The CLARE2.0 parser was used with this grammar, since preliminary tests unexpectedly indicated that the standard CLARE2.5 parser was more than twice as slow.

Of the initial 175 sentences, the CLARE2.5 grammar failed to parse 42 (in several cases because punctuation is strictly required but is missing from the corpus). The ANLT grammar also failed to parse three of these, plus an additional four. These sentences were removed from the sample, leaving 129 which are made up of four sets, comprising 47 declarative sentences (forming a set called *decl*), 38 wh-questions and other sentences with gaps (*wh/slash*), 20 passives (*passive*), and 24 sentences containing co-ordination (*conj*). The sample contains a total of 861 words, with a mean sentence length of 6.7 words. Appendix A contains a list of the sentences. Table 3.1 shows the parse times for the three configurations for each set of sentences in the sample. The figures include garbage collection time, and phrasal (where appropriate) processing. Table 3.2 gives the total numbers of syntactic analyses found for all of the sentences.

What might be thought the more straightforward comparison between the CLE and ANLT parsers—when both are using the ANLT grammar—is in fact problematic. In the experiment, the configuration using the CLARE2.0 parser is almost 50 times slower than that using the ANLT parser; however, Alshawi (personal communication) points out that the former is not tuned for use with grammars containing large numbers of rules, and the ANLT grammar contains more

	CLARE2.5 parser, CLARE2.5 grammar (phrasal+syntactic)	ANLT chart parser, ANLT grammar	CLARE2.0 parser, ANLT grammar
decl	6.1+ 43.0	19.6	557.7
wh/slash	3.6+ 157.7	25.7	1092.0
passive	3.2+ 21.1	19.1	1687.4
conj	4.8+ 38.2	11.1	353.3
totals	17.7+ 260.0	75.5	3690.4

Table 3.1. Parse Times (in CPU seconds on a Sun Sparc ELC workstation) for each Set of Sentences in the Sample.

[21] Hiyan Alshawi pointed out this facility and ran the sample through a version of CLARE with it disabled.

	CLARE2.5 parser, CLARE2.5 grammar	ANLT chart parser, ANLT grammar	CLARE2.0 parser, ANLT grammar
no. of analyses	278	283	262[22]

Table 3.2. Total Numbers of Analyses for all Sentences in the Sample.

than five times as many rules than does the (non-phrasal part of the) CLARE2.5 grammar. When presented with a large number of rules, the indexing that the CLE uses to store the compiled prediction rule table appears to perform badly. With the ANLT grammar, some entries in the table contain more than 150 rules, and each time an entry in the table is accessed the system will, in effect, make a copy of all of the rules in it. In contrast, in the ANLT parser, access to large numbers of rules (as described in section 3.2) is highly optimised.

Comparing the two parsers in the configurations where each uses the grammar originally written for it, the ANLT parser is a factor of 3.7 faster than the CLE parser over the whole sample[23]. Given that both grammars have broadly similar (wide) coverage, and return remarkably similar numbers of syntactic analyses for the same inputs, the only remaining major differences between the two parsing systems are in the parsing algorithm used and in the programming language in which they are implemented (Common Lisp (Steele, 1984) for the ANLT, Prolog for the CLE). Current Prolog and Lisp compiler and run-time support technologies are all quite advanced, and in fact the CLE parser takes advantage of Prolog's built-in unification operation which will be coded very efficiently. It therefore seems unlikely that the use of the different programming languages could wholly account for a difference of a factor of more than three in favour of the ANLT[24]. It may thus be concluded that it is the parsing and unification algorithms together with their associated optimisations that allow the ANLT parser to perform so well in practice. Indeed, for the sentences in the experiment, running on a Sun Sparc ELC workstation, the ANLT parser processes words at a rate of more than eleven a second.

[22] This figure should of course be 283, but is not, due to a suspected bug in the CLARE parser involving local ambiguity packing, and also to the fact that kleene plus categories in the ANLT grammar were translated into single ordinary categories when used with CLARE, meaning that parses containing more than two co-ordinated phrases were not found.

[23] If the time taken for mophological analysis and creation of logical forms is included, the ANLT system is approximately 2.3 times faster than CLARE.

[24] It is possibly the case, though, that the difference is partly due to the processing strategies and types of algorithm that the programming language leads the implementor of a parser to adopt; Prolog supporting backtracking search and the use of the system database when intermediate results need to be cached, and Lisp, breadth-first search with the accumulation of alternative results in lists.

3.4 Experiments with Top-down Predictive Parsing

The chart parsing algorithm described in this chapter is purely bottom-up. It makes no use of top-down information about what constituents are expected to be present at any particular point in the input on the basis of the preceding left context. (Earley's (1970) parsing algorithm for CF grammars uses top-down information, and chapter 5 describes an implementation of an improved Earley-like parser for unification grammar). Using such information would not improve the complexity of the chart parser, but might result in practical space and time improvements. For example, Pratt (1975) reports that incorporating top-down information into a (purely bottom-up) CKY parser to prevent it hypothesising constituents bottom-up that could not form part of a complete analysis, given the portions of rules already partially instantiated, resulted in a reduction by a factor of five in the number of constituents built for one set of test data. Moore & Dowding (1991) describe a unification-based chart parser in which similar techniques lead to a practical improvement in parse times of a factor of six over an earlier bottom-up version of the same parser. The CLE (Alshawi, 1992) also employs top-down prediction within the framework of a backtracking left corner unification parser, to avoid considering applying a rule whose first daughter has been found, but whose mother could not form part of a complete analysis. All three of these parsers rely on precompilation of the grammar to construct a 'left-corner reachability table'. Given a (non-terminal) unification grammar category, the table gives the set of possible categories that a constituent of that category can start with, i.e. have as its left corner. During parsing, the table is consulted whenever a constituent is about to be built to check that it is a possible left corner of any rule daughter in which the previous daughters have already been found. The constituent is discarded if it is not a possible left corner.

The successful use of top-down prediction in these parsers suggest that it would be worthwhile to add it to the chart parser to improve its practical performance. The parsing algorithm could be changed so that on the addition of an active edge to the chart at a given vertex, a prediction for the edge's next needed category starting at that vertex would be made. When a new lexical entry is encountered, a check should be made that it is a left corner of one of the predicted categories at that point, and when an inactive edge has matched the first daughter of a rule, the rule mother should be checked to see if it is a left corner of a predicted category. A version of the parser was created incorporating these changes, and some experiments carried out to assess their effectiveness.

In the experiments, the reachability table was a two-dimensional array indexed by the category identifiers of the predicted and the left corner categories. Two versions of the table were used: the first was a conventional table with each entry being a list of predicted / possible-left-corner pairs of unification categories; the second was a bit-array in which entries were 1 if a category whose identifier was equal to the entry's predicted index had a left corner category

whose identifier was equal to its left corner index, and zero otherwise. Both versions of the table were created by filling in mother / first daughter relationships from each rule individually, and then applying variants of the transitive closure operation to complete the table, using the Floyd-Warshall algorithm (Sedgewick, 1988). Each stage of this algorithm uses the information in the (i, k) and (k, j) table entries to update the (i, j) entry. In the case of the unification category table, the updating consists of adding new predicted / possible-left-corner pairs taken from combinations of the pairs in the two input entries in which the left corner category of the first pair unifies with the predicted category of the second pair. Adding a new pair to an entry involves a generalisation operation on the current contents[25]. The bit-array version of the table can be completed much more simply with a series of inclusive-or vector operations on slices of the array.

Creating the unification category version of the reachability table for the ANLT grammar (around 750 rules) was computationally very expensive, taking more than 20 minutes of CPU time on a DEC 3100 workstation. The bit-array version was very cheap to create, taking approximately 5 seconds. Table 3.3 summarises the performance on a set of sentences, relative to the bottom-up parser, of the top-down predictive version of the parser using both types of left corner reachability information.

The top-down predictive parser with either type of reachability table was slower than the bottom-up parser, and when using the unification category table it was slower by a factor of more than a hundred. Some optimisations to the predictive parser are definitely possible, but none that would allow it to overtake the bottom-up one, given that only 20% of the inactive edges are filtered out. In fact, if the primary purpose of the table is to use left-context information to limit the places at which gaps can occur (c.f. Moore & Dowding, 1991), the maximum reduction in the number of inactive edges built with the ANLT grammar can be on average only 45% (since this is the average reduction when all rules introducing gaps are removed), which results in a decrease in parse time of a similar figure. The bit-array version of the reachability table turned

Parser	No. inactive edges	CPU time
Bottom-up parser	1.00	1.0
Top-down predictive parser		
Unification category table	0.81	101.7
Bit-array table	0.98	1.4

Table 3.3. Performance of the Top-down Predictive Version of the Parser.

[25] This process has to be repeated until quiescence (with a bound on the depth of embedding and the subsequent generalisation of parts of embedded feature structures) if the grammar contains sets of rules in which mothers and first daughters can be applied to each other indefinite numbers of times and the applications build recursive feature structures, e.g. when the rule `[F @v] --> [F G[H @v]]...` is applied to itself.

out to be of very little use. These results are somewhat disappointing, and run counter to those reported by Moore & Dowding; however, a parser which makes effective use of left context information and in practice runs significantly faster than the bottom-up parser is described in chapter 5.

4. LR Parsing in a Unification-Based Grammar Framework

For some classes of unambiguous grammars (depending on the exact parsing technique being used) a *predictive* parser can be constructed, which, by taking into account what has already been analysed can unambiguously determine the next action that should be performed. In the case of ambiguous grammars there may be several actions possible at any one stage, though many of the predictive techniques can be adapted, or *generalised*, to deal with such grammars. In contrast to the bottom-up chart parser described in chapter 3, a predictive parser uses information about the context in which rules may be applied successfully, rather than attempting to apply all rules at every point. When every action possibility can be distinguished unambiguously, a predictive parser can parse text in time linear in the length of the string. This chapter discusses techniques for constructing a *generalised predictive parser* for a unification-based NL grammar.

The two most popular predictive parsing techniques are LL parsing and LR parsing. Of these, LR is the more powerful, in the sense that it can construct conflict-free parsers for a larger set of unambiguous grammars. (For example, left recursion in a grammar causes the LL technique to produce a parser containing action conflicts, whereas it does not for LR). For this reason, only the LR technique is considered below.

4.1 LR Parsing

LR parsing (Knuth, 1965; Aho, Sethi & Ullman, 1986; Chapman, 1987) is a table-driven parsing technique, in which an (often simple) driver program takes an input and builds an analysis guided by a (LR) parse table constructed from the grammar indicating what the parser should do at each stage. Although originally developed for use in the syntax analysis phase of programming language compilers, where the grammar of the language will have been designed to be unambiguous and to allow the driver program to be a standard deterministic shift-reduce parser, it is also possible to apply the LR technique to ambiguous grammars (e.g. NL grammars). Lang (1974) describes a technique for efficiently driving in pseudo-parallel certain classes of non-deterministic push-down automaton (NPDA; Aho, Sethi & Ullman, 1986); LR parse table construction methods for ambiguous CF grammars can be used to derive such NPDAs. Tomita (1987) describes a more intuitive approach to non-deterministic LR parsing, generalising shift-reduce parsing to operate breadth-first with a 'graph-structured' stack. The LR tables constructed

for these parsers are allowed to contain conflicting actions at points which correspond either to ambiguities in the grammar, or to alternative derivations that the table cannot distinguish between, given the current context.

Non-deterministic LR parsing offers the prospect of fast all-paths parsing, since the use of an LR parse table ensures that rule invocation is as constrained as possible given currently known automatic grammar precompilation techniques. Much of the attraction of the technique stems from the fact that the real work takes place in a precompilation phase and the run time behaviour of the resulting parser is relatively simple and directed. The approach has been extended to probabilistic CF grammar (PCFG, see chapter 6). Chapter 5 presents an efficient, novel LR parsing algorithm, discusses other LR and LR-like non-deterministic parsing algorithms, compares their theoretical complexity and makes practical comparisons between implementations of some of them.

Recently, Briscoe & Carroll (1991; 1993) have investigated using LR parse tables to form the basis of a context dependent language model for probabilistic parsing. In contrast to an individual PCFG rule, an action in an LR parse table implicitly encodes information about the left and right context of the state of the parse in which it is applicable. Deriving probabilities relative to the parse context allows a probabilistic parser to distinguish situations in which identical rules reapply in different ways across different derivations or apply with differing probabilities in different contexts. Chapter 6 describes the techniques employed in these investigations, and the results of experiments evaluating the approach.

However, before an LR parser can be applied to an input, a parse table for the grammar has to be constructed. This is computationally the most expensive and arguably the most complex aspect of LR parsing. This chapter describes the form of an LR table, how it drives a simple shift-reduce parser, discusses alternative methods of generating LR parsers for unification-based grammars and special steps that have to be taken to be able to construct a table for a large, wide-coverage grammar, such as the ANLT.

4.1.1 Shift-Reduce Context-free LR Parsing

A simple context-free grammar (G1) is shown in Figure 4.1. The start symbol of the grammar is `S'`. This grammar is ambiguous in that it allows right- or left-branching analyses of both PP attachment and noun compounds. Figure 4.2 gives an LR parse table for G1. The table consists of two parts: an *action* table and a *goto* table. Rows in the table represent parse *states*, and columns represent terminal and non-terminal symbols of the grammar. The table contains action conflicts in states six and nine, the former corresponding to noun compound ambiguity and the latter to PP attachment ambiguity.

```
1:  S' --> S                    6:  NP --> Det N
2:  S --> NP VP                 7:  NP --> NP PP
3:  VP --> Vt NP                8:  N --> N N
4:  VP --> Vi                   9:  PP --> P NP
5:  NP --> ProNP                10: N --> N@
```

Figure 4.1. G1: A Simple Context-Free Grammar.

The easiest way to understand the table's function is to examine a simple derivation in which the table drives a standard shift-reduce parser (e.g. Aho, Sethi & Ullman, 1986). Figure 4.3 gives the derivation for a sentence such as *he loves her* consisting of a pronoun (`ProNP`), transitive verb (`Vt`) and a second pronoun. Since there is no ambiguity in the sentence, the derivation can be represented as a single stack (with the top element on the right). The parse table for G1 is set up to expect to know the terminal category of the next word in the input at any stage in the parse; this is called the 'lookahead symbol'. Once all the words have been processed, the lookahead item is the 'end symbol', `$`.

The parser begins in state zero with an otherwise empty stack and the lookahead item being the terminal symbol associated with the first word of the input. The parse table entry for state zero with lookahead `ProNP` instructs the parser to *shift* and move to state 2. The shift causes `ProNP` to be pushed onto the parse stack followed by the new state number, and the lookahead item to be set to the terminal symbol associated with the next word in the input. In state 2 with lookahead `Vt` the table instructs the parser to *reduce* with rule five (`NP --> ProNP`), so the current state is popped and the topmost category is replaced with the mother category of rule five. The parser next consults the goto part of the table to discover which state it should move to when in state zero with the non-terminal `NP`. The goto table instructs the parser to move to state 7 so this state is pushed onto the stack. Now two further shift actions occur (exactly as before) and the end symbol (`$`) becomes the lookahead item. At the next step, the second pronoun is reduced to `NP`, but this time the goto table instructs the parser to move to state 12 (from state 11). In state 12 another reduction is called for, using rule three (`VP --> Vt NP`), but this time two terminals on the stack are popped and replaced with `VP` because the rule has two daughters. Two further reductions bring the parser to the *accept* action and the parse halts successfully.

4.1.2 LR Parse Tables

An LR parser uses a left-to-right scan of an input (hence the "L" in LR) and constructs a 'rightmost derivation in reverse' (hence the "R"). The number of input symbols of lookahead (k)

Action

State	$	Det	N@	P	ProNP	Vi	Vt
0		s3			s2		
1	r1						
2	r5			r5		r5	r5
3			s4				
4	r10		r10	r10		r10	r10
5	r6		s4	r6		r6	r6
6	r8		r8/s4	r8		r8	r8
7				s8		s13	s11
8		s3			s2		
9	r9			r9/s8		r9	r9
10	r7			r7		r7	r7
11		s3			s2		
12	r3			s8			
13	r4						
14	r2						
15	acc						

Goto

State	N	NP	PP	S	S'	VP
0		7		1	15	
1						
2						
3	5					
4						
5	6					
6	6					
7			10			14
8		9				
9			10			
10						
11		12				
12			10			
13						
14						
15						

Figure 4.2. An LR Parse Table for Grammar G1.

Stack	*Input*	*Lookahead symbol*
0	he loves her $	ProNP
0 ProNP 2	loves her $	Vt
0 NP	loves her $	Vt
0 NP 7	loves her $	Vt
0 NP 7 Vt 11	her $	ProNP
0 NP 7 Vt 11 ProNP 2	$	$
0 NP 7 Vt 11 NP	$	$
0 NP 7 Vt 11 NP 12	$	$
0 NP 7 VP	$	$
0 NP 7 VP 14	$	$
0 S	$	$
0 S 1	$	$
0 S'	$	$
0 S' 15	$	$

Figure 4.3. Parse derivation for *he loves her*.

that are used in making decisions on which action to perform next is appended to the name of the technique, although it is often omitted when it is 1. If we consider the derivation (i.e. generation)of a string from the start symbol of the grammar, at each step there are two choices: which nonterminal to expand, and which rule to use to expand it. A *rightmost derivation* is one in which only the rightmost nonterminal is expanded at any step. When parsing bottom-up, the input is a string for which a parse tree is to be constructed, starting at the leaves and working up to the top (the start symbol). At each stage in the parse, when the appropriate rule is applied at the correct point, reducing a substring (corresponding to the daughters) to the rule mother, the parser constructs a rightmost derivation in reverse.

The LR precompilation process results in a parser control mechanism which enables the parser to always be able to identify the *handle* , or appropriate substring in the input to reduce, together with the rule of the grammar with which to perform the reduction, in order to construct a rightmost derivation. (If the grammar is ambiguous or if there are alternative derivations that cannot be distinguished in the current context then there may be more than one handle for a given string). In a shift-reduce LR parser, the state on the top of the stack encodes all the information the control mechanism needs for it to be able to identify the current handle (or set of handles). The goto part of an LR table is essentially a finite state automaton which, given the state currently on top of the stack and a grammar symbol being incorporated into the current derivation, can identify the next handle (set). Intuitively, each state corresponds to the parsing process having consumed input—or being about to do so—which spans some initial portion of one or more of the rules in the grammar. For a given rule, this is represented by an *LR(0) item*,

consisting of the rule with a dot at some position in the right hand side. (This is the case even in LR(k) parsing for $k > 0$, since the information required is always the same). For example, the items stemming from the rule `S --> NP VP` are

```
[S --> · NP VP]
[S --> NP · VP]
[S --> NP VP ·]
```

The first item indicates that a string derivable from `NP VP` is expected next, and the second, that a string derivable from `NP` has just been seen in the input, and that one from `VP` is now expected. An LR(1) item is like an LR(0) item except that it contains a terminal symbol as a second component, the *lookahead* of the item, for example `[S --> NP VP ·, $]`. Lookaheads in items may be of length greater than 1 in LR(k) parsing in the case where k is greater than 1.

The action part of an LR table encodes a finite state automaton which recognises *viable prefixes*, that is, prefixes of a string that do not continue past the rightmost handle of that string. The states in this automaton are the same as those identified by the goto automaton, a state being formed from a set of items. Each set is formed by a *closure* operation: for each item `[A -->` α`·B`β`]` (where α and β are possibly empty sequences of terminals and / or nonterminals) in the set, and for each grammar rule `B -->` γ, the item `[B -->` `·`γ`]` is added to the set unless it is already present. Forming the item sets starts with the computation of the closure of the set containing initially just `[S' --> ·S]`. If `[A -->` α`·a`β`]` is a member of set I then the value of the function *goto*(I, a) is the set of items formed from the closure of `[A -->` α`a·`β`]`. The goto automaton is constructed from this function, the argument a being restricted to only nonterminal values. Once the closures have been computed, if a set which forms state I contains an item whose dot is after the last daughter, i.e. `[A -->` α`·]`, the rule `A -->` α must be a handle at the current point of the input string, and a reduction by this rule is called for. An item of the form `[A -->` α`·a`β`]` where a is a terminal and $goto(I, a) = J$, gives rise to an action specifying a shift to state J.

Grammars for programming languages will almost always give rise to deterministic automata; however, it is possible to generalise the LR technique to apply to ambiguous CF grammars, such as those describing natural languages, by building non-deterministic automata in which some states contain action conflicts (arising from a single string licensed by the grammar having alternative handles). An action conflict may be of one of two types: between a shift and one or more reduce actions, or between several reduce actions calling for reductions with distinct rules. Shift-shift conflicts can never occur. The parse table for G1 above contains two shift-reduce conflicts. Blank entries in the table correspond to illegal configurations of the stack and the input buffer at a given point in the parse; if the parser reaches one of these entries the input is ungrammatical.

4.1.3 Variants of LR

The table construction method outlined in the previous section produces an LR(0) parser: the collection of sets of items that are constructed is termed the *canonical* LR(0) collection. A grammar for which a conflict-free LR(k) parse table can be constructed is called an LR(k) grammar. An LR(0) parser is quite weak, since even most programming languages are not LR(0). An LR(0) parse table for grammar G1 would contain several action conflicts, for example, once a transitive verb followed by a noun phrase has been recognised, the parse table would not be able to say whether a verb phrase should be constructed immediately from the two constituents, or whether the next symbol should be shifted to start to form a prepositional phrase that might modify the noun phrase. However, looking at G1 in figure 4.1, it is evident that the first of these options is only applicable when the end of the sentence is reached, and the second only when the next word in the sentence is a preposition. One word of lookahead is needed to be able to distinguish between the two cases.

An SLR ('Simple LR') parser is the simplest type of lookahead LR parser. An SLR(1) parse table is derived from the canonical LR(0) collection of item sets in the same way as an LR(0) table, with the single exception that the item `[A --> α·]` results in a reduction by rule `A -->` α only when the lookahead symbol is a member of the set *FOLLOW*(*A*). *FOLLOW*(*A*) is defined for a nonterminal *A* to be the set of terminals that can appear immediately to the right of *A* in some derivation of a string. For grammar G1, the conflict mentioned above between reducing a transitive verb and noun phrase to a verb phrase (`VP`), or shifting the next symbol, would be resolved by looking at the next symbol, and if it is in *FOLLOW* (*VP*) the reduction would take place, otherwise the shift would be performed. In G1, *FOLLOW*(*VP*) = {*$*}, so the reduction is (correctly) called for only in the case when the lookahead symbol is `$` (i.e. at the end of the sentence). However, with G1 in the case in which a determiner and noun have been recognised, even the SLR technique is not powerful enough to discriminate between the choice of reducing the constituents to a noun phrase, or shifting a following `N@` terminal to form part of a compound nominal.

Canonical LR(1) is the most powerful LR(1) parsing technique. A canonical LR(1) parse table can be constructed in an analogous way to an LR(0) one, but using LR(1) items instead of LR(0) items. The closure operation on an item set *I* involves adding `[B --> ·γ, b]` in response to an existing item `[A --> α·Bβ, a]` and a rule `B -->` γ, as before, but only when *b* is in *FIRST*(*βa*), that is, when *b* can begin a string derived from *βa*. The goto function can also be computed in a very similar way to the LR(0) goto, except that items are now of the form `[A --> α·aβ, b]`. When creating the action table from the item sets, the item `[A --> α·, a]` should result in a reduction by rule `A -->` α only when the lookahead symbol is *a*. For grammar G1, the canonical LR(1) table contains 23 states and only two action conflicts, each of which

corresponds to a point of structural ambiguity in the grammar for which a potentially unlimited amount of lookahead would be needed to make the correct choice of action.

Thus, an SLR table can contain action conflicts for certain types of grammatical phenomena where a canonical LR table can discriminate between the alternative actions. A third technique, LALR ('LookAhead LR') is intermediate in power between SLR and full LR, but has the advantage over the latter that the parse table for a grammar is often much smaller, always having the same number of states as the SLR and LR(0) tables. Figure 4.2 shows the LALR(1) table for grammar G1. The table contains 15 states, as opposed to 23 for the canonical LR(1) table, but still has only (the same) two action conflicts. For larger grammars, for example those for programming languages, Aho, Sethi & Ullman (1986:236) remark that canonical LR(1) tables are often more than an order of magnitude larger than the corresponding LALR(1) table. For this reason, and because LALR(1) is usually powerful enough for programming language grammars, most parser-generators for programming languages use LALR(1) (e.g. Johnson, 1975; Klein & Martin, 1989; Grosch, 1990), and the LR(1) technique appears to be rarely used in practice. The literature contains a number of different approaches to LALR table construction. Section 4.2.3 discusses the suitability of the various construction methods for creating tables for large, ambiguous NL grammars.

4.1.4 Extended Lookahead LR Parsing

Standard approaches to LR parsing use a fixed number of lookahead symbols in order to choose between otherwise conflicting actions. The number of symbols of lookahead is invariably taken to be 1 to control the size of the parse table (Chapman, 1987). Bermudez (1991) presents a technique, called LAR, which constructs LR-like parsers for unambiguous CF grammars in which an LR(0) parser is effectively augmented by a set of lookahead automata, each of which is able to perform an unlimited amount of lookahead. The automata are used to skip sections of the input containing local ambiguities which are eventually resolved, and thus deterministically choose the action that should be performed by the LR(0) parser at the current point.

Hektoen (1991) adapts the LAR technique to construct tables for (ambiguous) NL grammars to reduce the non-determinism resulting from local ambiguity as much as possible. LAR table construction is computationally very expensive since for all action conflicts in the LR(0) table the process simulates the complete non-deterministic parsing of all possible input strings up to the resolution of the local ambiguity. On the basis of the results of experiments with a small ambiguous grammar, Hektoen concludes that the analysis that the LAR technique performs does not extend lookahead in a way that is relevant to phenomena that occur in NL, and also that the technique is too inefficient to be applied to realistic NL grammars on current computer hardware.

This result indicates that, in the foreseeable future, there is little hope of automatic table construction techniques being successfully applied to such grammars expressed in more general frameworks, such as LR-regular (Culik & Cohen, 1973).

4.2 Creating LR Parse Tables for Unification Grammars

4.2.1 Previous Approaches

Tomita (1987) describes a system for non-deterministic LR parsing of context-free grammars consisting of atomic categories, in which each CF production may be augmented with a set of tests (which perform similar types of operations to those available in a unification grammar). At parse time, whenever a sequence of constituents is about to be reduced into a higher-level constituent using a production, the augmentation associated with the production is invoked to check syntactic or semantic constraints such as agreement, pass attribute values between constituents, and construct a representation of the higher-level constituent. (This is the standard approach to parsing with attribute grammars). The parser is driven by an LR parse table; however, the table is constructed solely from the CF portion of the grammar, and so none of the extra information embodied in the augmentations is taken into account during its construction. Thus the predictive power of the parser to select the appropriate rule given a specific parse history is limited to the CF portion of the grammar which must be defined manually by the grammar writer. This requirement places a greater load on the grammar writer and is inconsistent with most recent unification-based grammar formalisms, which represent grammatical categories entirely as feature bundles (e.g. Gazdar *et al.*, 1985; Pollard & Sag, 1987; Zeevat, Klein & Calder, 1987). In addition, it violates the principle that grammatical formalisms should be declarative and defined independently of parsing procedure, since different definitions of the CF portion of the grammar will, at least, affect the efficiency of the resulting parser and might, in principle, lead to non-termination on certain inputs in a manner similar to that described by Shieber (1985).

For example, consider the following ANLT metagrammar PS rule which analyses a sentence as a noun phrase followed by a verb phrase.

```
S: [N -, V +, BAR 2, SUBJ +, PER @per, PLU @plu, VFORM @vf] -->
       [N +, V -, BAR 2, PER @per, PLU @plu, CASE NOM]
       [N -, V +, BAR 2, SUBJ -, PER @per, PLU @plu, VFORM @vf].
```

To be able to use this rule in Tomita's LR parsing framework, the rule must be manually split up to form, firstly, a CF rule in which a chosen set of feature / value pairs from each category are

represented by an atomic symbol, and secondly, an augmentation in which each category contains the respective 'residue' of features. If the feature set for the creation of the CF rule is chosen to be {N V} for each of the three categories, the CF rule and residue would be as follows.

```
S-CF: Vb --> Nn Vb.
S-Residue:
   [BAR 2, SUBJ +, PER @per, PLU @plu, VFORM @vf]
   [BAR 2, PER @per, PLU @plu, CASE Nom]
   [BAR 2, SUBJ -, PER @per, PLU @plu, VFORM @vf]
```

However, it is not obvious what the set of features should be when forming the CF rules; in this example, BAR (and additionally SUBJ for the verbal categories and CASE for the nominal category) would also be candidates, since these features have fully specified values. It will be difficult for the grammar writer to make such substitutions in a consistent way, and still more difficult to make them in an optimal way for the purposes of LR parsing, since both steps involve the manual comparison of all the categories appearing in each rule of the grammar.

Constructing the LR parse table directly and automatically from a unification grammar would avoid these drawbacks. In this case, the LR parse table would be based on complex categories, with unification of complex categories taking the place of equality of atomic ones in the standard LR parse table construction algorithm (Osborne, 1990; Nakazawa, 1991). However, this approach is computationally prohibitively expensive: Osborne (1990:26) reports that his system (implemented in Hewlett Packard Common Lisp on an HP 9000/350 workstation) takes almost 24 hours to construct the LR(0) states for a unification grammar of just 75 productions. Also, in the general case, the procedure is not guaranteed to terminate.

Shieber (1985) describes a technique, *restriction*, for filtering the information contained in a category, or DAG, to increase the efficiency of, or induce termination in parsers and other applications which use unification grammars. In theory, this technique could be used in both of the approaches just described, in one to induce statically a CF grammar from a unification grammar (by creating terminal and non-terminal categories from disjoint sets of equivalence classes of all the DAGs that have the same DAG as their restriction), and in the other to speed up and guarantee termination when directly constructing a parse table from a unification grammar (Shieber, 1985:151). However, in the latter case, for large grammars it seems unlikely that restriction could reduce the time taken by the computation by the orders of magnitude necessary, and in both cases, the (manual) definition of effective restrictors is difficult, requiring detailed knowledge of the dynamic, runtime behaviour of a grammar.

4.2.2 Constructing a CF Backbone

Approaches to LR parsing with unification grammars involving either the direct construction of parse tables or the initial manual derivation of a CF grammar are unsatisfactory. This section describes an alternative, novel approach to this task which is computationally tractable for realistic sized grammars and also safe from inconsistency. The starting point is an ANLT object grammar, though the techniques involved should generalise to many other unification-based formalisms. From the object grammar, a CF *backbone* of rules containing categories with atomic names and an associated residue of feature name-value pairs is automatically constructed. Each backbone grammar rule is generally in direct one-to-one correspondence with a single object unification grammar rule. The LR parse table is then constructed from the CF backbone grammar. The parser is driven by this table, but in addition when reducing a sequence of constituents the parser performs the unifications specified in the relevant unification grammar rule to form the category representing the higher-level constituent, and the derivation fails if one of the unifications fails. This parser is thus similar to Tomita's, except that it performs unifications rather than invoking CF rule augmentations; however the main difference between the approach described here and Tomita's is the way in which the CF grammar which drives the parser comes into being.

This approach results in an on-line parser. An off-line strategy would not work in general, since it is possible for the automatic backbone generation process to create a CF grammar that is not sufficiently constrained for termination to be guaranteed. Thus, in a more general context, only strategies that are variants of Maxwell & Kaplan's (in press) 'interleaved pruning' can be employed. However, any on-line strategy employing the approach outlined above will have behaviour identical to a parser using just the original unification grammar.

Even though a unification grammar will be, at best, equivalent to a very large (and at worst, if features are employed in recursive or cyclic ways, possibly infinite) set of atomic-category CF productions, in practice the technique described here has produced LR parsers that perform well from backbone grammars containing only about 30% more productions than the original unification grammar. The construction method produces an optimal CF grammar, in the sense that there is no alternative, consistent CF grammar that captures more information from the unification grammar, under a one-to-one rule mapping. Thus the construction method guarantees that the resulting LR parser will terminate and will be as predictive as the source grammar in principle allows. The method will always produce an identical or more constrained backbone than one induced using restriction (see previous section), since it makes optimal use of all information from every part of each category, rather than just from the restriction of each category.

Building the backbone grammar is a two-stage process:

(1) Compute the largest maximally specific set (in terms of subsumption) of disjoint categories covering the whole grammar and assign to each category a distinct atomic category name. That is:

```
let Disjoint := {}
for ∀c ∈ CATEGORIES
   let merge := {b ∈ Disjoint s.t. b unifies with c}
   if merge = {} then
      Disjoint := Disjoint ∪ {c}
   else
      let s := most specific category s.t.
                ∀m ∈ merge ∪ {c}, s subsumes m
      Disjoint := {s} ∪ Disjoint − merge
assign a distinct name to each category in Disjoint
```

(2) For each unification grammar rule, create a backbone grammar rule containing atomic categories, each atomic category being the name assigned to the category in the disjoint category set which unifies with the corresponding category in the unification grammar rule:

```
let Backbone := {}
for ∀r = <rule c1, [c2 ··· cn]> ∈ RULES
   let b := <rule b1, [b2 ··· bn]>
      where bi is the name assigned to the (single) category in
      Disjoint which unifies with ci, for i=1, n
   Backbone := Backbone ∪ {b}
```

For example, from the object grammar rules compiled from the ANLT metagrammar PS rules in Figure 4.1a (which analyse a sentence as a noun phrase followed by a verb phrase, a noun phrase as a determiner followed by an `N1`, and a verb phrase as an intransitive verb or as a transitive verb followed by a noun phrase), step 1 would create the set *Disjoint* shown in Figure 4.1b. (Note that the value for `CASE` on the nominal categories in the grammar has 'collapsed' down to a variable, but that the two lexical verbal categories remain distinct). Figure 4.1c shows the backbone rules that would be built in step 2.

Algorithms for creating LR parse tables assume that the terminal vocabulary of the grammar is distinct from the non-terminal one, so the procedure described above will not deal properly with a unification grammar rule whose mother category is assumed elsewhere in the grammar to be a lexical category. For this case, the procedure is modified to associate with such categories two different atomic categories, one terminal and one non-terminal, and to augment the backbone grammar with a unary rule expanding the non-terminal category to the terminal.

Two other aspects of the ANLT object grammar formalism require further minor elaborations to the basic algorithm. Firstly, a rule may introduce a gap by including the feature

```
S:  [N −, V +, BAR 2, SUBJ +, PER @per, PLU @plu, VFORM @vf] -->
       [N +, V −, BAR 2, PER @per, PLU @plu, CASE NOM]
       [N −, V +, BAR 2, SUBJ −, PER @per, PLU @plu, VFORM @vf].

N2: [N +, V −, BAR 2, PER @per, PLU @plu, CASE @case] -->
       [SUBCAT DET]
       [N +, V −, BAR 1, PER @per, PLU @plu, CASE @case].

VI: [N −, V +, BAR 2, SUBJ −, PER @per, PLU @plu, VFORM @vf] -->
       [N −, V +, BAR 0, PER @per, PLU @plu, VFORM @vf,
        SUBCAT INTRANS].

VT: [N −, V +, BAR 2, SUBJ −, PER @per, PLU @plu, VFORM @vf] -->
       [N −, V +, BAR 0, PER @per, PLU @plu, VFORM @vf, SUBCAT TRANS]
       [N +, V −, BAR 2, PER @per, PLU @plu, CASE ACC].
```

Figure 4.1a. ANLT Metagrammar PS Rules.

```
{S−1:  [N −, V +, BAR 2, SUBJ +, PER @, PLU @, VFORM @]
 NP−2: [N +, V −, BAR 2, PER @, PLU @, CASE @]
 VP−3: [N −, V +, BAR 2, SUBJ −, PER @, PLU @, VFORM @]
 DET−4:[SUBCAT DET]
 N1−5: [N +, V −, BAR 1, PER @, PLU @, CASE @]
 V−6:  [N −, V +, BAR 0, PER @, PLU @, VFORM @, SUBCAT INTRANS]
 V−7:  [N −, V +, BAR 0, PER @, PLU @, VFORM @, SUBCAT TRANS]}
```

Figure 4.1b. Categories in *Disjoint*.

```
S−1  --> NP−2 VP−3.
NP−2 --> DET−4 N1−5.
VP−3 --> V−6.
VP−3 --> V−7 NP−2.
```

Figure 4.1c. Backbone Grammar.

specification `[NULL +]` on the gapped daughter: for each such daughter an extra rule is added to the backbone grammar expanding the gap category to the null string. Secondly, for rules containing kleene plus daughters, a new non-terminal category is created for each such daughter and two extra rules are added to the backbone grammar. One (unary) rule expands the new category as the original category, and the other (binary) rule expands the new category to the original category followed by the new one. Without further modifications this would engender a right-branching binary tree structure, rather than the intended flat sequence of categories, but a

parser can easily be modified to flatten out the tree structure during processing. For example, in the rule

```
N2/EITHER: N2 --> N2[CONJ EITHER], ( N2[CONJ OR] )+.
```

licensing noun phrases of the form *either kim or lee or sandy*, the second daughter may be repeated an indefinite number of times. Corresponding to this unification grammar rule there would be three backbone grammar rules, forming for this noun phrase the (unflattened) structure shown in Figure 4.2.

Grammars written in other, 'lower-level' unification grammar formalisms, such as PATR (Shieber, 1984), commonly employ treatments of the type just described to deal with phenomena such as gapping, coordination and compounding. However, the processing techniques described here permit the grammar writer to continue to write grammars which use the full facilities of the metagrammatical formalism, and also allow the algorithmic derivation of an appropriate backbone grammar to support LR parsing.

The major task of the backbone grammar is to encode sufficient information (in the atomic categoried CF rules) from the unification grammar to constrain the application of the latter's rules at parse time. The nearly one-to-one mapping of unification grammar rules to backbone grammar rules described above works quite well when applied to the ANLT grammar, except for a couple of exceptions which create spurious shift-reduce conflicts during parsing resulting in an unacceptable degradation in performance. The phenomena concerned are coordination and unbounded dependency constructions.

The ANLT grammar uses three very general rules to form nominal, adjectival and prepositional phrases following a conjunction; the categories in these rules lead to otherwise disjoint categories for conjuncts being merged, giving rise to a set of overly general backbone grammar rules. For example, the rule in the ANLT metagrammar for forming a noun phrase conjunct introduced by a conjunction is

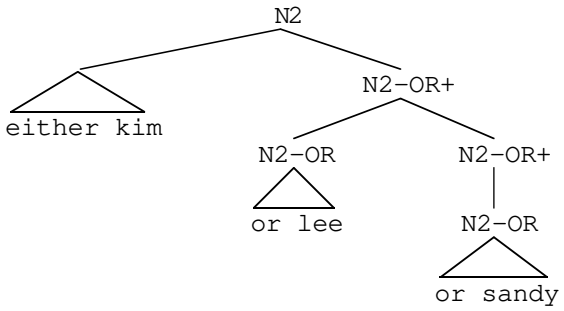

Figure 4.2. Backbone parse tree for *either kim or lee or sandy* .

```
N2/CONJ: N2[CONJ @con] --> [SUBCAT @con, CONJN +], H2.
```

The variable value for the CONJ feature in the mother means that all N2 categories specified for this feature (e.g. N2[CONJ EITHER], N2[CONJ NULL]) are generalised to the same category. This results in the backbone rules, when parsing *either kim or lee helps*, being unable, after forming a N2[CONJ EITHER] constituent for *either kim*, to discriminate between the alternatives of preparing to iterate the constituent (as in the phrase *kim, lee or sandy helps* where *kim* would be N2[CONJ NULL]), or shifting the next word *or* to start a new constituent. In the system, this problem is solved by declaring CONJ to be a feature which may not have a variable value in an element of the disjoint category set. This directs the system to expand out each unification grammar rule which has a category containing this feature with a variable value into a number of rules fully-specified for the feature, and to create backbone rules for each of these. There are eight possible values for CONJ in the grammar, so the general rule for forming a nominal conjunct given above, for example, ends up being represented by a set of eight specialised backbone grammar rules.

In the ANLT grammar, unbounded dependency constructions (UBCs) are analysed by propagating the preposed constituent through the parse tree (*via* non-lexical major categories) as the value of the SLASH feature (section 2.2). Since each category that is specified for SLASH will usually appear at least once in the grammar with a variable value for the feature, the backbone grammar will contain no information about the possible instantiations of SLASH in these categories, and will therefore lose the information in the unification grammar about where gaps are allowed to occur, significantly degrading the performance of a parser. To carry the information over into the backbone a declaration is made that ensures that wherever SLASH occurs with a variable value, the value is expanded out into two values: [NOSLASH +], and a notional value unifying with anything except [NOSLASH +]. In experiments with a smaller grammar employing gap threading (Pereira & Shieber, 1987), the same techniques for expanding out and inference on the values of the (in this case atomic) features used for threading the gaps were able to produce a backbone grammar (and parse table) that had the same constraining power with respect to gaps as the original grammar. A sufficiently constraining backbone grammar has also been built from a previous release of the ANLT grammar (Grover *et al.*, 1989) which analysed gaps in yet another different way.

The construction of CF backbones for grammars written in formalisms with minimal phrase-structure components and (almost) completely general categories, such as HPSG (Pollard & Sag, 1987) and UCG (Zeevat, Klein & Calder, 1987), though, would require more extensive inference on patterns of possible unification within nested categories and appropriate expanding-out of the categories concerned to enable an LR parser to work effectively. This and other areas of complexity in unification-based formalisms need further investigation before this system can be

claimed to be capable of producing a useful CF backbone for any unification-based grammar. In particular, declaring certain category-valued features so that they cannot take variable values may lead to non-termination in the backbone construction for some grammars. However, it should be possible to restrict the set of features which are considered in category-valued features in an analogous way to Shieber's (1985) restrictors for Earley's (1970) algorithm, so that a parse table can still be constructed.

4.2.3 Building LR Parse Tables for Large NL Grammars

Constructing an LR Parse Table for the ANLT

The backbone grammar generated from the ANLT grammar is large: it contains more than 550 distinct categories and 2,000 productions. Even programming language grammars, typically with up to 350 categories and productions, can give rise to canonical LR(1) tables containing several thousand states (Aho, Sethi & Ullman, 1986:236), whereas LALR(1) tables would contain only several hundred. However, Spector (1988) argues that LR(1) tables are often so much larger than SLR(1) and LALR(1) ones only because the standard construction methods (i.e. the one outlined above and Knuth's (1965) method that splits LR(0) states based on the propagation of lookaheads) produce canonical LR tables, and that canonical tables can be larger than is strictly necessary. Spector explains how to construct *minimal-state* tables which are nevertheless full LR(1), and which he claims are comparable in size to those that are produced by the LALR technique.

Spector's construction technique first computes the LR(0) states, then the full LR(1) lookahead sets for each item that would call for a reduce action. Any states with conflicting reduce-reduce actions for a single lookahead are split into the minimum number of states required to ensure that, in the LR automaton, every path of shifts to any of these states from predecessor states is not branched. The technique cannot remove shift-reduce conflicts since, as Pezaris (1989) observes, LR(0) and LR(1) are essentially the same in this respect[26]. For grammar G1, the minimal LR(1) table contains the same number of states (15) as the LR(0) and LALR(1) tables, rather than the 23 of the *canonical* LR(1) table, since there are no reduce-reduce conflicts. However, the minimal LR(1) table for the ANLT backbone grammar would contain an unmanageable number of states; taking a single state with reduce-reduce conflicts at random (one which turns out to be reducing a phrase which ends in a standard `N1` constituent), there are 26 immediate predecessor states, and in turn several of these each have of the order of 50 immediate

[26] Although an LR(0) table for a grammar might contain more shift-reduce conflicts than the LR(1) table, given the same strategy for resolving such conflicts (e.g. a shift always being performed in preference to a reduce), the pattern of shifts and reduces for any well-formed input will be the same.

predecessors. Splitting states to determinise these predecessor paths would result in a massive increase in the number of states.

For a large ambiguous NL grammar, it thus appears that even minimal-state LR(1) table construction is impractical. LALR(1) is the next most powerful standard LR technique. Several algorithms for constructing LALR parse tables have been put forward in the literature; the approaches they take can be grouped into three categories.

(1) The canonical LR(1) item sets are constructed, and then sets with common *cores* (items when lookaheads are disregarded) are merged (Anderson, Eve & Horning, 1973:20ff; Aho, Sethi & Ullman, 1986:236ff). The LALR(1) table is constructed from the resulting sets in the same way as a canonical LR(1) table, as described above in section 4.1.3.

(2) The LR(0) sets of items are computed, and then the items are augmented with lookahead symbols. The parse table is constructed from the item sets in the same way as in the previous approach, noting that lookaheads are required only for *kernel* items (the item `[S' --> ·S]`, where `S'` is the start symbol of the grammar, and all items which have a symbol to the left of the dot). To add the lookaheads, Anderson, Eve & Horning (1973:21ff) (also Aho, Sethi & Ullman, 1986:240ff; Johnson, 1975) give an algorithm which computes the LR(1) closure of each LR(0) *kernel* item for a dummy lookahead symbol, in order to determine the lookaheads which are generated *spontaneously* and those which *propagate* to other items; the algorithm then makes repeated passes over items propagating lookaheads until quiescence is reached. LaLonde, Lee & Horning (1971) present an alternative algorithm which computes the lookaheads for a given LR(0) item by performing a single recursive traversal of all the item sets. A similar but more efficient version of this algorithm is given by Kristensen & Madsen (1981).

(3) An 'indirect' approach is used. DeRemer & Pennello (1982) transform the parse table construction problem into a set of well-known directed graph problems, which in turn are solvable by efficient algorithms. Ancona *et al.* (1991) construct a 'minimal' set of inadequate states (i.e. containing action conflicts), and augment each such state with a set of lookahead states to resolve the conflicts.

DeRemer & Pennello (1982:633) state that their graph-based algorithm does not work for grammars which are not LR(k) for any k. Ancona *et al.*'s technique fails if a lookahead state itself is found to be inadequate, as would be the case with a non-LR(k) grammar. Ambiguous grammars fall into this class, and there is no obvious way of extending these techniques to cope with them, so approaches of these sort cannot be used for the ANLT backbone grammar.

Approach (1) cannot be used either, since it involves an attempt to compute the canonical LR(1) table, and, as discussed above, even the construction of the *minimal* state table is impractical for the ANLT grammar.

An initial implementation of approach (2) with a cut-down version of the ANLT backbone grammar following the technique of Anderson *et al.* demonstrated that the LR(1) closure operation for only a single dummy lookahead was too expensive to be practicable for the number and size of LR(0) states in the full grammar, even with schemes for caching the LR(1) closures of non-kernel items constructed at intermediate stages of the computation. (The technique also does unnecessary work since it has to compute the LALR(1) lookahead for non-kernel as well as kernel items). Kristensen & Madsen's algorithm avoids performing the LR(1) closure operation by computing the lookaheads for a given (kernel) item during a single traversal of all the LR(0) item sets. An extension to their basic algorithm (Kristensen & Madsen, 1981:69ff) marks items and sets that have been visited in the traversal, allowing the computation to tell whether the calculation of the lookahead set for a particular item has been completed, is underway, or has not yet started. This enables partially computed lookahead sets to be cached (with the computation yet to be done explicitly marked), and items whose lookahead sets are found to subsume those of others are able to just copy the results from the subsumed sets. An implementation of the basic algorithm was not able to compute the lookaheads for the ANLT backbone in any reasonable amount of time, but with the incorporation of a careful implementation of the extension, the computation became practical.

The ANLT backbone grammar gives rise to 4,180 LR(0) item sets. Only the kernel items are computed for each set, non-kernel items being cached and retrieved only when needed. Being able to partition the items in this way is especially useful with the ANLT grammar, since the mean number of kernel items in each LR(0) set is approximately 8.5, whereas the mean number of non-kernel items per state is almost 550.

Representing an LR Parse Table for a Large Ambiguous Grammar

To keep the size of the parse table within reasonable bounds, appropriate data structures must be used to represent it. This is an important issue in a practical parsing system: Tomabechi (1991:316) indicates that the full version of his system always ends up running in 'swap space', with associated performance penalties, because the LR table consumes so much memory. The table for the ANLT backbone grammar contains approximately 150,000 goto entries (non-terminal / state pairs), 620,000 shift actions (terminal / state pairs), and 1,140,000 reduce actions (terminal / rule-number pairs); however, of the goto entries only 2,600 are distinct and of the shift actions only 1,600 are distinct, most states contain just reduce or just shift actions, and in

any one state very few different rules are involved in reduce actions[27]. Taking advantage of the characteristics of this distribution, in each state the system represents (in Common Lisp (Steele, 1984)):

(a) a set of goto entries as a list of (non-terminal / state) *conses* sorted into a canonical order, list elements and tails of lists shared where possible between states,

(b) a set of shift actions as a list containing a single (large) integer (the list shared when possible between states), where if the state shifts to state *s* on lookahead *t*, the element indexed by *t* in an auxiliary array will contain *s* together with a number *n*, and bit *n* in the binary representation of the integer will be 1,

(c) a set of reduce actions as, for each rule involved, a *cons* whose second element is the rule number and whose first is a bit-vector (shared when possible between states) whose *n*th bit is 1 if the reduce should occur with the *n*th terminal as lookahead,

(d) an accept action as a *cons* with the first element being the lookahead symbol.

The parse table for the ANLT grammar occupies approximately 510 kilobytes of space on disc, 450 kilobytes when loaded into memory, and so represents each action (shift, reduce or goto) in an average of less than 1.9 bits. On current computer hardware, a table occupying this amount of space makes little impact on parsing system size and can easily be accomodated in main memory, avoiding the performance costs of using swap space. For the ANLT grammar, the representation achieves a similar order of space saving to the *comb vector* representation suggested by Aho, Sethi & Ullman (1986:244ff) for unambiguous grammars (see Klein & Martin (1989) for a survey of representation techniques). In contrast to conventional techniques, though, the parse table is represented absolutely faithfully, in that error entries are not replaced with more convenient non-error ones in order to save extra space. Parsers driven by the table are thus able to detect failures as soon as theoretically possible with the LALR(1) technique. This is an important property for non-deterministic parsing with ambiguous grammars (chapter 5), since many alternative analysis possibilities may be under consideration at any stage, and those which will ultimately fail should do so at the earliest opportunity so as not to consume resources unnecessarily. In addition, when parsing interactively with these tables (chapter 6), early failure is a labour-saving feature.

Table 4.1 compares the size of the LALR(1) parse table for the ANLT grammar with others

[27] Seventy-five percent of the 4,180 states contain at least one action conflict. There are 360,000 shift-reduce conflicts and 430,000 reduce-reduce conflicts, fairly uniformly distributed across the terminal lookahead symbols. In almost two-thirds of the latter conflicts, the rules involved have an identical number of daughters. One implication of this finding is that an approach to conflict resolution such as that of Shieber (1983) where reduce-reduce conflicts are resolved in favour of the longer reduction may not suffice to select a unique analysis for realistic NL grammars.

Grammar	Number of CF rules / categories	Number of LR(0) states	Number of kernel items	Total number of actions
Pascal[28]	158 / 124	275	?	2883
Modula-2[29]	227 / 194	373	420	3238
Wright *et al.*[30]	677/ ?	1011	6167	126322
Tomita, Japanese[31]	800 / ?	?	?	?
ANLT (756 PS rules)	2066 / 567	4180	35455	1916533

Table 4.1. Sizes of grammar and LALR(1) parse tables.

reported in the literature. From these figures, the ANLT grammar is more than twice the size of Tomita's (combined morphological and syntactic) grammar for Japanese (Tomita, 1987:45). The grammar itself is about one order of magnitude bigger than that of a typical programming language, but the LALR(1) parse table, in terms of number of actions, is two orders of magnitude bigger. Although Tomita (1984:357) anticipates LR parsing techniques being applied to large NL grammars written in formalisms such as GPSG, the sizes of parse tables for such grammars grow more rapidly than he predicts. (For unambiguous computer languages, Purdom (1974) states that the space requirements of LR tables tend to grow as only a linear function of grammar size). However, for large, realistic NL grammars such as the ANLT, the table size is still manageable despite Johnson's (1989) worst-case complexity result of the number of LR(0) states being exponential on grammar size[32].

As might be expected, and Table 4.2 illustrates, parse table construction for large grammars is CPU-intensive. As a rough guide, Grosch (1990) quotes LALR(1) table construction for a grammar for Modula-2 taking from about 5 to 50 seconds, so scaling up two orders of magnitude, the timings for table construction for the ANLT grammar fall in the expected region.

Grammar	Backbone computation	LR(0) state construction	lookahead computation	parse table construction
ANLT	140	1100	4900	110

Table 4.2. Timings for LALR(1) parse table construction (in seconds of CPU time on a Sun SparcServer 390 running Sun Common Lisp).

[28] Figures given by Klein & Martin (1989).

[29] Grammar from Spector (1983) with optionality expanded out; statistics taken from a parse table constructed by the system described here.

[30] Figures from Wright, Wrigley & Sharman (1991).

[31] Figures from Tomita (1987:45).

[32] Schabes (1991) describes a variant of LR parse tables in which the number of states is guaranteed to be polynomial even in the worst case. Chapter 5 discusses an implementation of this variant and presents conclusions based on using it with the ANLT grammar.

5. Non-deterministic LR Parsing with Unification Grammars

This chapter discusses algorithms that have been proposed for non-deterministic CF parsing driven by LR and LR-like parse tables, and presents a novel, efficient CF LR parsing algorithm. It goes on to describe the implementations of two unification-based table-driven parsers, and makes some empirical comparisons between them on the basis of their practical performance.

Section 5.1 discusses briefly Lang's (1974) non-deterministic parsing framework, goes on to describe Tomita's (1987) LR parsing algorithm by means of an example, and summarises some recent results which show that the latter algorithm has exponential worst-case time complexity. Section 5.2 describes a parsing algorithm based on Kipps' (1989) non-deterministic LR recogniser, an elegant reformulation of Tomita's algorithm. Modifications to the parsing algorithm are then presented which improve its complexity from exponential to polynomial. In section 5.3, this algorithm is extended to unification grammar (using the methodology for constructing unification-based LR parsers that was proposed in chapter 4), and an implementation is described. The section goes on to describe an implementation of a second unification parser based on an LR-like algorithm of Schabes (1991). The complexities of these two unification-based parsers are analysed, and their practical performance is compared to that of the bottom-up chart parser described in chapter 3.

5.1 Non-deterministic Context-free LR Parsing

5.1.1 Lang's Algorithm

Lang (1974) describes a technique for efficiently driving in pseudo-parallel certain classes of non-deterministic push-down automaton (NPDA; Aho, Sethi & Ullman, 1986); LR parse table construction methods for ambiguous CF grammars can be used to derive such NPDAs. The resulting parser has $O(n^3)$ time and space complexity.

The parsing algorithm is like that of Earley (1970) in that it constructs item sets, $n+1$ for an input of length n; each item consists of a tuple $\langle p, A, i, q, B, j \rangle$ which indicates that the input symbols $x_i \ldots x_j$ have been successfully analysed, the current state being p, top of stack A, and last input symbol x_i, and that B was last on top of the stack when the state was q and x_j was the last input symbol. The NPDA transition function takes the current pushdown state, the symbol

currently on top of the stack, and the next symbol of the input string, popping and/or pushing a single symbol onto the stack, possibly moving to the next input symbol, and then entering a new state. After terminating successfully, the item sets form a type of packed parse forest from which the parses can be extracted.

Although constituting a general theoretical framework for non-deterministic parsing, the formulation presented by Lang is complicated and difficult to understand, and probably for this reason appears to have been little studied subsequently by other researchers. However, it certainly warrants further investigation as a possible technique for implementing predictive parsers to obtain good practical performance.

5.1.2 Tomita's Algorithm

Tomita (1985, 1986, 1987) describes a more intuitively appealing parsing algorithm for CF grammar, in which an LR parse table is used to drive a shift-reduce parser that represents the state of the parse with a *graph-structured stack*, a generalisation of the standard (linear) first-in first-out stack (section 4.1.1). A graph-structured stack is similar to a chart (e.g. Gazdar & Mellish, 1989) in that it is a data structure in which each distinct sub-analysis that has been found is represented only once. With this algorithm, the LR table is allowed to contain multiple entries (i.e. action conflicts), perhaps caused by ambiguities in the grammar. Whenever the parser reaches a state in which there is an action conflict, the stack divides, and branches corresponding to the analyses for each of the actions are pursued, conceptually in parallel. Conversely, if separate analyses end up in the same state at the same point in the input string, the branches of the stack corresponding to the analyses are joined together at that state, and subsequent actions are applied to the single merged analysis, rather than to each branch separately.

A graph-structured stack is a directed acyclic graph, consisting of *vertices* connected together by *arcs*. Each vertex is labelled with a state number, and each arc contains a single analysis, although the analysis might encode several distinct ones by means of local ambiguity packing (section 3.1.3). In this situation, packing must always be possible due to the fact that if several distinct analyses are associated with a single arc, they must all have the same top category, since any rule subsequently applied to the arc to form a higher-level constituent expects a single category to be on the arc (being the same as the corresponding daughter in the rule). Because CF grammar categories are atomic and the test that is applicable for packing is category equality, the analyses will therefore all be packable into a single top-level analysis.

A reduce action with a rule with k daughters, when applied to a vertex v, locates all ancestor vertices at a distance of k arcs from v. For each ancestor a, the action builds a new (packed) analysis containing the rule mother and the sequence of sub-analyses on each path of arcs to a,

and puts this on an (if necessary newly created) arc from *a* to a vertex labelled with the state specified in the goto table. If parsing halts successfully, the analysis on the single arc then in the graph-structured stack will be a packed parse forest, encoding all possible analyses of the input string. The forest will incorporate subtree sharing (section 3.1.3) if, on each reduce action, the new analysis built contains just pointers to (rather than copies of) the sub-analyses on the arcs.

The operation of the algorithm can be illustrated with an example, using grammar G1 from section 4.1 (shown in figure 4.1) and its LALR(1) parse table (same section, figure 4.2), parsing the sentence *I saw a man in the park with a telescope*. The algorithm commences by creating a single vertex labelled with the start state of the parse table, zero. In state zero with the lookahead `ProNP` (the word *I*), the table specifies the action to be a shift to state 2. Figure 5.1a shows the stack after the shift. (Arcs are represented by straight lines, vertices by circles, and analyses by structures of the form `<node category node-list-or-word packed-analyses>`). In state 2 with lookahead `Vt`, the table specifies a reduce action by `NP --> ProNP`; the start vertex is the only ancestor one arc distant, so an arc is created from that vertex to one labelled with state 7 (i.e. the `NP` goto from state zero). The word *saw* is then shifted, resulting in the situation in figure 5.1b. The parse is deterministic until after *the park* is reduced to `NP`, as shown in figure 5.1c (with the internal structure of analyses indicated by parentheses). However, in state 9 with lookahead `P`, there is an action conflict between performing a reduction with `PP --> P NP`, and shifting the preposition to enter state 8. The parser performs both actions, but with the proviso that all reductions are done before any shifts. The reduction makes *in the park* a `PP`, the constituent being put on an arc ending at state 10, and this is followed by a further reduction with `NP --> NP PP` and ancestor vertex labelled with state 11, a goto to state 12, and finally a shift from there to state 8 on the word *with*. The action conflict shift from state 9 to state 8 now takes place. The situation at this point is depicted in figure 5.1d. Note that the two shifts to state 8 have resulted in previously divergent analyses being merged, and that the subtrees analysing *a man*, *in*, *the park* and *with* are shared between analyses (sharing being indicated in the figure by curved lines from parts of analyses).

Next, the words *a* and *telescope* are shifted and reduced to an `NP` on an arc from state 8 to 9. In state 9, with lookahead `$`, the table specifies a reduction with `PP --> P NP`. This time, there are two ancestor vertices two arcs distant: the ones labelled with states 9 and 12 at the start of the arcs containing the node for *with* in figure 5.1d. However, both of these states have gotos to state 10 on `PP`, the rule mother, so the two arcs that result (covering *with a telescope*) starting at the ancestor vertices both end at a single vertex labelled with state 10. A few more reductions to `PP` and `NP` take place, until *a man in the park with a telescope* ends up being analysed as an `NP` in two different ways, due to ambiguity in prepositional phrase attachment. The analyses are both

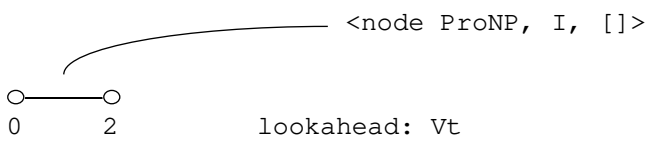

Figure 5.1a. The Stack after Shifting the Word *I*.

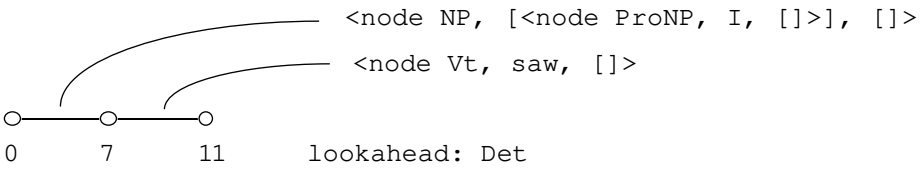

Figure 5.1b. After Shifting the Word *saw*.

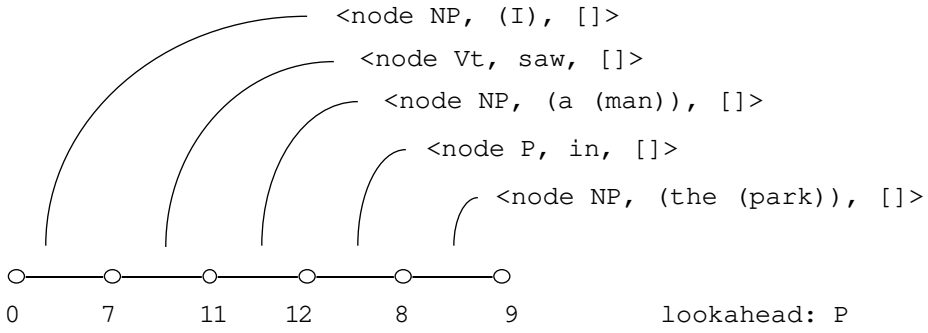

Figure 5.1c. After Reducing *the park* to a Noun Phrase.

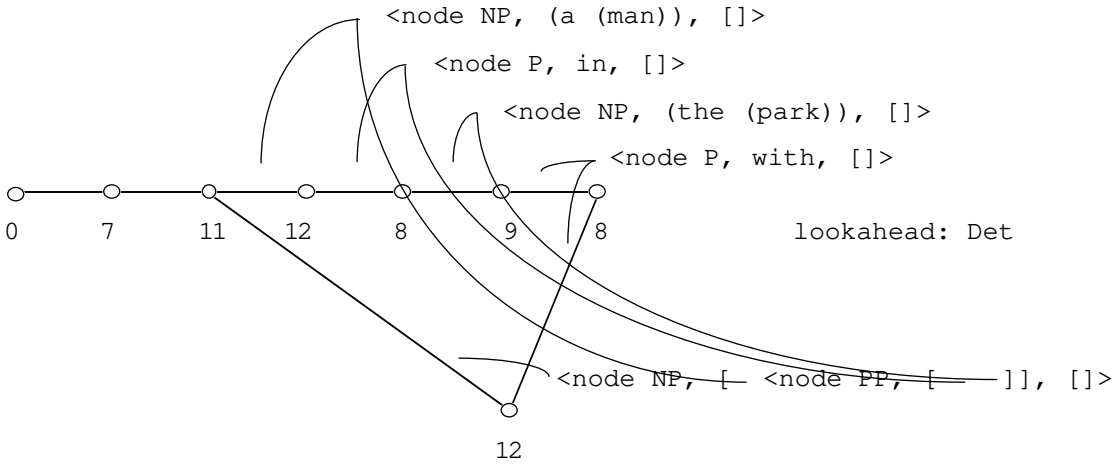

Figure 5.1d. After Shifting the Word *with*.

on an arc from state 11 to 12, and are packed (figure 5.2). Finally, reductions with `VP --> Vt NP`, `S --> NP VP` and `S' --> S` take place, resulting in a stack with a single arc from state zero to 15. The table specifies an accept action in the latter state with lookahead `$`, so the parse halts successfully and the arc contains the complete parse forest.

5.1.3 Complexity of Tomita's Algorithm

Time Complexity

Tomita (1985:761) comments that when parsing an input of length n with a grammar exhibiting "dense ambiguity" (for example the grammar in figure 5.3 which generates the set of strings of the form a^+), his algorithm may require more than $O(n^3)$ time. Johnson (1989) follows up and expands on this remark by describing a sequence of grammars L_m (each member of which generates the same set of strings a^+) such that on sufficiently long inputs the algorithm performs more than $\Omega(n^m)$ operations. L_m for $m > 0$ is shown in figure 5.4, where S^{m+2} abbreviates a sequence of S's of length $m+2$. The result follows from the fact that the number of different sequences of daughter nodes of the topmost `S` node licensed by the third rule in the grammar, `S --> S`$^{m+2}$, for an input string of at least $m+2$ `a`'s is $\Omega(n^m)$. Since Tomita's algorithm would construct each of these sequences explicitly one-by-one in the final reduce action with this rule,the algorithm itself performs more than $\Omega(n^m)$ operations. So, in particular, when parsing

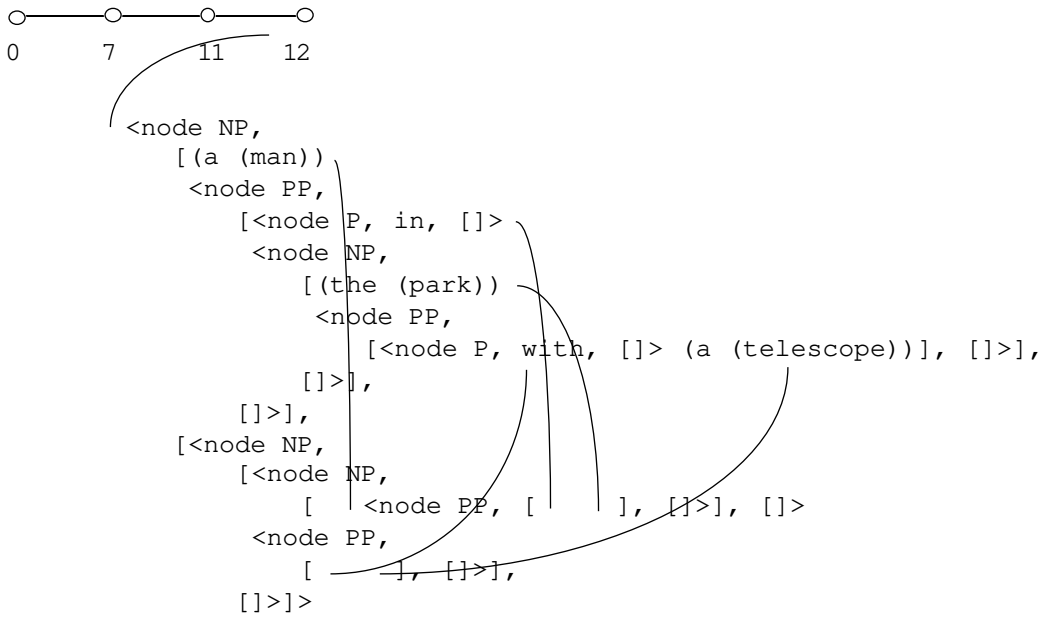

Figure 5.2. The Stack after Reducing the Object Noun Phrase.

```
S --> a
S --> S S
S --> S S S
S --> S S S S
```

Figure 5.3. Tomita's "Densely Ambiguous" Grammar.

```
S --> a
S --> S S
S --> S^{m+2}
```

Figure 5.4. Johnson's Grammar L_m.

with grammar L_4, in which the third rule contains six daughters, the algorithm would have a time complexity of $O(n^4)$.

Kipps (1989) presents a stronger result than this by analysing the actual algorithm in detail and showing that it has a time complexity of $O(n^{\rho+1})$, where ρ is the number of daughters in the longest production in the grammar. The traversal of arcs in the graph-structured stack during a reduce action in order to find ancestor vertices is the dominant factor in the complexity. However, Tomita's algorithm can be modified to improve this complexity: section 5.2 presents a novel version of the algorithm is which is only $O(n^3)$ in the worst case.

Looking at the complexity of Tomita's algorithm with respect to grammar size, Johnson (1989) shows that there are some grammars for which the total number of operations performed is an exponential function of the size of the grammar. The result follows from the fact that for certain (pathological) grammars, for example G_m in figure 5.5 (generating the set of strings a^+), the number of LR(0) states is exponentially related to the grammar size, and that there are some inputs which cause an exponential number of distinct states to be visited in the course of a parse.

```
S --> A_i          0 ≤ i ≤ m
A_i --> B_j A_i    0 ≤ i, j ≤ m, i ≠ j
A_i --> B_j        0 ≤ i, j ≤ m, i ≠ j
B_j --> a          0 ≤ j ≤ m
```

Figure 5.5. Johnson's Grammar G_m.

Space Complexity

Kipps (1989:199) argues that the space requirement for a recogniser based on Tomita's parsing algorithm is proportional to n^2 in the worst case. The argument is based on the fact that the space requirements are dominated by the requirements for the graph-structured stack, the major data-structure used, and that the number of vertices in the stack at each position in the input string is bounded by a constant. However, Kipps does not take into account the fact (pointed out by Johnson, see above) that the number of vertices at each position is dependent on the number of states in the grammar's LR(0) machine, and for grammars such as G_m, for some inputs, the number of vertices may be exponentially related to the size of the grammar. Therefore, the space bound for such a recogniser is n^2, aside from any dependency on the grammar, but for an arbitrary grammar the worst-case bound is exponentially related to the size of the grammar.

Tomita's parsing algorithm requires space both for the graph-structured stack and for the parse forest that it constructs as its result. Again ignoring grammar dependencies, Tomita (1985:762) states that his parse forest representation takes $O(n^3)$ space for "most Natural Language grammars", but can take more than this for grammars with dense ambiguity. For example, the grammar in figure 5.3 takes $O(n^5)$ space. He also claims (1985:762) that although Earley's parse forest representation takes at most $O(n^3)$ space for an arbitrary grammar, the representation is incorrect. (Schabes, personal communication) argues that the Earley parse forest representation is, in fact, correct, and conjectures that Tomita's misconception stems from an incorrect *implementation* by him of Earley's parser.

A further potential source of sub-optimal space usage in Tomita's algorithm is when two arcs cover the same input, contain analyses whose top category is the same grammar symbol, but have distinct start or end states. (This might correspond to the two sets of analyses applying in two different syntactic contexts). In this case, the analyses on the arcs cannot be packed, whereas in an algorithm which did not distinguish context, the analyses would be packed, saving space, and also time in subsequent computation. Billot & Lang (1989) and Lankhorst (1991) observe this phenomenon when parsing using an LR(1) table, leading to parse times with some grammars that are significantly worse than when an LR(0) table is used (or equivalently, SLR(1) or LALR(1) since they have the same number of states). When this happens, packing is observed to decrease in density compared to LR(0) as the ambiguity of an input increases, since ambiguous left context is (often unnecessarily) represented explicitly. The results (for space utilisation) in the experiments reported in section 5.3 below suggest that this type of sub-optimal behaviour is not a problem in LALR(1) parsing with the ANLT grammar.

5.2 A New, Efficient CF LR Parsing Algorithm

5.2.1 The Basic Algorithm

To analyse the time complexity of Tomita's (1987) parsing algorithm (as discussed in the previous section), Kipps (1989) reformulates the algorithm, presenting (an elegant) non-deterministic LR recogniser for CF grammars, and shows that the process of popping states off the graph-structured stack during a reduce action results in a time bound that is exponential on the length of the input. Kipps then goes on to describe a modified version of the algorithm in which certain types of intermediate result are cached in order to make the state-popping process efficient. The modified algorithm has polynomial time complexity (aside from any dependency on grammar size). In this section a novel non-deterministic CF LR *parser* is described, based on (the modified version of) Kipps' recogniser and retaining the recogniser's time complexity bound. The parser uses a graph-structured stack in the same way as Tomita's algorithm, and again like that algorithm, the parser performs sub-analysis sharing (where if two or more trees have a common sub-analysis, that sub-analysis is represented only once), and local ambiguity packing (in which sub-analyses which have the same top node and cover the same input have their top nodes merged, being treated by higher level structures as a single sub-analysis). However, the parser has a polynomial time complexity, improving on the exponential worst-case complexity of Tomita's. The algorithm has not been implemented, but a unification-based non-deterministic LR parser based on it has. This parser is described in section 5.3; it also forms the basis of the probabilistic non-deterministic parser in chapter 6.

First, a preliminary version of the CF parsing algorithm will be described which has the same time complexity as Tomita's algorithm and Kipps' unmodified reformulation (i.e. without any caching). Section 5.2.2 demonstrates that this version of the algorithm can in some circumstances produce spurious duplicate analyses, and describes a modification that remedies the problem. The algorithm is then further modified to cache intermediate results, reducing its complexity to $O(n^3)$.

The algorithm (given formally in figure 5.6 below) takes as input a sequence of words, which is assumed to be non-empty, and returns a list of analyses (each one a complete constituent covering the whole input), forming a packed parse forest. Two global data-structures are used: a list of analyses found, and a vector holding the graph-structured stack (the ith element being a list of all the graph vertices which have incoming arcs representing constituents ending at position i in the input string). A vertex is represented by a structure containing three fields: the position in the string of the end of the incoming arcs, an LR state number, and a list of arcs: i.e. `<vertex position, state, arc-list>`. Each arc is a structure with two components: a start vertex, and an analysis: i.e. `<arc vertex, analysis>`. To simplify the description of the

algorithm, an analysis is represented by a node structure (`<node category node-list-or-word packed-analyses>`) in the CF version. (In the unification-based version of the algorithm described in section 5.3, as in the chart parser described in chapter 3, an analysis consists of a node and a node variable table). A rule is represented by `<rule mother-category daughter-category-list>`.

The algorithm first initialises the zeroth stack element to a list containing a single vertex with the *position* field being zero, the *state* field being the start state in the LR table, and the *arc list* being empty. The rest of the stack elements are initialised to empty, as is the list of analyses. Parsing consists of, for each word in the input at position *i*, retrieving the lexical entries for the word (as a list of categories), initialising a list *Done* (used in the processing of reduce actions) to empty, and performing all shift, reduce and accept actions on vertices in the stack at position *i*-1. If this does not result in any vertices being created in the stack at position *i*, there cannot be any analyses so processing stops immediately. When the end of the input is reached, the shift-reduce-accept process is repeated with a lexical entry list containing the single atom '`$`'. Finally the list of analyses found is returned.

The shift-reduce-accept process consists of two stages. Firstly, for every vertex in the stack at position *i*-1, the vertex is added to the *Done* list, and—in `REDUCE/ACCEPT`—if a reduce action with a given production is specified by the LR parse table, given the vertex state and the lexical entries for the current word, `REDUCE` is called on the vertex and production; if an accept action is specified, the analyses on all the incoming arcs are appended to the list of analyses found. `REDUCE/ACCEPT` is repeated for any new vertices which `REDUCE` adds to the stack at position *i*-1, until quiescence. Secondly, for every vertex in the stack at position *i*-1, if there is a category in the list of lexical entries for the current word for which the LR parse table—in the state of the vertex—specifies a shift action, `SHIFT` is called on the vertex, the state being shifted to, and a partial analysis: a node structure containing the category and the word.

Given a vertex, a state and a (partial) analysis, `SHIFT` puts the analysis on an arc which starts at the given vertex and finishes at a second vertex with the given state at position *i* in the input. It does this by checking to see if there is already an arc between the two vertices; if there is one, the analysis must have the same top category as that already on the arc, and so the two are packed to form a single analysis. Otherwise, if the second vertex is present in the stack, a new arc containing the analysis is constructed between the two vertices. If the second vertex is not present, it is created, added to the stack, and joined to the first vertex with a new arc containing the analysis.

`REDUCE` takes a vertex and a production, and first calls `ANCESTORS` which finds all the ancestors of the vertex at a distance equal to the number of daughters in the production, for each path to an ancestor creating an arc starting at the ancestor and containing an analysis with top category being the production mother, and daughters being the analyses on the arcs forming the

path. REDUCE then iterates over these new arcs, adding each to the stack ending at a vertex at position *i*-1 in the input with state given by the entry for the mother in the goto table of the ancestor's state. To add an arc to the stack, REDUCE checks whether such a vertex already exists. If it does not, the vertex is created and the arc added to its incoming arcs list. Otherwise, if there is also already an arc from the ancestor to it, the analysis on the new arc is simply packed into the analysis on the existing arc. If there is no such arc, the new arc is added to the list of those running between the ancestor and the vertex (at [3] in figure 5.6); additionally, if the vertex is in the *Done* list, the arc may have missed some reductions that were performed on the vertex, so a *clone*, or variant, of the vertex with just the new arc as its single incoming arc is passed to REDUCE/ACCEPT.

Before adding the new arc to those between the ancestor and the vertex (at [3]), a check is made to see if there is already an arc running from a variant of the ancestor to the vertex. If there is, the arc is not added (the variant must have been cloned to form the ancestor). However, in this case the vertex will already have been processed so the arc may have missed some reductions, therefore a clone of the vertex with just the new arc as its single incoming arc is passed to REDUCE/ACCEPT.

In the formal definition of the algorithm just described, in figure 5.6, the object following a '|' in a list is itself a list which contains all the elements except the first, and '$|x|$' returns the number of elements in the list x. Objects which are not relevant to processing are denoted by '_', lists are enclosed in square brackets, and sets in braces.

5.2.2 Modifications to the Algorithm for Correctness and Efficiency

Correctness Modifications

The processing steps in the parsing algorithm described in the previous section correspond closely to those in the recogniser presented by Kipps; however the algorithm is incorrect at one point. At [3], an arc is created which finishes at vertex *v"*, and if *v"* is in the list *Done*, the vertex is cloned with the arc as its single incoming arc, and REDUCE/ACCEPT invoked. But if *v"* is the vertex currently being processed (i.e. it is the vertex which was last added to *Done*), any reductions not yet done on *v"* will be performed twice on the arc: once when it is an incoming arc of *v"*, and once when it is the incoming arc of the clone vertex. This may result in spurious duplication of analyses. To fix this problem, at [3], an arc must not be immediately inserted into the stack if *v"* is currently being processed. Instead, the insertion should be delayed until the processing is finished. The modifications detailed in figure 5.7 to the algorithm of figure 5.6 accomplish this.

```
PARSE(w_1 ··· w_n)
   let U_0 := [<vertex 0, StartState, []>], U_1...n := [], Analyses := []
   for i from 1 to n+1
      let Done := [],
          c_i := if i = n+1 then [`$'] else LEXICAL-ENTRIES(w_i)
      until no new vertices in U_i-1
         for ∀new v ∈ U_i-1 , Done := [v | Done], REDUCE/ACCEPT(v)      [1]
      for ∀v = <vertex _, s, _> ∈ U_i-1
         for ∀c ∈ c_i
            if ∃`shift s'' ∈ ACTIONS(s, c) then
               SHIFT(v, s', <node c, w_i, []>)
      if U_i = [] then return []
   return Analyses

REDUCE/ACCEPT(v = <vertex _, s, l>)
   for ∀`reduce p' ∈ ACTIONS(s, c_i), REDUCE(v, p)
   if `acc' ∈ ACTIONS(s, c_i) then
      ∀<arc _, a> ∈ l, Analyses := [a | Analyses]

SHIFT(v, s, a)
   if ∃v' = <vertex _, s, l> ∈ U_i then
      if ∃v'' = <arc v, <node _, _, p>> ∈ l then p := [a | p]
      else l := [<arc v, a> | l]
   else U_i := [<vertex i, s, [<arc v, a>]> | U_i]

REDUCE(v, p = <rule m, ds>)
   for ∀arc = <arc v_1' = <vertex j', s', _>, a> ∈ ANCESTORS(v, |ds|, {[]}, m)
      let s'' := GOTO(s', m)
      if ∃v'' = <vertex i-1, s'', l''> ∈ U_i-1 then
         if ∃<arc v_1', <node _, _, p>> ∈ l'' then p := [a | p]        [2]
         else
            if ∃<arc v_2' = <vertex j', s', _>, _> ∈ l'' then
               REDUCE/ACCEPT(<vertex i-1, s'', [arc]>)
            else l'' := [arc | l'']                                    [3]
               if v'' ∈ Done then REDUCE/ACCEPT(<vertex i-1, s'', [arc]>)
      else U_i-1 := [<vertex i-1, s'', [arc]> | U_i-1 ]

ANCESTORS(v = <vertex _, _, l>, k, alts, m)
   if k = 0 then
      return {<arc v, <node m, al, []>>, al ∈ alts}
   else return
      ∪_<arc v', a'>∈l ANCESTORS(v', k-1, {[a' | al], al ∈ alts}, m)
```

Figure 5.6. The Non-deterministic CF LR Parsing Algorithm—Preliminary Version.

```
replace line at [1] by:
       for ∀new v = <vertex _, _, l> ∈ U_{i-1}
          let ToAdd := []
          Done := [v | Done], REDUCE/ACCEPT(v)
          for ∀<arc v', a'> ∈ ToAdd
             if ∃v" = <arc v', <node _, _, p>> ∈ l then p := [a' | p]
             else l := [<arc v', a'> | l]

replace line at [3] by:
          else
             if Done = [v" | _] then ToAdd := [arc | ToAdd]
             else l" := [arc | l"]
```

Figure 5.7. Correctness Modifications to the Preliminary Parsing Algorithm.

In the modified version, at [3], if *v"* was the last vertex that was added to *Done*, the arc is added to a list, *ToAdd*. At [1], *ToAdd* is initialised to empty, the vertex is processed, and when processing is complete each arc in *ToAdd* is added as a incoming arc to the vertex, or if such an arc already exists then the analysis on the arc is packed into the one already there.

Efficiency Modifications

Kipps (1989) shows that the time complexity of his recognition algorithm is $n^{\rho+1}$ for an arbitrary grammar. The complexity is dominated by the number of steps taken by his REDUCE function by virtue of its invocation of ANCESTORS: $\sim i^{\rho}$ in the worst case, where i is the current position in the input string, and ρ, the maximum number of daughters in any production. The parsing algorithm presented above may be analysed in a similar way and shown to have the same complexity. Kipps (1989:198) goes on to describe how the recogniser's ANCESTORS can be turned into effectively a table look-up function, changing the complexity of the recogniser to n^3. This section describes how a novel realisation of this technique, in conjunction with a change in the representation of analyses, can also be applied to the parsing algorithm of the previous section to produce a new algorithm whose complexity is only n^3.

In the parsing algorithm, when ANCESTORS is applied to a vertex in the course of a reduce action, the function returns one arc for every path by which the vertex can be reached from any ancestor k (the number of daughters in the reducing production) arcs distant. Each arc contains an analysis whose top category is the mother of the production, and daughters, a list of the analyses on the arcs forming the path. The number of paths is bounded by the number of ways of partitioning the input symbols occurring between the ancestor and vertex positions among the k

symbols on the right hand side of the production. For an ancestor at position *j*, the number of paths (or equivalently, arcs) is bounded by

$$\sum_{0}^{m_1 = j} \sum_{0}^{m_2 = m_1} \cdots \sum_{0}^{m_{k-1} = m_{k-2}} 1.$$

Summing this expression for *j* from zero to *i* gives the result i^k. `REDUCE` processes each one of these arcs separately, so the worst-case complexity of each invocation of this function is at least as bad as $\sim i^k$. Therefore, just modifying `ANCESTORS` to perform a table look-up cannot improve the complexity of the algorithm: the bound on the number of arcs returned by `ANCESTORS` must be improved as well. This can be achieved by changing the representation of a node so that instead of a list of nodes as the second component, a node contains a list of *node sequence alternatives*, where a node sequence is defined as being either the empty list, or a list whose first element is a node, and the rest of whose elements are alternative node sequences. With the change, `ANCESTORS` need return only one arc for each distinct ancestor vertex at distance *k*, since all possible sequences of nodes lying on alternative paths to the ancestor can be encoded in a single list of node sequence alternatives held in the arc. Figure 5.8 gives an example of the result of a call to `ANCESTORS` when reducing the verb phrase *saw the man in the park with a telescope* using the production `VP --> V NP PP`. (Note that only one arc is returned, even though there are two paths from the vertex `x` to `y`) . Of course, with the new representation for nodes, the final parse forest created by the parser is represented differently; however, the new and old representations are completely equivalent.

Now that `ANCESTORS` returns a single arc for each ancestor vertex in its result (making a total of *i* arcs), `REDUCE` executes in $\sim i$ steps, excluding the time for `ANCESTORS` . However,

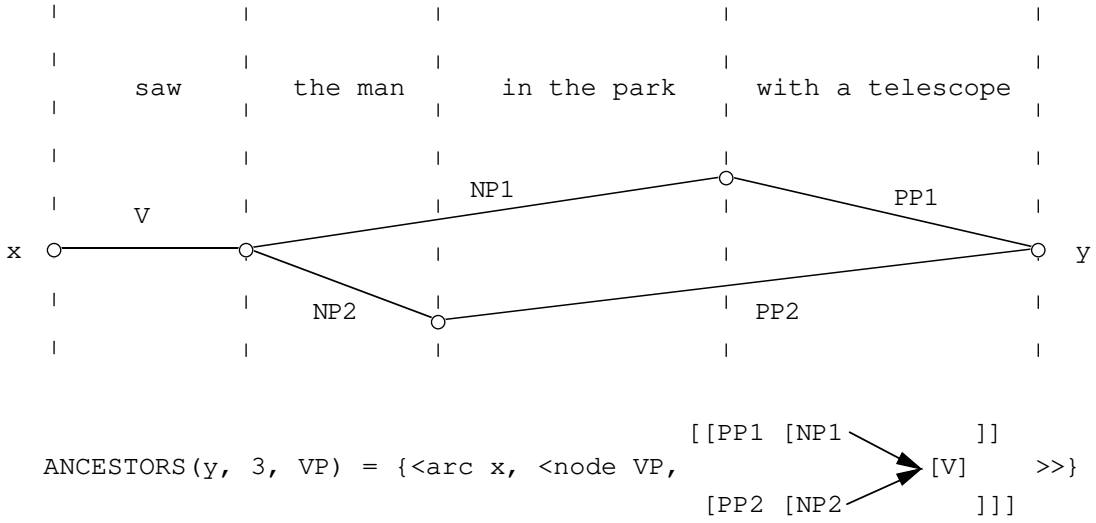

Figure 5.8. Node Sequence Alternatives in Arcs Returned by `ANCESTORS`.

ANCESTORS still has a time bound of $\sim i^{\rho}$ since on each invocation it follows all paths to ancestors, in the process potentially recursing down a particular segment of a path more than once. To eliminate this redundancy, the second modification is to introduce a cache (with constant-time access and update properties) keyed firstly on vertices in the graph-structured stack, and secondly on integers from 1 to ρ [33]. Each cache entry keyed on v and k contains a list of arcs representing all the possible node sequence alternatives whose start vertex is a distance k constituents in the stack from vertex v.

All cache elements are initially empty. In the modified version of the algorithm, the procedure ARC-ADDED (figure 5.9) is invoked to update the cache (held in *Cache*) before any further processing takes place every time the algorithm adds a new arc to an existing list of arcs, or creates a new vertex with an incoming arc. (This happens at two places in SHIFT, and four in REDUCE). ARC-ADDED takes the vertex, v, at which the arc finishes, and the arc itself (with start vertex v'). First, it adds to the existing elements of the cache, keyed on (v, 1), a version of the arc with its analysis in the form of a list of node sequence alternatives. Next, the function updates the cache elements (v, k) for all integer values of k from 2 up to ρ. It does this by considering all the cache entries (v', k-1), and extending each of them with the analysis in the new arc. To incorporate the analysis into the cache, ARC-ADDED checks whether there is already an entry in (v, k) with the same start vertex as an entry being extended; if so, the (v, k) entry's node sequence alternatives are augmented by a sequence consisting of those of the entry being extended with

```
ARC-ADDED(v, arc = <arc v', a'>)
    Cache_v,1 := [<arc v', [a']> | Cache_v,1]
    for k = 2 to ρ,
        ∀<arc v", <node m", alts, _>> ∈ Cache_v',k-1,
            if ∃<arc v", <node _, alts", _>> ∈ Cache_v,k then    [1]
                alts" := [[a' | alts] | alts"]
            else
                Cache_v,k :=
                    [<arc v", <node m", [[a' | alts]], []>> | Cache_v,k]

ANCESTORS(v = <vertex _, _, l>, k, m)
    if k = 0 then
        return {<arc v, <node m, [[]], []>>}
    else
        return
        {<arc v', <node m, alts', p'>>,
            ∀<arc v', <node _, alts', p'>> ∈ Cache_v,k}
```

Figure 5.9. Efficiency Modifications to the Preliminary Parsing Algorithm.

[33] The cache could be implemented straightforwardly as a three-dimensional array with indices being vertex state, vertex string position, and integers from 1 to ρ.

the addition of the new arc analysis, otherwise a new entry is added to (v, k) with this same sequence as the only alternative.

The cache is always up-to-date since the entries in it are affected only by the addition and deletion of arcs in the stack, and by analyses being packed into existing ones; the algorithm never deletes arcs, `ARC-ADDED` is invoked to update the cache immediately after every addition of an arc, and since a node that is packed into will be present in the cache, when packing occurs the cache is updated automatically as a side-effect. Because the cache always contains the correct lists of ancestor vertices (and associated node sequence alternatives) at every distance k ($1 \leq k \leq \rho$) from any vertex, the new version of `ANCESTORS` (figure 5.9) needs to do little more than a table-lookup (and no longer being recursive does not now need to take the third argument, *`alts`*). When k is equal to zero, the cache is not consulted, the function returning a set containing a single arc starting at v itself, with an analysis with top node being the production mother, m, and a single node sequence alternative consisting of the empty list. When k is greater then zero, copies of all the cache entries keyed on (v, k) are returned, with m assigned as the top node in each analysis.

In practice, with a grammar containing mostly unary and binary branching rules, the overheads associated with introducing caching may negate any gains accruing from no longer having to traverse long paths to construct lists of ancestors. The question of whether caching improves parsing performance for a particular grammar would have to be investigated empirically.

5.2.3 Complexity of the Modified Algorithm

Time Complexity

The modifications described in the previous section to the preliminary version of the parsing algorithm serve to reduce its time complexity in the general case to $O(n^3)$. This section derives the complexity result. Kipps (1989:197) makes two important observations that form the basis for the complexity analysis: the number of vertices in any U_i is bounded (by the number of states in the parse table), and the number of parents of a vertex in U_i is proportional to i. These observations also hold for this parser.

In the `PARSE` function, for each word in the input, `SHIFT` is called a bounded number of times (with an upper limit given by the result of multiplying the number of lexical entries for the word by the number of vertices in U_i). Since the number of vertices in U_i is bounded, `SHIFT` itself executes in a bounded number of steps, calling `ARC-ADDED` once. `PARSE` also calls `REDUCE/ACCEPT`, once for each vertex in U_i, so the number of calls is again bounded. For each

word w_i, the number of top level calls to REDUCE is bounded by the number of lexical entries for the word multiplied by the number of productions in the grammar. The recursive call to REDUCE (via REDUCE/ACCEPT) is performed no more than $\sim i$ times; this is because REDUCE/ACCEPT can be called no more than once for each parent of a vertex in U_i, of which there are at most $\sim i$.

On each call to REDUCE, ANCESTORS is called once, returning $\sim i$ arcs, and REDUCE iterates over each one. Each iteration invokes ARC-ADDED up to twice, and can give rise to a single recursive call to REDUCE on a clone vertex. In such a recursive call, on each iteration over the clone's $\sim i$ ancestors no further recursion is possible, and furthermore, no new vertices can be created in the stack since any that would normally have been created will already exist from the processing of the vertex of which the clone is a variant. Therefore, only packing into existing analyses will occur, so ARC-ADDED will not be invoked during the recursion. ARC-ADDED takes time proportional to i, for an arc with start vertex v' at position i in the input, since there are $\sim i$ entries in each cache location. (This time bound assumes that arcs in $Cache_{v,k}$ are indexed on their start vertex so that the test in ARC-ADDED at [1] in figure 5.9 can be performed in unit time). Since it merely performs a table lookup, ANCESTORS also executes in $\sim i$ time when called on a vertex at position i.

The worst case time bound is dominated by the time taken by REDUCE . The function is called a bounded number of times at top level, and each such call on a vertex at position i takes up to time $\sim i^2$ if the function recurses on a clone vertex. Summing for i from zero to n+1 gives a time bound proportional to n^3 for the algorithm[34].

Space Complexity

In the modified parsing algorithm, the graph-structured stack takes space proportional to n^2 in the worst-case (by the same reasoning as in section 5.1.2), aside from grammar dependencies. The parse forest incorporates subtree sharing and local ambiguity packing, and, in addition, sequences of constituents are shared between analyses in cases where more than one reduction takes place at a single vertex in the graph-structured stack, or paths to ancestor vertices split. The shared constituent sequences are exactly the node sequence alternatives which were introduced in section 5.2.2, corresponding to the rightmost daughters of rules (compare with the leftmost daughter sharing in the chart parser, discussed in section 3.2.1). By a similar argument to that in section 3.2.1, invoking Billot & Lang's (1989) grammar of parses, the space complexity of the parse forest is $O(n^3)$ for an arbitrary grammar.

[34] Note that the caching obviates the requirement to completely enumerate all possible node sequences in a reduce action; the sequences are thus effectively 'packed'. Complete enumeration of all of the sequences, as in Tomita's algorithm, leads to Johnson's (1989) lower bound of $\Omega(n^m)$ operations for the grammar L_m in figure 5.4 (see section 5.1.2).

However, this parse forest complexity analysis ignores the situation where analyses with the same top category and spanning the same segment of the input string, which would normally be packed, in fact appear on distinct arcs due to the states involved encoding contextual distinctions (Billot & Lang, 1989:146). The frequency with which this situation can occur is related to the number of places in the parse table at which each rule is recognised, and therefore on the amount of ambiguity in the grammar and on the table construction method used. It is not clear whether this worsens the $O(n^3)$ space complexity of the parse forest, but nevertheless the complexity is unaffected by dense ambiguity in the same way as is Tomita's algorithm. However, as in Tomita's algorithm, the space taken by the graph-structured stack is the dominant term for the algorithm as a whole when grammar dependencies are taken into account.

5.2.4 Other Efficient Non-deterministic CF LR-like Algorithms

Leermakers (1991) describes a deterministic LR recognition algorithm, which, instead of being expressed in terms of a finite state automaton, represents each state in the canonical LR(0) collection as an executable procedure. During the course of a parse, procedures that are currently being run are held on the main execution stack, which serves the same purpose as the stack that is explicitly maintained by a shift-reduce parser. Applying the algorithm to non-LR (e.g. ambiguous) grammars results in exponential time complexity, but if the procedures are implemented as *memo-functions*, the complexity with respect to input length is reduced to $O(n^3)$. (A memo-function is a procedure such that if it is called with the same arguments as in a previous invocation, it returns the previous result without recomputing it). The technique used to attain cubic complexity in the parsing algorithm presented in this chapter is closely related, in that it employs caching to avoid the `ANCESTORS` procedure having to repeatedly compute the same lists of arcs. The two techniques also share the property of being exponential on grammar size, due to the use of the LR(0) state collection.

Nederhof (1992) casts left-corner (LC) parsing in a similar framework to that of Tomita's generalised LR technique. A graph-structured stack is used, the parser works breadth-first, and the output is a packed parse forest, but containing additional subtree sharing in which sequences of constituents are shared between analyses (as in the algorithm described in this chapter). Nederhof's parser has cubic time and space complexity for an arbitrary CF grammar, but has advantages over LR, the most important being the fact that the size of the parse table required is only quadratic in the size of the grammar, and can be can be generated relatively quickly and efficiently. This would make the technique suitable for use during grammar development, whereas LR is not. Moreover, the parse forests created are as compact as possible, in contrast to the LR technique where space utilisation can be non-optimal, as discussed in section 5.2.3 above.

Nevertheless, the LR technique is more powerful, in that it produces more predictive parse tables, and for this reason "generalised LR parsing is more efficient than generalised LC parsing" (Nederhof, 1992:9). However, an empirical comparison between the performance of parsers using the two techniques would be interesting.

Schabes (1991) describes an algorithm for constructing a predictive parsing table for a CF grammar, and an algorithm for a non-deterministic chart-based parser driven by such a table. The table compiles out the predictive component of Earley's (1970) parser, and has cubic time complexity. Schabes' technique is discussed further in section 5.3.2 below, and evaluated empirically in section 5.3.4.

5.3 Non-deterministic Unification-based LR Parsing

In section 4.2, an approach to LR parsing with a unification grammar was outlined, involving the initial automatic derivation of an atomic-categoried CF 'backbone' from the grammar and construction of an LR parse table for this backbone. A parser would be driven by the table, but on reductions it would additionally perform the unifications specified in the unification grammar rules to form feature-based categories representing the higher-level constituents (and also abandon parts of derivations if the unifications concerned were to fail). This section describes how the algorithm presented above for a non-deterministic CF LR parser may be adapted and augmented to form a non-deterministic unification-based LR parser. The section also discusses issues raised by an implementation of this adapted algorithm, analyses its complexity, and makes an empirical comparison of its performance relative to an implementation of a unification-based version of a second, predictive LR-like parser that has been proposed in the literature as an alternative type of efficient parser.

5.3.1 Extending the CF Parsing Algorithm with Unification

The non-deterministic unification-based LR parser described here is an extension of the preliminary version of the CF LR parsing algorithm including the correctness modifications of section 5.2.2. (The efficiency modifications cannot be incorporated into the unification-based parser for reasons which are explained at the end of the section). An implementation of the parser forms part of the Alvey Natural Language Tools (chapter 2). It appears that no other non-deterministic unification-based LR parser has previously been described in the literature, and that no other comparable implementation exists.

The basic control structure of the unification parser is identical to that of the CF parser: the

parser is driven in the same way by an LR parse table, though the table will have been constructed from the backbone grammar. Most of the differences between the CF and unification-based algorithms are due to:

(1) having to perform unifications on each reduce action: no equivalent operation is needed in the CF algorithm since the constituent top categories of each in every sequence of reduced constituents by a production are guaranteed to be equal to the production's daughters, and

(2) feature-based category subsumption (section 2.4.3) being required as a check before packing: no equivalent operation is needed in the CF algorithm since all analyses on a given arc will automatically have the same (atomic) top category.

In the CF LR algorithm, an arc contains two fields: a start vertex and an analysis. The analysis field may implicitly contain several distinct analyses, but they will all be encoded as a single analysis through packing. In the unification-based version, an arc must be able to contain an explicit list of analyses, since it is possible in the check before packing that is now necessary (see (2) above) for the top nodes of two constituents on the same arc (nodes which therefore correspond to the same atomic category) not to be related by subsumption. Therefore, in this parser, arcs must be structures of the form `<arc vertex, analysis-list>`, and the algorithm modified accordingly to iterate over the second field where appropriate in `SHIFT`, `REDUCE` and `ANCESTORS`. Thus, variables in the algorithm called *a* (representing a single analysis) would be better named *as* (a list of analyses) in the unification version. In `REDUCE/ACCEPT`, an accept action should result in the list of constituents being appended to the list of analyses found. Each analysis now needs to have a variable table associated with it (section 2.4), so an analysis can no longer simply be a node record, but must be an analysis record, i.e. `<analysis node nvt>`. In `PARSE`, the set of backbone grammar categories corresponding to each lexical entry (which is now a feature-based category) must be determined. (The set usually contains only one element, but in general it may contain more than one if a lexical entry generalises two or more backbone grammar categories). In the implementation, this task is performed efficiently by using a vector of discrimination nets of feature values, similarly to the way rule retrieval is handled in the chart parser.

An additional complication, caused by not always being able to pack together all analyses on the same arc, is that when a new analysis is added to an arc and the analysis cannot be packed, the analysis may miss some reductions if the end vertex of the arc has already been processed, i.e. the vertex is a member of *Done*. Care must also be taken in the case where the vertex is the one currently being processed to ensure that reductions not yet done on it will not be performed twice on the arc (a similar situation to that dealt with by the correctness modification of section

5.2.2). Figure 5.10 shows a modification to the algorithm of figure 5.6 that gives the correct behaviour. (The operator '$\circ$' denotes append). The function PACK-ANALYSES that is invoked takes two lists of analyses, packs each analysis in the second list that can be into one in the first list, and returns the ones that cannot be packed. PACK-ANALYSES must also be invoked by SHIFT and by the modification in figure 5.7 at the points where they perform packing.

In Tomita's system, the value of an attribute in a CF rule augmentation may not always be uniquely determined, leading to the need sometimes to unpack analyses during construction of the parse forest. The system described here does not suffer from this potential source of inefficiency, since in it, a unification category feature always has a unique value (either a category, an atomic value, or an unbound variable).

On each reduce action, the parser must perform the unifications required by the unification grammar rule that corresponds to the backbone rule involved, and create one or more new constituents whose categories are instantiated versions of the unification rule mother. The parser carries out these tasks in the ANCESTORS function. For each arc that is traversed, the function creates a new set of alternative partial analyses by taking each constituent held in the arc and extending each partial analysis in the current set with the result of unifying the top category of the constituent with the category of the corresponding daughter in the unification grammar rule.(The unification algorithm is described in section 2.4). A partial analysis consists of a list of constituents together with a pair of variable tables, one holding values of variables in rule categories and the other, variables in the analyses on the arcs. If a unification fails, the constituent concerned is not added to any of the partial analyses, and if the unifications for all the constituents in an arc fail, further arcs are not traversed and the set of partial analyses concerned is discarded. When all the ancestors at a distance equal to the number of daughters in the rule have been reached, a new arc is created starting at each ancestor, as before, with the arc's analysis lists made up of new analysis structures created from the partial analyses. The node field in each analysis is filled with a node whose category is the result of COPY-CATEGORY (section 2.4.3) applied to the rule mother and rule category variable table, and whose node-list field is the

```
replace line at [2] by:
          let nas := PACK-ANALYSES(as', as)
          if nas then
             if Done = [v" | _] then
                ToAdd := [<arc v1', nas> | ToAdd]
             else as' := nas ∘ as'
             if v" ∈ Done then
                REDUCE/ACCEPT(<vertex i-1, s", [<arc v1', nas>])
```

Figure 5.10. The Packing Test in REDUCE for the Unification-based Algorithm.

list of arc categories in a partial analysis. The partial analysis category variable table becomes the value of the variable table field of the analysis structure.

The efficiency modifications to the CF algorithm cannot be applied to the unification-based version, since they rely on the observation that the analyses on the arcs (except for the top categories in them) which are returned by `ANCESTORS` will always be the same for any call to the function on a given vertex and with any production having a particular number of daughters. However, in a unification-based system, the daughters have to be unified with sub-analyses on the stack, and in general, both the categories and the variable tables will be different on each call. Therefore, the results are likely to be different each time.

The initial implementation of the unification LR parser, as described above, turned out to be much slower than expected. Investigation showed that it was often the case that the same unifications were being performed repeatedly, even during the course of a single reduce action. The duplication was happening in cases where distinct arcs in the stack contained identical feature-based categories, and these arcs were traversed by `ANCESTORS` in the course of a reduction by a single unification grammar rule. In some cases this was due to the fact that the backbone grammar is less constraining than the unification grammar, and in others, because some unification grammar rules map into more than one backbone rule. By caching lists of partial analyses (and thus unification results) keyed on arc analysis, backbone rule daughter, and variable tables, and ensuring fast access to cache entries, this wasted effort was eliminated and the speed of the parser improved by a factor of about three, at little space cost. The practical performance of this parser is discussed in section 5.3.4 below.

5.3.2 Schabes' LR-like Parser

Schabes (1991) describes an algorithm for constructing a predictive parsing table for a CF grammar, and an algorithm for a left-to-right non-deterministic parser driven by such a table. The table compiles out the predictive component of Earley's (1970) parser, and Schabes demonstrates that his parser always takes fewer steps than Earley's, although its time complexity is the same: $O(n^3)$. The table construction method used ensures that the number of states in the parse table is linearly related to the size of the grammar, resulting in the number of operations performed by the parser being at worst a polynomial function of grammar size. The space complexity is also $O(n^3)$, since the parser uses Earley's representation of parse forests. Schabes (1991:107) claims that an experiment carried out by Tomita, the results of which heavily favour the latter's CF parser over an implementation of Earley's parser, "shows no conclusive outcome" when replicated. Further on in the same paper, Schabes also attributes to Kipps the statement that the modified, $O(n^3)$, version of the Kipps recogniser "does not lead to a practical parser" (although

no such statement is made in the Kipps (1989) paper he refers to). On the basis of these claims and the demonstrated time complexity, the techniques that Schabes describes deserve practical investigation, and so a parser was implemented based on them, but adapted for unification-based parsing. This parser is described below. In the next two sections, the theoretical and practical aspects of the parser are compared with the LR parser of the previous section.

The construction of a unification parser based on Schabes' techniques follows the methodology developed for unification-based LR parsing described in chapter 4 and section 5.3.1 above: a table is computed from a CF 'backbone', and a parser, augmented with unification and feature-based subsumption operations, is driven by the table.

The states in the parse table that are built for Schabes' parser differ from the standard LR(0) collection in that each state contains exactly one kernel item (whereas in LR(0) a state can contain more than one), and the *goto* function is split into two functions $goto_k(state, X)$ and $goto_{nk}(state, X)$ to distinguish transitions on symbols from kernel and non-kernel items in the state. Schabes describes a method for constructing the states in the parse table, and the unification parsing system implements this method. However, in order to allow the unification-based parser to be compared fairly with the one described in the previous section, one symbol of lookahead is added to the table by applying Kristensen & Madsen's (1981) LALR construction technique[35] (see section 4.2.3 for a description of this technique), modified slightly to take the goto state sets as being the union of the sets of kernel and non-kernel goto transition states. The parse table that results for the ANLT grammar would occupy a large amount of storage if the actions in it were not represented using appropriate data structures[36]. Therefore, to hold the LALR(1)-like tables that are built, the system uses a variant of the technique described in section 4.2.3 for compactly representing standard LR(1) tables for ambiguous grammars.

Schabes describes an algorithm for a non-deterministic chart-based shift-reduce CF recogniser driven by the parse table, and explains how this can be turned into a parser constructing an Earley-like parse forest representation by, for a shift action, including a pointer from the item added to the chart to the one which caused the shift to take place, and for a reduce action, from the item added to the chart to the (in this case) two items causing it. This representation is identical to that produced by the node sequence alternatives of the $O(n^3)$ non-deterministic CF LR parsing algorithm in section 5.2. For the shift and reduce actions to be implemented efficiently in this parser, and the $O(n^3)$ time bound to be attained, it must be possible to retrieve in unit time all items in the chart having a given state, start and end position in the input string.

In the implementation of the parser, an item in the chart has four fields: a state, a set of node

[35] The successful application of this technique supports Schabes' (1991:109) assertion that "several other methods (such as LR(k)-like and SLR(k)-like) can also be used for constructing the parsing tables [...]".

[36] The Schabes parse table for the ANLT grammar contains approximately 4,800 states, with a total of 1,360,000 (shift, reduce and goto) actions.

sequence alternatives, and a start and end position. The parser works similarly to the CF version that Schabes describes, except that on each reduction of an item with a (backbone) production, the daughters of the unification rule that corresponds to it are unified with each alternative sequence of constituents in the item. Each analysis that results is, if possible, packed into one computed previously. Representing the chart as a three-dimensional array indexed on state, and start and end positions would guarantee constant-time access to items, as required for the theoretical cubic time bound. However, for say a ten word sentence with the ANLT grammar, such an array would contain almost half a million entries (and on typical computer hardware, would consume of the order of two megabytes of memory even when empty). For this reason, the implementation trades off a large reduction in space requirements for a moderate increase in CPU time by employing a sparse representation to store the array (since in practice only a small proportion of the possible entries are ever filled). In this parser, the same sort of duplication of unifications occurs as in the LR parser, so lists of partial analyses (and thus unification results) are cached in the same way.

The parser works quite successfully (section 5.3.4 gives an evaluation of its practical performance), demonstrating that the methodology for building unification-based parsers from a backbone CF grammar, compiling a parse table from the backbone, and performing unifications at parse time is generally applicable and is not restricted just to standard LR techniques.

5.3.3 Complexity of Unification Versions of LR and Schabes' Parsers

The time complexity of both the LR CF parser presented in section 5.2 and the LR-like parser described by Schabes is $O(n^3)$, aside from any dependency on the grammar. However, the unification versions of the LR and LR-like parsers both turn out to have time bounds that are greater than $O(n^3)$ in the worst case. (This phenomenon is analogous to the increase in complexity of the bottom-up chart parser of chapter 3 when it was extended from context-free to unification grammar).

In both the LR and Schabes LR-like parsers, on each reduce action the daughters of the rule involved have to be unified with every possible alternative sequence of the sub-analyses that are being consumed by the rule. (No analogous operation is necessary in a CF parser). If the rule contains ρ daughters, and the sub-analyses span k words in the input string, then there are $O(k^{\rho-1})$ such sequences (i.e. the number of ways of partitioning a line containing k intervals into ρ segments), and so this is the worst-case bound for the number of unification operations necessary in a single reduction. In the LR parser, `REDUCE` is called a bounded number of times at each position i in the input string, so summing for k from zero to i, and i from zero to $n+1$ gives a bound of $O(n^{\rho+1})$ on the total number of unifications. In the LR-like parser, the reduction

operation may be applied a bounded number of times to each item in the chart spanning positions i to j ($i \leq j$). This leads to the same summation, giving the bound of $O(n^{\rho+1})$ for this parser as well. Intuitively, one can think of the unification versions of the parsers in effect expanding and flattening out the packed sequences of sub-analyses on each reduction, nullifying any useful efficiency properties the representation might have.

The two parsers cache the results of unifications, but this can only give an improvement of at most a constant factor, since cached results serve only to decrease the amount of processing performed by the second and subsequent reductions over the same segment of the input; the complexity bounds still stand even if just a single reduction is performed for each segment. Thus, the worst-case time complexity of the LR and LR-like unification-based parsers is exponential in sentence length. Although this theoretical bound might suggest that parsing performance would be severely degraded on long sentences, this does not seem to be borne out in practice: with the ANLT grammar, the LR parser implementation is able to compute parse forests for sentences of up to around thirty words in length and having many thousands of analyses.

5.3.4 Performance Comparison of Unification-based Chart and LR Parsers

LR parsing has been advocated as a suitable technology for implementing Natural Language parsers in order to obtain good practical performance (Tomita, 1985; Wright, Wrigley & Sharman, 1991). To assess how well LR parsing meets this goal when parsing with unification grammars, an experiment was carried out, using the ANLT grammar, to compare the performance of the two predictive LR-based parsers described in this chapter to that of the ANLT bottom-up chart parser (shown in chapter 3 to attain state-of-the-art performance). Before performing the experiment, the implementations of the three parsers were checked carefully to ensure that they were all of a similar high standard[37]. Similar implementation techniques are used in all the parsers; in addition, the parsers all share the same unification module, and they have all been profiled with the same tools and optimised to a similar extent. Both of the predictive parsers employ one symbol of lookahead, incorporated into the parsing tables by the LALR technique.

The experiment used the sentences from the four test sets of chapter 3, a total of 129 sentences, with a mean length of 6.7 words. For each of the three parsers, the CPU time and storage allocated while parsing the sets of sentences were added up. Table 5.1 gives a summary of the results. (The time for the chart parser is more than double the corresponding figure given in section 3.4, since this experiment was not run on a Sun Sparc ELC workstation, but on a

[37] In Tomita's (1985) experiments, the relative qualities of the implementations of his non-deterministic LR parser and his version of an Earley parser are open to question in the light of Schabes' (1991:107) comments that when the experiments were duplicated, the results did not support Tomita's conclusions.

	CPU time	Storage allocated
Bottom-up chart parser	177.9	47.0
LALR(1) parser	113.3	33.6
Schabes-type parser	227.3	38.5

Table 5.1. Parse Times (in CPU seconds on a DEC 3100 workstation) and Storage Allocated (in megabytes) for the Test Sentences of Chapter 3.

relatively slow DEC 3100). The total numbers of unifications performed might be thought to be a useful statistic to aid the comparison, since unification is the basic operation in the parsers;however, a meaningful count cannot be given for the chart parser due to its rule retrieval optimisations, and the number of unifications is factored into the time and space figures, together with the overheads that are an inseparable part of the parsing strategy used in each case.

The statistics indicate that the non-deterministic LALR(1) parser is approximately 35% faster than the chart parser, and allocates about 30% less storage. Although (as argued in chapter 1) parse time is important factor in practical parsing of long and ambiguous texts, parsimonious use of storage is also a significant consideration; the reduced storage requirements of the LALR parser compared to the chart parser—in addition to the better parse times—thus make it more suitable for such tasks. However, when taken in the context of the enthusiastic advocation of non-deterministic CF LR parsing by some researchers (see the start of this section), the performance figures for the (unification-based) LALR parser appear somewhat disappointing. There is scope for experimenting with backbone grammars which expand out different unification grammar features, and manually tuning the resulting grammars to attempt to improve parsing performance. Whilst such efforts would probably have a limited amount of success, it seems unlikely that any dramatic gains would accrue. Moreover, subsequent modifications to the unification grammar would be likely to invalidate any tuning efforts of this sort.

The only modest performance advantage of the LALR parser over the chart parser helps confirm the negative result of the top-down processing experiments with the latter using the ANLT grammar, described in section 3.3. In those experiments, only a 20% decrease in the number of inactive edges was observed in a modified version of the chart parser when a left corner reachability table was used, and in fact no more than a 45% decrease would be possible if gaps could be predicted only where they actually occurred. The empirical results reported in this section demonstrate that performance gains of around 30–35% over the unmodified chart parser can be achieved with the ANLT grammar by precomputing a context free predictive (LR) component and using this to drive a parser. It therefore seems unlikely, with this grammar, that

yet more sophisticated precompilation techniques (perhaps applied directly to the unification grammar) would give a performance improvement of more than about 50% over the chart parser. Thus, when the techniques described in chapter 3 are used, bottom-up chart parsing is still a competitive technology for unification-based parsing.

All three of the parsers in this experiment exhibit exponential factors in their worst-case time complexities. However, the relationship between length of sentence and parse times with the ANLT grammar and the sentences tested appears merely to lie somewhere between linear and quadratic. There are probably many reasons why performance is much better than the complexity results might suggest, but the most important may be that more than 90% of the rules in the grammar have no more than two daughters, and that very few rules license both left and right recursion (for instance of the sort that is typically used to analyse noun compounding, i.e. `N --> N N`).

Looking at the performance of the Schabes-type parser, the figures show it to be twice as slow as the LALR parser, and also slower than the chart parser. However, it is intermediate between the two in terms of storage allocated. The space difference and some of the difference in speed between it and the LALR parser are due to the fact that it performs around 15% more unifications. This might be expected since the corresponding finite state automaton (FSA) is not determinised—to avoid theoretical exponential time complexity on grammar size—and thus pays a price at run time with a dynamic programming simulation of non-determinism with is essentially the same as the algorithm that determinises a FSA (Weir, personal communication). It appears that most of the remaining difference in speed is due to the space-time trade-off that is necessary in order to be able to represent the chart in a reasonable amount of memory. (This practical aspect is something that no analysis of complexity can identify).

Part of Schabes' motivation for developing his parsing algorithm was to eliminate the theoretical exponential time complexity of Tomita's algorithm on grammar size. Schabes does this by ensuring that the number of states in the parse tables used by his algorithm is merely a linear function of the size of the grammar, instead of an exponential one as with LR tables. However, in the case of the ANLT grammar, despite the fact that there are fewer actions in the Schabes-type parse table than in the LALR(1) table (a total of 1,360,000; see section 4.2.3 for statistics on the size of the latter), the Schabes table nonetheless contains more states[38]. This suggests that, when considering the complexity of parsers, the issue of parse table size is of minor importance for real-world NL grammars (as long as an implementation represents the table compactly), and that improvements to complexity results with respect to grammar size, although interesting from a theoretical standpoint, may have little practical relevance for the processing of Natural Language.

[38] The table contains one state for every kernel dotted item implicit in the grammar. Schabes (1991:111) suggests that selected portions of the table could be determinised to decrease the non-determinism in the finite state control; however, the resulting table would contain more states than the original table.

6. Probabilistic LR Parsing

Realistic, wide-coverage NL grammars (such as the ANLT grammar described in chapter 2) often produce large numbers of analyses for syntactically legitimate inputs. This chapter describes a practical parsing system capable of returning probabilistically highly-ranked analyses for material drawn from a specific corpus on the basis of minimal (supervised) training and manual modification. The system has been tested on definitions from a machine-readable dictionary, and the results of experiments with this corpus are reported[39].

6.1 Probabilistic Approaches to Parsing

6.1.1 Previous Approaches

In the field of speech recognition, statistical techniques based on hidden Markov modelling are well established: sufficiently so to be described in textbooks, for example Holmes (1988:129f). The two main algorithms utilised are the Viterbi (1967) algorithm and the Baum-Welch algorithm (Baum, 1972). The former provides a polynomial solution to the task of finding the most probable derivation for a given input and a stochastic regular grammar, and the latter, a polynomial method of performing iterative re-estimation of the parameters of a (hidden) stochastic regular grammar by considering all possible derivations over a corpus of inputs. Baker (1982) demonstrates that Baum-Welch re-estimation can be extended to context-free grammars (CFGs) in Chomsky Normal Form (CNF).

Typically, in applications of Markov modelling in speech recognition, the derivation used to analyse a given input is not of interest; rather what is sought is the best (most likely) model of the input. In any application of these or similar techniques to parsing though, the derivation selected is of prime interest. Baum (1972) proves that Baum-Welch re-estimation will converge to a local optimum in the sense that the initial probabilities will be modified to increase the likelihood of the corpus given the grammar and 'stabilise' within some threshold after a number of iterations over the training corpus. However, there is no guarantee that the global optimum will be found, and the *a priori* initial probabilities chosen are critical for convergence on useful probabilities (e.g. Lari & Young, 1990). The main application of these techniques to text has been in the robust, lexical tagging of corpora with part-of-speech labels (e.g. Garside, Leech & Sampson,

[39] Much of the research reported in this chapter stems from extensive discussions with Ted Briscoe. The disambiguation method of section 6.2.1, the technique of associating probabilities with LR tables (section 6.2.3), and the normalisation calculation described in section 6.3.2 represent joint work.

1987; de Rose, 1988; Meteer, Schwartz & Weischedel, 1991; Cutting *et al.*, 1992).

Fujisaki *et al.* (1989) describe a corpus analysis experiment using a probabilistic CFG in Greibach Normal Form[40] containing 7,550 rules on a corpus of 4,206 sentences. The (unsupervised) training process involved automatically assigning probabilities to each CF rule on the basis of their frequency of occurrence in all possible analyses of each sentence of the corpus. These probabilities were iteratively re-estimated using a variant of the Baum-Welch algorithm, and the Viterbi algorithm was used in conjunction with the CKY parsing algorithm (Kasami, 1965; Younger, 1967) to efficiently select the most probable analysis after training. The model was restricted in that many of the possible parameters (rules) defined over the category set were initially set to zero and training was used only to estimate new probabilities for a set of predefined rules. Fujisaki *et al.* suggest that the stable probabilities will model semantic and pragmatic constraints in the corpus, but this will only be so if these correlate with the frequency of rules in correct analyses, and also if the 'noise' in the training data created by the incorrect parses is effectively factored out. Whether this is so depends on the number of 'false positive' examples with only incorrect analyses, the degree of heterogeneity in the training corpus, and so forth. In experiments reported by Fujisaki *et al.*, when reparsing the corpus that was originally used for training, in 72 out of 84 sentences examined the most probable analysis was also the correct analysis. Six of the remainder were false positives and did not receive a correct parse, whilst the other six did, but it was not marked as the most probable. Thus the success rate (per sentence) is 85%.

Sharman, Jelinek & Mercer (1990) conducted a similar experiment with a grammar in ID/LP format (Gazdar *et al.*, 1985; Sharman, 1989). ID/LP grammars separate the two types of information encoded in CF rules—immediate dominance and immediate precedence—into two rule types which together define a CFG. This allows probabilities concerning dominance, associated with ID rules, to be factored out from those concerning precedence, associated with LP rules. In this experiment, a supervised training regime was employed. A grammar containing 100 terminals and 16 non-terminals and initial probabilities based on the frequency of ID and LP relations was extracted from a manually parsed corpus of about one million words of text. The resulting probabilistic ID/LP grammar was used to parse sentences drawn from the same corpus. In addition, lexical syntactic probabilities were integrated with the probability of the ID/LP relations to rank parses. In an experiment with 42 sentences of 30 words or less drawn from the training corpus, 18 of the parses were identical to the original manual analyses, whilst a further 19 were 'similar', yielding a success rate of 88%.

Pereira & Schabes (1992) report an experiment using Baum-Welch re-estimation to infer a grammar and associated rule probabilities from a category set containing 15 non-terminals and

[40] Greibach Normal Form is similar to CNF, except that each production contains a terminal category as its leftmost daughter.

48 terminals, corresponding to the Penn Treebank lexical tagset (Santorini, 1990). The training data was 770 sentences, represented as tag sequences, drawn from the treebank. The system was trained in an unsupervised mode and also in a 'semi-supervised' mode in which the manually parsed version of the corpus was used to constrain the set of analyses used during re-estimation. In supervised training, analyses were accepted if they produced bracketings consistent but not necessarily identical with those assigned manually. In supervised mode, training not only converged faster but also resulted in a grammar in which the most probable analysis was compatible with the manually assigned analysis of further test sentences drawn from the treebank in a much greater percentage of cases: 78% as opposed to 35%. This result indicates very clearly the importance of supervised training, particularly in a context where the grammar itself is being inferred in addition to the probability of individual rules.

Sampson, Haigh & Atwell (1989) describe a parser which, in order to assign an analysis, uses a statistically-defined measure of 'closest fit' to the set of hand-assigned analyses contained in training data taken from the LOB treebank (Sampson, 1987). The approach, based on simulated annealing, attempts to ensure that analyses of new data will conform as closely as possible to existing ones. Sampson *et al.* express the results of experiments with this technique in terms of a tree similarity measure with respect to correct analyses assigned by hand. For a 13-sentence sample the mean similarity measure was 80%, but only one input received a fully correct analysis. These results suggest that the technique in its current form is not reliable enough for practical NL analysis. Moreover, the analyses assigned, based on the LOB treebank scheme, are not syntactically determinate (for example syntactic relations in unbounded dependency constructions are not represented).

6.1.2 Problems with Probabilistic CFG

Although the experimental results reported for the probabilistic parsing systems mentioned above are quite impressive, there are nevertheless several inherent problems with probabilistic CFG (including ID/LP) based systems. Firstly, although CFG is an adequate model of the majority of constructions occurring in natural language (Gazdar & Mellish, 1989), it is clear that wide-coverage CFGs will need to be very large indeed, and this will lead to difficulties of (manual) development of consistent grammars and, possibly, to computational intractability at parse time (particularly during the already computationally expensive training phase). Secondly, without further apparatus, the analyses returned by grammars based on a linguistically relatively unsophisticated formalism such as CFG (even in an ID/LP version), or derived by analogy with manual analyses of the training corpus and therefore not well-formed with respect to any given generative grammar (e.g. Sampson *et al.*'s system), are not likely to be detailed enough to

support subsequent semantic interpretation. Thirdly, associating probabilities with CF rules means that information about the probability of a rule applying at a particular point in a parse derivation is lost. This leads to complications distinguishing the probability of different derivations when the same rule can be applied several times in more than one way. Figure 6.1 gives an example of a probabilistic CFG (PCFG)—based on the grammar G1 of section 4.1—in which each production is associated with a probability and the probabilities of all rules expanding a given non-terminal category sum to one.

In PCFG, the probability of a particular parse is the product of the probabilities of each rule used in the derivation. Thus the probability of parse a) in figure 6.2 is 0.09. Parse b) must have an identical probability, though, because the same rule is applied twice in each case. Similarly, the probabilities of c) and d) are also identical (0.09) for essentially the same reason. However, the rules concerned are natural treatments of noun compounding and prepositional phrase attachment in English, and the different derivations correlate with different interpretations. For example, a) would be an appropriate analysis for *toy coffee grinder*, whilst b) would be appropriate for *cat food tin*, and each of c) and d) yields one of the two possible interpretations of the noun phrase *the man in the park with the telescope*. Ideally, these structural configurations should be kept probabilistically distinct in case there are structurally conditioned differences in their frequency of occurrence, as would be predicted, for example, by the theory of parsing strategies (e.g. Frazier, 1988). Fujisaki *et al.* (1989) propose a rather inelegant solution for the noun compound case which involves creating 5,582 instances of 4 morphosyntactically identical rules for classes of word forms with distinct bracketing behaviour in noun-noun compounds. This technique enlarges the grammar excessively, merges probabilistic lexical information with probabilistic structural information, and also multiplies out the parameters of the probabilistic model.

Moreover, PCFGs cannot model the context dependence of rule use; for example, a noun phrase is more likely to be expanded as a pronoun in subject position than elsewhere (e.g. Magerman & Marcus, 1991) but only one global probability can be associated with the relevant CF production. Thus the PCFG model predicts (incorrectly) that e) and f) will have the same probability of occurrence. These considerations suggests the need for a technique which employs

```
1:  S' --> S          (1.0)        6:  NP --> Det N     (0.3)
2:  S --> NP VP       (1.0)        7:  NP --> NP PP     (0.3)
3:  VP --> Vt NP      (0.4)        8:  N --> N N        (0.3)
4:  VP --> Vi         (0.6)        9:  PP --> P NP      (1.0)
5:  NP --> ProNP      (0.4)        10: N --> N@         (0.7)
```

Figure 6.1. A Probabilistic Version of G1.

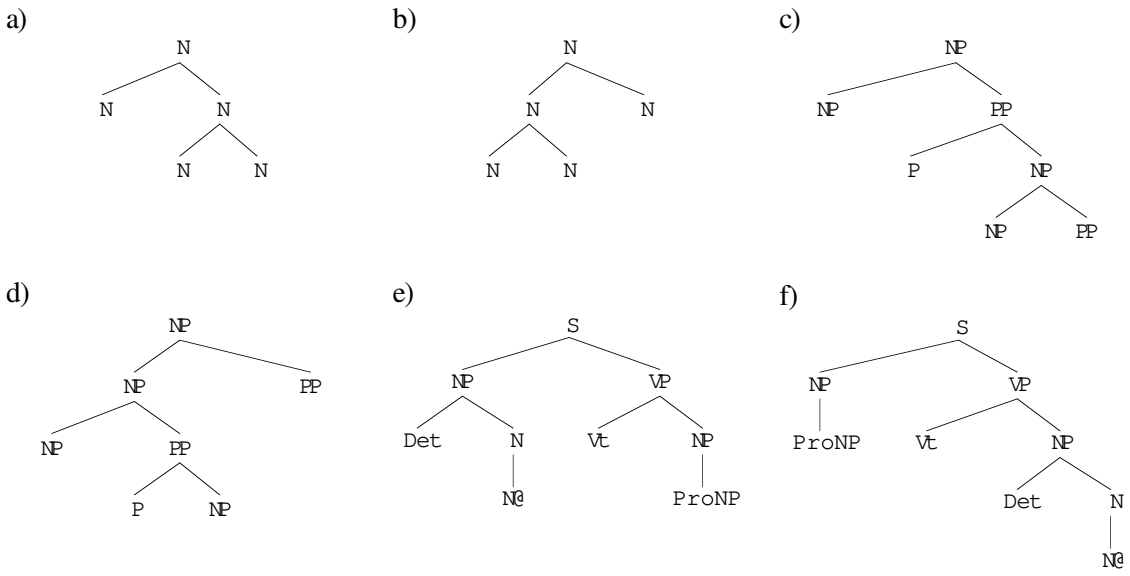

Figure 6.2. Context-free Derivations.

a more adequate grammatical formalism than CFG and a more context dependent probabilistic model.

Several researchers (Wright & Wrigley, 1989; Wright, 1990; Ng & Tomita, 1991; Wright, Wrigley & Sharman, 1991) have proposed using the LR technique as a practical method of parsing with a PCFG. In these approaches, an LR parse table is derived from the CF component of the grammar in the standard way, but in addition, the rule probabilities are distributed amongst sets of actions in the LR parse table. The distribution is performed in such a way as to ensure that the product of the probabilities attached to the LR actions which are taken in the derivation of any given analysis will be identical to the probability which would be assigned to the analysis by the original PCFG.

The following sections describe a novel approach to probabilistic parsing, using the LR parsing technique as a way to obtain a finite-state representation of a non finite-state grammar incorporating information about parse context, and demonstrate that LR parse tables do indeed provide an appropriate amount of contextual information to solve the problems described above. In this approach, a CF backbone grammar and LR parse table is computed for the ANLT grammar (using the techniques described in chapter 4), and probabilities are associated directly with the parse table, rather than being associated with rules of the grammar as in PCFG (or equivalently, being mapped from a PCFG into an LR parse table). An LR parse state encodes information about the left and right context of the current parse. In the system, probabilities are assigned to transitions in the LALR(1) table *via* a process of supervised training based on computing the frequency with which transitions are traversed in a disambiguated corpus of parse histories constructed using a user-driven, interactive LR parser. Deriving probabilities relative to

the parse context allows a probabilistic parser, unlike a PCFG, to distinguish derivations which differ only in terms of order of application of the same set of CF backbone rules (within a context defined by the LALR(1) table) or situations in which identical rules apply with differing probabilities in different contexts. The probabilistic LR parser remains a stochastic first-order finite-state model, though, because the LALR(1) table defines a non-deterministic finite-state machine and the total probability of an analysis is computed from the sequence of transitions taken to construct it.

6.1.3 Language Models and Decision Procedures

The approach outlined above does not result in a *language model* (as is PCFG), since the probability of a sentence is not the sum of the probabilities of all its possible derivations. The approach instead constitutes a *decision procedure*. Language models trained using Baum-Welch re-estimation maximise the sum of the probabilities of the sentences in a corpus given a grammar. However, it is debatable whether this property is what is required when the task is to select a single, plausible derivation as the analysis for each sentence, since there is no clear connection between the probability of a derivation and that of a sentence. For this task, it would be more appropriate to maximise the probability of the *correct derivation*, given a sentence in the training corpus. In particular, the implications of the re-estimation process and the interpretation of the probabilities contained in a language model can be counterintuitive. For example, consider the two PCFGs in figure 6.3[41]. With the first grammar, the probability of the string *a* is 0.1, and that of the string *a a* (whose derivation involves one instance of rule 1, and two instances of rule 2) is 0.009. With the second grammar, *a* is assigned probability 0.9, and *a a*, 0.081. Thus the string of length one receives the higher probability in both cases, even when the rule `S --> S S` has a much higher probability than `S --> a`. Also, the string of length two turns out to be much more likely than that of length one when the *unary* rule has the higher probability. Thus, it is not clear that language models are suitable for the task of parse selection.

Other researchers have noticed these problems with language models and PCFG, and have described approaches to parsing based on the same idea of using context-sensitive derivation

```
1:  S --> S S       (0.9)        1:  S --> S S       (0.1)
2:  S --> a         (0.1)        2:  S --> a         (0.9)
```

Figure 6.3. Two Probabilistic CF Grammars.

[41] Ted Briscoe pointed out this example. Note that the grammars do not generate *normalised* probabilistic languages, since in neither case does the sum of the probabilities of all the sentences generated sum to 1 (Wright, 1992).

probabilities. For example, *Pearl* (Magerman & Marcus, 1991) and *Picky* (Magerman & Weir, 1992) use decision procedures which estimate the probability of a derivation as the geometric mean of the product of the conditional probabilities of each CFG rule appearing in the derivation given the (non-terminal trigram) context of its dominating rule and the part-of-speech trigram at that point. Chitrao & Grishman (1990) also describe a context dependent probabilistic model in which the probability of each possible expanding production for each non-terminal appearing on the right hand side of a rule is recorded separately, and report a 23% decrease in incorrect analyses over a previous (standard) PCFG. However, neither of these approaches can distinguish the structures resulting from noun compound ambiguities for example, such as in figure 6.2 a) and b). Moreover, although achieving promising results in a limited domain, these systems are based on CFG and thus are unlikely to produce analyses that are detailed enough to support semantic interpretation.

The framework and implemented system described in the rest of this chapter is intended to constitute a principled and linguistically adequate approach. The next section describes how the disambiguated training corpus that the system uses is constructed and how probabilities are derived from it, section 6.3 describes the probabilistic parser, and 6.4 reports the results of a corpus parsing experiment using the probabilistic LR system.

6.2 Training using Interactive Incremental LR Parsing

6.2.1 Constructing a Disambiguated Training Corpus

The major problem with attempting to employ a disambiguated training corpus is to find a way of constructing this corpus in an error-free and resource-efficient fashion. Even manual assignment of lexical categories ('tagging') is slow, labour intensive and error prone. The greater complexity of constructing a complete parse makes the totally manual approach very unattractive, if not impractical. Sampson (1987:83) reports that it took 2 person-years to produce the LOB treebank of 50,000 words. Furthermore, in that project, no attempt was made to ensure the analyses were well-formed with respect to a generative grammar. Attempting to manually construct analyses consistent with a grammar of any size and sophistication would place an enormous additional load on the analyst. Leech & Garside (1991) discuss the problems which arise in manual parsing of corpora concerning accuracy and consistency of analyses across time and analyst, the labour intensive nature of producing detailed analyses, and so forth. They advocate an approach in which simple 'skeleton' parses are produced by hand from previously tagged material, with checking for consistency between analysts. These skeleton analyses can then be augmented automatically with further information implicit in the lexical tags. Whilst this

approach may well be the best that can be achieved with fully manual techniques, it is still unsatisfactory. Firstly, the analyses are crude, and it is not clear how to train an existing grammar with such unrelated analyses. Secondly, the quality of any grammar obtained automatically from the parsed corpus is likely to be poor because of the lack of any rigorous checks on the form of the skeleton parses. Such a grammar might, in principle, be trained from the parsed corpus, but there are still likely to be mismatches between the actual analysis assigned manually and any assigned automatically.

These training problems can be overcome by employing a method which produces a training corpus using the same grammar which ultimately is to be trained, thus guaranteeing that any analysis assigned will be a member of the set defined by the grammar. Using a similar type of technique to that employed in the construction of a substantial lexicon for the ANLT grammar from unreliable and indeterminate data (Carroll & Grover, 1989), the disambiguated training corpus is created semi-automatically, with manual interaction being restricted to the selection between alternatives defined by the ANLT grammar. Briscoe (1987) demonstrates that the structure of the search space in parse derivations allows a left-to-right, incremental mode of parse selection to be effective. For example, in noun compounds analysed using a recursive binary-branching rule (`N --> N N`) the number of analyses correlates with the Catalan series (Church & Patil, 1982; see section 3.2). However, Briscoe (1987:154f) shows that with a simple bounded context parser (with one word lookahead) set up to request help whenever a parse indeterminacy arises, it is possible to select any of the 14 analyses of a 5 word compound with a maximum of 5 interactions and any of the 1,430 analyses of a 9 word compound with around 13 interactions. In general, resolution of the first indeterminacy in the input will rule out approximately half the potential analyses, resolution of the next, half of the remaining ones, and so on. For "worst case" CF ambiguities, this approach to parse selection appears empirically to involve numbers of interactions which increase at little more than linear rate with respect to the length of the input.

It is possible to exploit this insight in two ways. One method is to compute all possible analyses represented as a (packed) parse forest and ask the user to select between competing sub-analyses which have been incorporated into a successful analysis of the input. In this way, only genuine global syntactic ambiguities would need to be considered by the user. However, the disadvantage of this approach is that it relies on a prior (and perhaps CPU-intensive) on-line computation of the full set of analyses. The second method involves a user interacting with an incremental parser to guide it through the search space of possibilities. This has the advantage of being guaranteed to be computationally tractable, but the potential disadvantage of requiring the user to resolve local syntactic ambiguities which will not be incorporated into a successful analysis. Nevertheless, using LR and augmented lookahead techniques this problem can be minimised, and, because time-consuming on-line computations must be avoided when interacting with a user, the system uses the latter incremental method.

6.2.2 The Interactive LR Parsing System

The interactive, incremental parsing system asks the user for a decision at each choice point during the parse. However, to be usable in practice, such a system must avoid, as far as possible, presenting the user with choices that could be ruled out either by using more of the left context, or by looking at words yet to be parsed. The approach that is implemented goes some way to addressing these points, since the parser is as predictive as the backbone grammar and LR technique allow, and the LALR(1) parse table allows one word lookahead to resolve some ambiguities (although, of course, the resolution of a local ambiguity may potentially involve an unlimited amount of lookahead).

An LR parser faces an indeterminacy when it enters a state in which there is more than one possible action, given the current lookahead. In a particular state there cannot be shift actions to more than one state, but there can be several reduce actions each specifying a reduction with a different rule (perhaps in addition to a shift action). When parsing, each shift or reduce alternative will necessarily lead to a distinct analysis, and so the indeterminacy represents a point of syntactic ambiguity (although it may not correspond to a genuinely global syntactic ambiguity in the input, on account of the limited amount of lookahead).

In the ANLT grammar and lexicon, lexical ambiguity is at least as pervasive as structural ambiguity. A naive implementation of an interactive LR parser would ask the user the correct category for each ambiguous word as it was shifted; many open-class words are assigned upwards of twenty lexical categories by the ANLT lexicon with relatively fine distinctions between them, so this strategy would be completely impracticable. To avoid asking the user about lexical ambiguity, the parser uses the technique of *preterminal delaying* (Shieber, 1983), in which the assignment of an atomic preterminal category to a lexical item is not made until the choice is forced by the use of a particular production in a later reduce action. After shifting an ambiguous lexical item, the parser enters a state corresponding to the union of states that would be entered on shifting the individual lexical categories. (Each union of states will in practice be small, since it being otherwise would imply that the current context was completely failing to constrain the following input). Since, in general, several unification grammar categories for a single word may be subsumed by a single atomic preterminal category, Shieber's technique is extended to deal with a grammar containing feature-based categories by associating not just one, but a set of alternative feature-based analyses with each state, and letting the choice between them be forced by later reduce actions, just as with atomic preterminal categories.

In order not to overload the user with unnecessary choices concerning local ambiguities, the parser does not request help immediately it reaches a parse action conflict. Instead the parser pursues each option in a limited breadth-first fashion and only requests help with analysis paths which remain active. In the system, this type of lookahead is limited to up to four

indeterminacies ahead. In practice, such checking is cheap in terms of machine resources and very effective in cutting down both the number of choice points the user is forced to consider, and also the average number of options in each one. Table 6.1 gives an example of the reduction in user interaction achieved by increasing the amount of lookahead in the system.

Computation of the backbone grammar generates extra rules (as described in section 4.2 to expand out the values of certain features, and for other reasons) which do not correspond directly to single unification grammar rules. At choice points, reductions involving these rules are not presented to the user; instead the system applies the reductions automatically, proceeding until the next shift action or choice point is reached, and including the options at this latter point in those presented to the user. The final set of measures taken to reduce the amount of interaction required with the user is to ask if the string being parsed contains one or more gaps or instances of coordination before presenting choices involving either of these phenomena, blocking consideration of rules on the basis of the presence of particular feature-value pairs. Figure 6.4 shows the system parsing a string with a four choice lookahead. (The resulting parse tree is displayed with category aliases substituted for the actual complex categories). The requests for manual selection of the analysis path are displayed to the analyst in as terse a manner as possible and require some knowledge of the grammar and lexicon to be resolved effectively.

Figure 6.5 summarises the amount of interaction required in the experiment reported in section 6.4 below, in which a set of 150 noun definitions from the *Longman Dictionary of Contemporary English* (Procter, 1978) are parsed with a version of the ANLT grammar[42]. To date, the largest number of interactions that has been observed for a single input string is 55 for the (30-word) definition for *youth hostel*:

> *a hostel for usu young people walking around country areas on holiday for which they pay small amounts of money to the youth hostels association or the international yha*.

Action conflict lookahead	Number of choices	Mean number of options in each choice
none	6	2.3
1 choice	5	2.2
2 choices	3	2.0
3 choices	2	2.0
4 choices	1	2.0

Table 6.1. Amount of User Interaction when Parsing *the act of putting an end to something* with the ANLT Grammar using Different Amounts of Action Conflict Lookahead.

[42] This version is the third release ANLT grammar with minor modifications to allow it to parse definitions (of nouns) in *The Longman Dictionary of Contemporary English* .

```
Parse>> the act of putting an end to something

Are there any gaps in this phrase? n

Ambiguity in state 27/193 with (of putting an end to something $)
remaining in buffer. Analysis so far is the, act.
1: Shift word 'of' onto stack.
2: Reduce end 1 analyses with rule N1/N (giving category N1-9).

Which choice (1 - 2 / abort / finish)? 2

2384 msec CPU
2700 unifications, 2025 failures, 1 parse

(N2 the
    (N2 (N1 act
            (P2 (P1 of
                    (VP (V putting) (N2 an (N2 (N1 end)))
                        (P2 (P1 to (N2 something)))))))))
```

Figure 6.4. An Interactive Parse (with user input in bold).

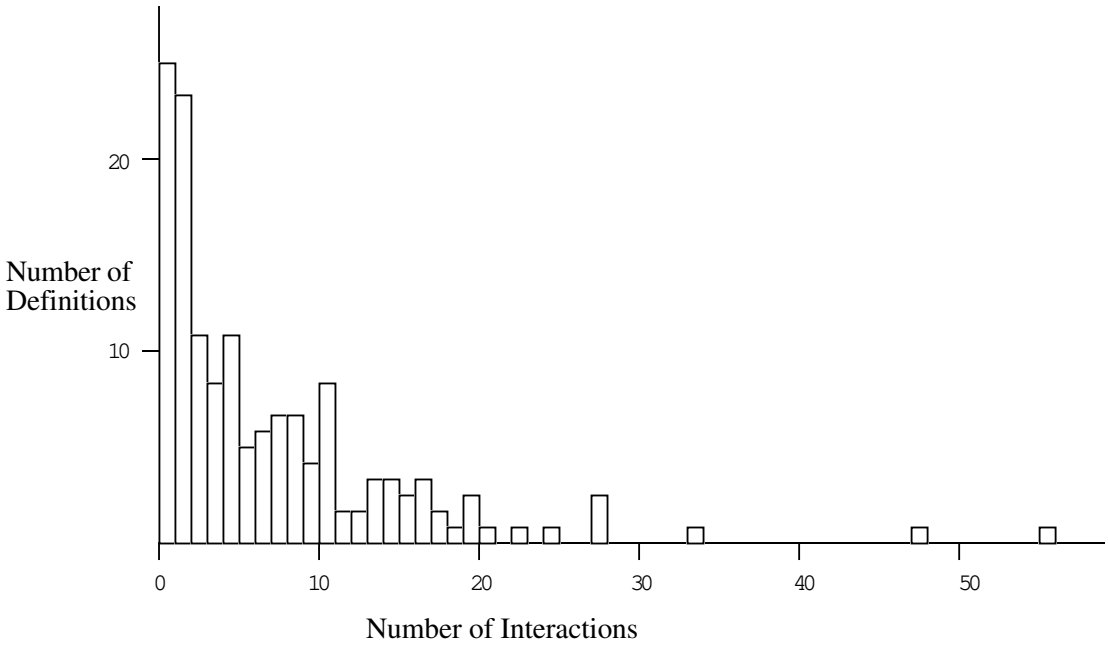

Figure 6.5. Frequencies of Particular Numbers of Interactions.

For this phrase, achieving the correct analysis interactively took the user about 40 minutes (including the addition of two lexical entries). Inputs of this length will often have many hundreds or even thousands of parses; computing the parse forest for this one takes of the order of two hours of CPU time (on a DEC 3100 workstation). Since in a more general corpus of written material the average sentence length is likely to be 30–40 words, this input illustrates clearly the problems with any approach based on *post hoc* on-line selection of the correct parse.

However, using the incremental approach to semi-automatic parsing, the correct analysis is demonstrably in the forest, and can be used as training material by means of an on-line computation.

6.2.3 Deriving Probabilities from Training Data

Associating Probabilities with an LR Parse Table

The sequences of shift, reduce and accept actions in the LALR(1) parse table that are performed by the interactive parsing system during the training phase are automatically stored, and the frequencies of parse table transitions that they represent are used to create a probabilistic version of the parse table for a non-deterministic probabilistic parser (section 6.3).

The most straightforward technique for associating probabilities with the parse table is to assign a probability to each action in the action part of the table (e.g. Wright, 1990; Ng & Tomita, 1991). In this approach, probabilities taken from a PCFG are distributed over sets of actions in such a way as to ensure that for every analysis licensed by the grammar, the product of the scores of the LR actions in its derivation is identical to the probability that would be assigned to the analysis by the PCFG. However, if this constraint is relaxed, and probabilities are associated directly with individual actions in the parse table (rather than being derived from a PCFG), then the resulting probabilistic model can be more sensitive to parse context. For example, in a derivation for the sentence *he loves her* using the grammar G1 of chapter 4 (figure 6.1), the steps of reducing the first pronoun and second pronoun in the sentence to `NP`—using the same rule `NP --> ProNP` in both cases—can be distinguished by the different lookahead items paired with the reduce actions relating to this rule (in state 5 of the parse table in figure 4.2); in the first case, the lookahead item will be `Vi`, and in the second, `$`. Although it is sensitive to parse context to a certain extent, this approach still does not make maximal use of the context encoded in the parse table, and it is possible to devise situations in which the reduction of a pronoun in subject position and elsewhere would be indistinguishable in terms of lookahead alone; for example, in this framework, if appropriate rules for adverbs are added to G1, then `Adv` would be the lookahead for reductions of pronouns in both subject and object positions in inputs

like *he probably loves her passionately*.

A slightly more complex approach is to further subdivide reduce actions according to the state reached after the reduce action has applied. This state is used together with the resultant non-terminal to define the state transition in the goto part of the parse table. Thus, this technique corresponds to associating probabilities with transitions in the automaton defined by the parse table, rather than with actions in the action part of the table. For example, a reduction of pronoun to NP in subject position in the parse table for G1 always results in the parser returning to state zero (from which the goto table deterministically prescribes a transition to state 7 with non-terminal NP). Reduction to NP of a pronoun in object position always results in the parser returning to state 11. Thus training on a corpus with more subject than non-subject pronominal NPs will now result in a probabilistic preference for reductions which return to 'pre-subject' states with 'post-subject' lookaheads. Of course, this does not mean that it will be impossible to devise grammars in which reductions cannot be kept distinct which might, in principle, have different frequencies of occurrence. However, this approach appears to be the natural stochastic, probabilistic model which emerges when using an LR table. Any further sensitivity to context would require sensitivity to patterns in larger sections of a parse derivation than can be defined in terms of such a table.

Deriving LR Parse Table Transition Probabilities

The first stage in the process of creating the probabilistic version of the parse table is to identify from each action stored in the training 'history', the transition in the parse table that was actually taken. This task is not trivial since, due to the use of the preterminal delaying techniques described in the previous section, most of the time the interactive parser is in a union of several states with a lookahead consisting of a set containing more than one atomic terminal category. Therefore, when an action is added to the parse history it is recorded as occurring in the context of a set of states and a set of lookaheads. To uniquely identify the transition corresponding to an action, the state and lookahead possibilities have to be determinised. To do this, the system first replays the actions backwards, for each reduce action pushing the daughters of the rule involved onto a stack; when a shift action is encountered, the action's lookahead (and the lookahead for all subsequent reduce actions up until the next shift) will be the terminal that is currently the top item of the stack, which is immediately popped. Once the lookahead for each action has been determined, the actions are replayed (forward) with reference to the parse table in order to determinise the state sets.

The second stage in the process is to compute the frequency with which each transition has been utilised over the whole of the training phase, and to normalise the frequencies obtained so

that for each transition from a given state the probabilities sum to one. The probabilities are actually stored separately from the LR parse table to ensure that otherwise-sharable transitions in the table (section 4.2.3) can still be represented compactly even if their probabilities differ. The probabilities are held as logarithms to allow them to be multiplied quickly during parsing.

Figure 6.6 shows a probabilistic LALR(1) parse table for G1 which was derived from a simple, partial (and artificial) training phase. Since probabilities are associated with transitions rather than with actions, there can be more than one transition corresponding to a single reduce action, with a separate probability for each depending on the state reached after the action; for example, in state 4 with lookahead `N@`, the probability of reducing with rule ten is 0.17 if the state reached is 3 and 0.22 if the state reached is 5. The actions which have no associated probabilities correspond to transitions which have not been utilised during the training phase; each is assigned a smoothed probability which is the reciprocal of the result of adding one to the total number of observations of transitions actually taken from that state. For this reason, the explicit probabilities in each row of the table sum to less than one. Unseen events are thus assigned differential probabilities in a manner analogous to the type of smoothing performed by the Good-Turing technique (Good, 1953). The goto part of the table is not shown because it is always deterministic and, therefore, probabilities are not associated with the entries in it.

Using LR Transition Probabilities

The difference between the approach described above and one based on PCFG can be brought out by considering parse derivations using the probabilistic parse table for G1. Assuming that probabilities are being used simply to rank parses, the total probability of an analysis can be computed by taking the product of the probabilities of each transition taken during its derivation. Figure 6.7 shows the two possible complete derivations for a sentence like *the winter holiday camp closed* consisting of a determiner, three nouns and an intransitive verb. The ambiguity concerns whether the noun compound is left- or right-branching, and, as discussed in section 6.1, a PCFG cannot distinguish these two derivations. The probability of each step can be read off the action table (figure 6.6), and is shown after the lookahead item in the derivations.

In step 8 a shift-reduce conflict occurs so the stack 'splits' while the left- and right-branching analyses of the noun compound are constructed. The a) branch corresponds to the right-branching derivation and the product of the probabilities is 4.6×10^{-8}, whilst the product for the left-branching b) derivation is 5.1×10^{-7}. Since the table was constructed from parse histories with a preponderance of left-branching structures this is the desired result. In practice, this technique is able to distinguish and train accurately on three of the five possible structures

State	$	Det	N@	P	ProNP	Vi	Vt
0		s3 (.50)			s2 (.50)		
1	r1 (0 .83)						
2	r5			r5 (8 .33)		r5	r5 (0 .50)
3			s4 (1.00)				
4	r10 (3 .11 6 .11)		r10 (3 .17 5 .22)	r10 (3 .11)		r10 (3 .11 5 .11)	r10
5	r6 (8 .13 11 .13)		s4 (.33)	r6 (11 .13)		r6 (0 .20)	r6
6	r8 (3 .17 5 .17)		r8 (3 .25) s4 (.17)	r8		r8 (3 .17)	r8
7				s8		s13 (.43)	s11 (.43)
8		s3 (.50)			s2 (.50)		
9	r9 (12 .40)			r9 (12 .40) s8		r9	r9
10	r7 (11 .40)			r7 (11 .40)		r7	r7
11		s3 (.75)			s2		
12	r3 (7 .43)			s8 (.43)			
13	r4 (7 .75)						
14	r2 (0 .84)						
15	acc (1.00)						

Figure 6.6. A Probabilistic Version of the Parse Table for G1.

```
                    0) 0                  (Det)
                    1) 0 Det 3            (N@1)  .50
                    2) 0 Det 3 N@ 4       (N@2) 1.00
                    3) 0 Det 3 N          (N@2)
                    4) 0 Det 3 N 5        (N@2)  .17
                    5) 0 Det 3 N 5 N@ 4   (N@3)  .33
                    6) 0 Det 3 N 5 N      (N@3)
                    7) 0 Det 3 N 5 N 6    (N@3)  .22

 8a) 0 Det 3 N 5 N 6 N@ 4  (Vi)  .17     8b) 0 Det 3 N          (N@3)
 9a) 0 Det 3 N 5 N 6 N     (Vi)          9b) 0 Det 3 N 5        (N@3)  .25
10a) 0 Det 3 N 5 N 6 N 6   (Vi)  .06    10b) 0 Det 3 N 5 N@ 4  (Vi)   .33
11a) 0 Det 3 N 5 N         (Vi)         11b) 0 Det 3 N 5 N     (Vi)
12a) 0 Det 3 N 5 N  6      (Vi)  .08    12b) 0 Det 3 N 5 N 6   (Vi)   .11
13a) 0 Det 3 N             (Vi)         13b) 0 Det 3 N         (Vi)

                   14) 0 Det 3 N 5        (Vi)   .17
                   15) 0 NP               (Vi)
                   16) 0 NP 7             (Vi)   .20
                   17) 0 NP 7 Vi 13       ($)    .43
                   18) 0 NP 7 VP          ($)
                   19) 0 NP 7 VP 14       ($)    .75
                   20) 0 S                ($)
                   21) 0 S 1              ($)    .83
                   22) 0 S'               ($)
                   23) 0 S' 15            ($)   1.00
```

Figure 6.7. Parse Derivations for *the winter holiday camp closed*.

for a four-word noun-noun compound; but it inaccurately prefers a completely left-branching analysis over structures of the form `((n n) (n n))` and `((n (n n)) n)`. Moving on to five-word noun-noun compounds, performance degrades further. However, this level of performance on such structural configurations is likely to be adequate, because correct resolution of most ambiguity in such constructions is likely to be dominated by the actual lexical items which occur in individual texts. Nevertheless, if there are systematic structural tendencies evident in corpora (for example, Frazier's (1988) parsing strategies predict a preference for left-branching analyses of such compounds), then the probabilistic model is sensitive enough to discriminate between them.

6.3 Probabilistic LR Parsing

The probabilistic parser uses the LR table transition probabilities derived in the training phase (as described in the previous section), to return the set of highest ranked analyses of an input. The parser consists of two processing stages: the first takes the input and computes a probabilistic packed parse forest, and the second extracts the n highest ranked analyses from the forest.

6.3.1 Computing a Probabilistic Parse Forest

The first stage of the probabilistic LR parser is based on the non-deterministic unification-based LR parser described in chapter 5 (briefly, a generalisation for unification grammar of Kipps' (1989) CF LR recogniser). This stage constructs a packed parse forest representation of the set of analyses licensed by the grammar for a given input. In this representation identical sub-analyses are shared between differing superordinate analyses, and sub-analyses covering the same portion of input are packed if the root categories are related by subsumption.

Ng & Tomita (1991) describe an LR parser for PCFG which associates probabilities with nodes in the graph-structured stack. When branches of the stack are merged (either due to a shift to the same state, or a goto action to an already existing vertex), then the vertex probability must be updated to be the sum of the products of the action and parent vertex probabilities. Because an addition operation is involved, this calculation is inconvenient to perform when probabilities are represented as logarithms (which make multiplication fast), so Ng & Tomita propose an approximate 'maximum likelihood approach' in which addition is replaced by maximisation. When splitting the stack on the reduction (popping) of a merged vertex, the probabilities of the parent vertices have to be recovered, requiring quite a complicated arithmetical manipulation. In general there can be more than one splitting in a single reduce action, and the arithmetic involved "can be tricky" (Ng & Tomita, 1991:157), leading them to propose reverting to a simpler type of (tree-structured) stack.

The probabilistic LR parser overcomes these problems by associating probabilities with partial analyses of the input, rather than with nodes in the graph-structured stack. Each node in the forest holds the probability of the sub-analysis rooted at that node; a distinct probability is also maintained for each packed sub-analysis at a node. A shift action creates a new parse forest node containing just the transition probability; the node created by a reduce action contains the product of the transition probability and the probabilities of each of the nodes reduced. The parser is thus able to merge and split stack configurations and in all cases still maintain a full record of all the probabilities computed up to that point.

Ideally, for any given input, the computation of the most probable analysis (or n-best analyses) defined by the probabilistic LR parser should not involve the construction of the complete parse forest, and thus an exhaustive search of the space of syntactically legitimate analyses defined by the grammar. However, it is not possible to introduce any Viterbi-style optimisation into the computation of local maximal paths through the probabilistic non-deterministic FSM defined by the parse table, because at any point in a derivation a maximal path may receive a probability of zero through unification failure, rendering a hitherto non-maximal local path maximal again. Unfortunately, the effects of feature propagation cannot be localised with respect to the computation of most probable sub-analyses, whilst any attempt to incorporate

featural information into the probabilistic component of the grammar would result either in an intractably large grammar, or a model with too many free parameters, or both. The parser therefore computes the complete parse forest.

6.3.2 Probabilistic Unpacking and Normalisation

Wright, Wrigley & Sharman (1991) describe a Viterbi-like algorithm to find the *n*-best analyses from a probabilistic parse forest, in which sub-analysis probabilities are the product of the probabilities of the lexical items spanned and of the PCFG rules involved in the sub-analysis derivation. The algorithm involves evaluating all the competing analyses for the same portion of input and pruning the lowest scoring ones. However, the competing (sub-)analyses at any stage might contain differing numbers of rule applications (since Wright *et al.* do not restrict the grammar to CNF, or some other notation which would guarantee that all analyses covering the same input would contain the same number of rules), and so the system would be likely to develop a bias in favour of analyses involving fewer rules, or equivalently 'smaller' trees, almost regardless of the training material[43]. Magerman & Marcus (1991) observe this problem in the experiments they perform, and propose that analysis probabilities be normalised by computing the geometric mean of the rule probabilities before they are compared. Using geometric means, as opposed to products, achieves a crude form of normalisation with respect to the number of rules in the analysis. This form of normalisation, though, cannot be integrated into Wright *et al.*'s unpacking algorithm without compromising the algorithm's soundness, since if all sub-analyses are normalised before pruning, then it is possible that a sub-analysis that is lower scoring after normalisation but which contains more rules than a competing one might be pruned, whereas the position could be reversed after the sub-analyses were integrated into a higher-level analysis. On the other hand, if sub-analyses were not normalised, then analyses involving smaller numbers of rules might be pruned even if ultimately they would form high-scoring complete analyses after the final normalisation step.

In addition, Wright *et al.*'s approach cannot be applied as it stands to unification-based systems because, in unpacking the forest, the remaining unifications involving the different featural extensions of packed nodes according to the superordinate and subordinate nodes to which they can be linked can lead to failure of a derivation encoded in the parse forest, unlike in the case of unpacking CFG parse forests.

The most straightforward way of extracting the *n*-best (normalised) analyses from a

[43] An approach to training based on a version of Baum-Welch re-estimation applied to non-CNF grammars (e.g. the trellis algorithm of Kupiec (1991)), although maximising the likelihood of the corpus given the grammar, would also be likely to incorrectly rank the analyses for individual sentences due to an inapproriate bias towards smaller trees.

unification-based probabilistic parse forest would be to exhaustively unpack the forest to retrieve each analysis, normalise its probability and return the highest scoring analyses. This method is impractical for large forests, and indeed, with the ANLT grammar, the longest string for which the parse forest has successfully been fully unpacked is only 22 words long. However, a heuristic (but nevertheless correct) technique for a best-first non-exhaustive search of the parse forest has been developed, which, in practice, makes the recovery of the most probable analyses much more efficient (allowing analysis of input strings containing over 30 words).

The technique is to identify the m-best analyses in the forest (where $m \geq n$) with a probabilistically-guided search, simultaneously normalising the scores for partial analyses of identical portions of the input. The best m complete analyses that have been identified are then unpacked from the forest, this process involving a small number of further unifications. The analyses for which these unifications all succeed are returned. If the number of analyses returned is less than the n wanted, the process could be repeated with a larger value for m. In practice, this approach leads to a considerable practical improvement in the time taken to produce the n-best analyses. Although the algorithm remains in the worst case exponential in sentence length, it will never need to search more than a fraction of the packed parse forest to recover the best analyses except in the case of pathological grammars. In addition, by separating the probabilistic computation from the creation of the parse forest, some of the problems induced by the non-local nature of feature propagation are avoided.

Identifying the Best Partial Analyses

The probabilistically-guided search to find the best analyses in a probabilistic packed parse forest involves maximising the (normalised) score at nodes in the parse forest at successive positions in the input, in a similar manner to the Viterbi algorithm. The method starts at the left edge of the forest (at string position zero, just before the first word), at each successive step normalising (as described in the next section) the probabilities assigned to all partial analyses spanning string positions zero to p which end at the same node in the forest, and then extending just the m highest scoring partial analyses at each node to reach position p+1. The process halts after pruning the lowest scoring analyses when p is equal to the length of the input, at which point all the analyses will be complete and will end at the root node of the forest. These analyses are returned as the m-best set.

A partial analysis is taken to end at the highest node which dominates the lexical node at the right edge of the analysis but which does not dominate the next lexical node to the right; or alternatively at the root node if there are no further words in the input. For example, the partial analyses from string position zero to two in the packed forest shown in figure 6.8a end at the

boxed nodes labelled N/COMP and N. Figure 6.8b shows the two analyses which are represented in the forest. Since the parser processes input from left to right and uses a shift-reduce strategy, it constructs right-most-first derivations. A partial analysis thus consists of one or more subtrees whose heights decrease or remain the same (but never increase) from left to right.

Each node in the forest contains the number and the product of the LR parse table transitions[44] taken up to that point in the analysis of the subtree dominated by the node. When extending a partial analysis with a new subtree, the transition count and product figures for the new extended analysis depend on the relationship between the heights of nodes at the roots of the subtrees currently making up the partial analysis and the height of the new (extending) subtree. If the root node of the new subtree is closer to the root of the forest than some of the current ones, then it dominates them, and so the figures for the extended analysis are those of the new node combined as appropriate with those of nodes in the partial analysis which preceded the dominated nodes. Otherwise a disjoint subtree is being added to the right edge of the partial analysis, and the figures at the new node are combined with those for all the current nodes. Packed nodes must be dealt with specially since the figures at a node take no account of nodes which are packed below it. In fact, it suffices to keep track of the differences between the figures at a node and those at the root of a subtree packed at that point whenever the subtree is incorporated into a partial analysis, and to factor in these differences whenever the figures at a node are subsequently required.

In the implementation, the parse tree corresponding to a partial analysis is represented

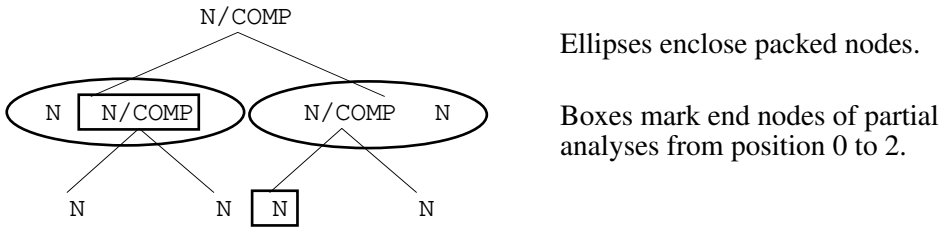

Figure 6.8a. A Packed Parse Forest.

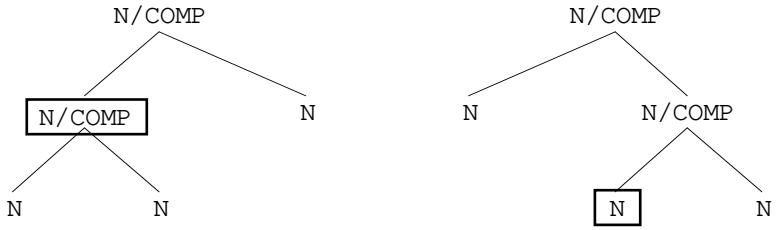

Figure 6.8b. Analyses that are Represented in the Forest.

[44] In fact what is stored is the sum of the logarithms of the transitions.

implicitly by the set of packed nodes that it contains; when the probabilistic search of the parse forest is finished, the actual tree is unpacked from the forest in an efficient depth-first traversal. Nodes at which there is no packing, or at which none of the packed nodes are in the given set, are incorporated into the tree. Packed nodes which are members of the set are incorporated, after performing the unification required to check that they are consistent with the rest of the analysis built so far, and the traversal continues inside the packed subtree.

Despite the fact that the current implementation of probabilistic unpacking is quite crude and could be speeded up significantly, for a sample of ten-word input strings with up to 150 analyses (from the experiment reported in section 6.4 below), the time the implementation takes to return the three highest ranked analyses from the parse forests is of the order of a factor of three less than in an exhaustive unpacking of the forest. In the course of searching for the best analyses, on average only about 20% of the nodes in the forests were visited. For longer inputs the difference is even more marked. The current implementation is able to return n-best analyses for every string for which the parser manages to create a parse forest; the longest such string is 31 words in length. Due to space and time constraints, the parser is unable to compute the full set of analyses for this string, but calculations indicate that it has at least two thousand analyses.

In the experiments carried out so far, once the forest has been searched for the highest scoring analyses, it appears that the great majority of unifications that take place when unpacking them are successful. In the experiment reported in section 6.4, setting up pruning of partial analyses to discard all but the best three at each stage (m), the number of final analyses returned for each input was only less than the three required (n) in the few cases where there were actually less than three analyses in total.

Normalising Partial Analyses

Since partial analyses spanning the same vertices may well have been derived via differing numbers of LR state transitions, scores for competing analyses must be normalised with respect to derivation length before they can be compared and the best ones identified. The scores of all the competing partial analyses which end at a given node in the forest are normalised to the length of the longest derivation in any of those analyses; for each derivation that is shorter by say d transitions, its score is multiplied by the geometric mean of the score raised to the power of d. So, for example, if at one stage the longest derivation was of length 5, two other derivations of length 3 and 4, with transition scores

a) 0.9 0.6 0.05
b) 0.5 0.2 0.2 0.08

would be multiplied by factors of, for a) the geometric mean of the three figures, squared (0.3^2 = 0.09), and for b), the geometric mean of the three figures (0.2), to give normalised scores. This computation can be performed quickly since transition scores are held as logarithms.

The need for the normalisation step reflects the fact that, although the model is more sensitive to context than PCFG, it is by no means a perfect probabilistic model of NL[45]. For example, the stochastic nature of the model and the fact that the entire left context of a parse derivation is not encoded in LR state information means that the probabilistic model cannot take account of, say, the pattern of resolution of earlier conflicts in the current derivation. Another respect in which the model is approximate is that probabilities are associated with the context-free backbone rather than with the unification grammar itself. Successful unification of features at parse time does not affect the probability of a (partial) analysis, whilst unification failure, in effect, sets the probability of any such analysis to zero. As long as the probabilistic model is only used to rank successful analyses, this is not particularly problematic. However, as mentioned above, parser control regimes attempting some form of best-first search using probabilistic information associated with transitions might return incorrect results, given this property.

6.4 Experiments Parsing LDOCE Definitions

6.4.1 The Experiments

In order to test the techniques and ideas described in the sections above, an experiment was carried out using as the test corpus a subset of noun definitions taken from the *Longman Dictionary of Contemporary English* (LDOCE; Procter, 1978)[46]. This corpus was chosen because the vocabulary employed is restricted (to approximately 2,000 morphemes), the average definition length is about 10 words (with a maximum of around 30), and each definition is independent, allowing phenomena such as ellipsis to be ignored. In addition, the language of definitions represents a recognisable sublanguage, allowing the task of tuning a general purpose grammar to be investigated. For the purposes of the experiment, a corpus of approximately 32,000 noun definitions was created from LDOCE by extracting the definition fields and normalising the definitions to remove punctuation, font control information, and so forth[47]. A lexicon was created for this corpus by extracting the appropriate lemmas and matching these against entries in the ANLT lexicon. The 10,600 resultant entries were loaded into the ANLT

[45] Marcus & Magerman (1991) argue that it is reasonable to use the geometric mean when computing the probability of sub-analyses because the independence assumptions which motivate using products do not hold for such an approximate model.

[46] Use of the machine readable version of LDOCE is by kind permission of Longman Group Ltd.

[47] The corpus contains about 17,000 unique headwords and 13,500 distinct word forms in the definitions. Its perplexity (PP) measures based on bigram and trigram word models and an estimate of an infinite model are PP(2) = 104, PP(3) = 41 and PP(inf) = 8 (Sharman, 1991).

morphological system (Ritchie *et al.*, 1987) and this sublexicon and the full ANLT grammar formed the starting point for the training process.

A total of 246 definitions, selected without regard for their syntactic form, were parsed semi-automatically using the interactive LR parsing system described in section 6.2[48]. During this process, further rules and lexical entries were created for some definitions which failed to parse. 150 were successfully parsed and 63 lexical entries and 14 rules were added. Some of the rules required reflected general inadequacies in the ANLT grammar; for example, rules were added to deal with new partitives and prepositional phrase and verb complementation. However, seven of these rules cover relatively idiosyncratic properties of the definition sublanguage; for example, the postmodification of pronouns by relative clause and prepositional phrase in definitions beginning *something that...*, *that of...*, parenthetical phrases headed by adverbs, such as *the period... esp the period...*, and coordinations without explicit conjunctions ending with *etc*, and so forth. Further special rules would be required to deal with parentheses to cover lexicographic conventions such as *a man (monk) or woman (nun) who lives in a monastery* which were ignored for this test. Nevertheless, the number of new rules required is not great and the need for most was identified very early in the training process. Lexical entries are more problematic, since there is little evidence that the number of new entries required would tail off as more training material was encountered. However, many of the entries required reflect systematic inadequacies in the ANLT lexicon rather than idiosyncrasies of the corpus. It took approximately one person-month to produce this training corpus. As a rough guide, a single interaction with the parser takes an average of 15 seconds. However, the time a parse takes can often be lengthened by incorrect choices (and the consequent need to back up manually) and by the process of adding lexical entries and occasional rules.

The resultant parse histories were used to construct the probabilistic parser (as described in the previous section). This parser was then used to reparse the training corpus and the most highly ranked analyses were automatically compared to the original parse histories[49]. All but 3 of the 150 definitions that were parsed manually were able to be reparsed in a breadth-first fashion. (These three are each over 25 words long). There are 22 definitions one word in length: all of these trivially receive correct analyses. There are 89 definitions between two and ten words in length inclusive (mean length 6.2). Of these, in 68 cases the correct analysis (as defined by the training corpus) is also the most highly ranked. In 13 of the 21 remaining cases the correct analysis is the second or third most-highly ranked analysis. Looking at these 21 cases in more detail, in 8 there is an inappropriate structural preference for ‘low’ or ‘local’ attachment (see Kimball, 1973), in 4, an inappropriate preference for compounds, and in 6 of the remaining 9

[48] Ann Copestake created the corpus and extracted the relevant lexical entries from the full ANLT lexicon. Antonio Sanfilippo and Ted Briscoe jointly performed the training.

[49] The results reported here are from using the latest versions of the ANLT grammar and LR parsing system. Briscoe & Carroll (1991; 1993) report earlier versions of this experiment using different versions of the grammar and parser in which the results differed in minor ways.

cases the highest ranked result contains a misanalysis of a single constituent two or three words in length. If these results are interpreted in terms of a goodness of fit measure such as that of Sampson, Haigh & Atwell (1989), the measure would be better than 96%. Taking correct parse per sentence as the measure then the result is 76%. For definitions longer than 10 words this latter figure tails off, mainly due to misapplication of such statistically-induced, but nevertheless structural, attachment preferences. The correct analysis was not returned for any definition of more than 15 words in length. Figure 6.9 summarises the results.

A further 55 LDOCE noun definitions not drawn from the training corpus were parsed, each containing up to 10 words (mean length 5.7). Of these, in 41 cases the correct parse is the most highly ranked, in 6 cases it is the second or third most highly ranked, and in the remaining 8 cases it is not in the first three analyses. This yields a correct parse per sentence measure of 75%. Examination of the failures again reveals that a preference for local attachment of postmodifiers accounts for 5 cases, a preference for compounds for 1, and the misanalysis of a single constituent for 2. The others are mostly caused by the lack of lexical entries with appropriate SUBCAT features. Figure 6.10 shows the analysis for the unseen definition of *affectation* which has 20 parses of which the most highly ranked is correct.

Figure 6.11 shows the highest ranked analysis assigned to one definition of *aid*. This is an example of a false positive which, in this case, is caused by the lack of a lexical entry for *support* as an intransitive verb. Consequently, the parser finds, and ranks highest, an analysis in which *supports* and *helps* are treated as transitive verbs forming verb phrases with object noun

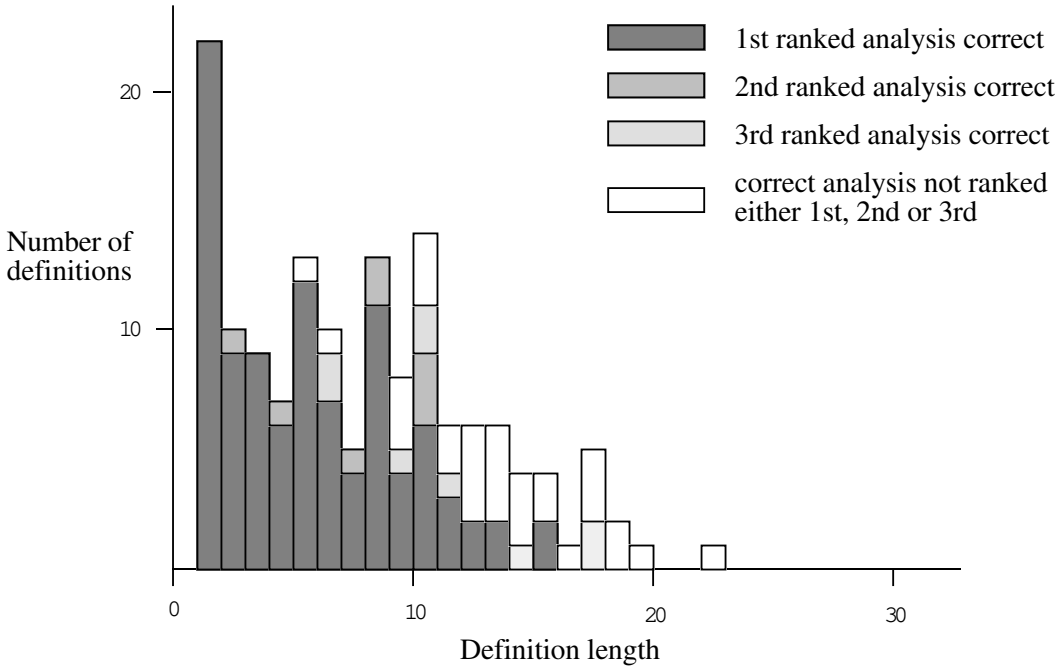

Figure 6.9. Correctness of Results for Reparsed Definitions with Respect to Length.

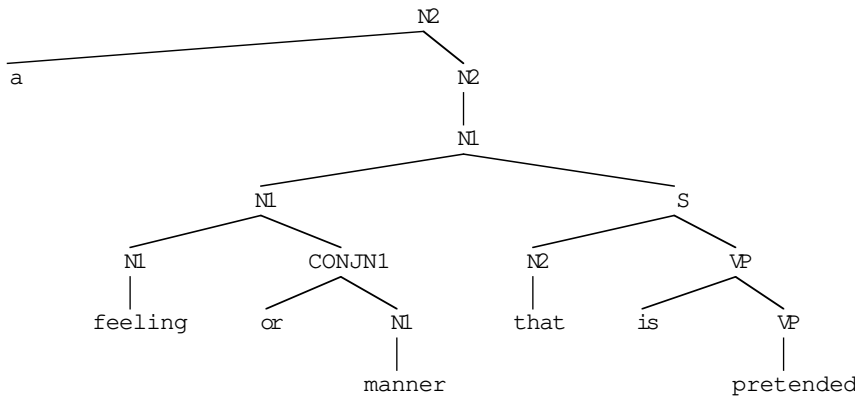

Figure 6.10. Parse Tree for *a feeling or manner that is pretended*.

phrase gaps, and *that supports or helps* as a zero relative clause with *that* analysed as a prenominal subject (compare *a person or thing that* ***that*** *supports or helps*). It is difficult to fault this analysis and the same is true for the other false positives observed to date. Such false positives present the biggest challenge to this type of system. One hopeful sign is that the analyses assigned such examples appear to have low probabilities relative to most probable correct analyses of other examples. However, considerably more data is required before it is possible to decide whether this trend is robust enough to provide the basis for automatic identification of false positives.

Appendix B contains the definitions in the training corpus between two and ten words long, and all of the test corpus. The number of parses for each definition are given, with an indication of whether the correct analysis was one of the three most probable.

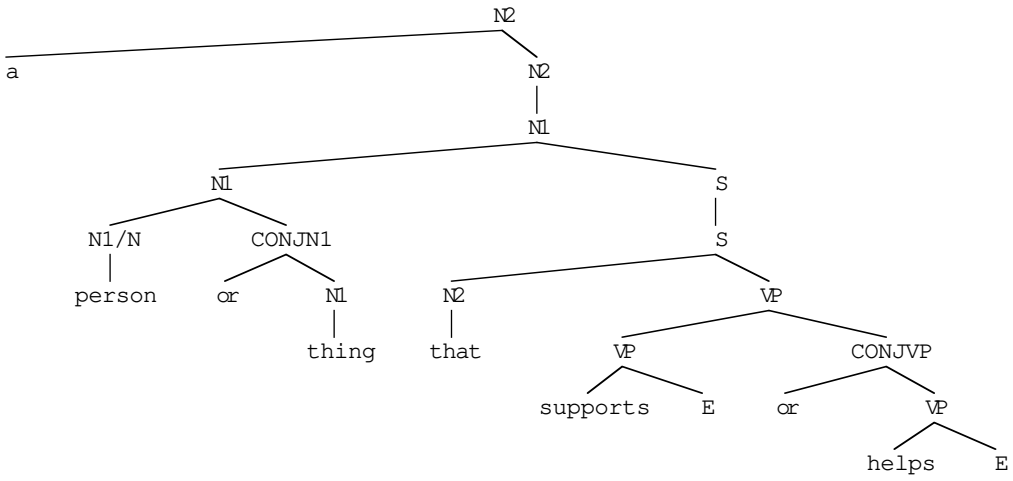

Figure 6.11. Parse Tree for *a person or thing that supports or helps*.

6.4.2 Discussion of Experimental Results

Fujisaki *et al.* (1989) report a success rate per sentence (with an average length of 11 words) of 85%, when reparsing the training corpus; this figure is impressive, but it is difficult to evaluate properly in the absence of full details concerning the nature of the corpus. For example, if the corpus contains many simple and similar constructions, the unsupervised training that they perform is more likely to converge quickly on a useful set of probabilities. Also, Fujisaki *et al.* appear to treat lexemes as terminal categories, so since their grammar is in Greibach Normal Form, it represents a substantial amount of lexical information. For example, two possible productions might be `VP --> believe Sfin` and `VP --> believe NP`; associating differential probabilities with these rules would allow the probabilistic model to represent the relative probability of *believe* subcategorising for a sentential complement or object noun phrase.

Sharman, Jelinek & Mercer's (1990) experiments yield a success rate (based on a tree similarity metric) of 88% for sentences drawn from the training corpus. In Pereira & Schabes' (1992) experiments, on a test corpus, 78% of the most probable analyses are consistent (with respect to bracketing) with manually assigned analyses. What is noticeable about these experiments is that the results appear to be no better than those of Fujisaki *et al.*, despite the use of supervised training and more sophisticated grammatical models. It is likely that these differences derive from the corpus material used for training and testing and the lack of such detailed lexical information, and that the results reported by Fujisaki *et al.* will not be achieved with all corpora.

In the preliminary experiments with the ANLT grammar and the LDOCE definitions sublanguage reported above, results comparable to these systems were achieved on the basis of a quite modest amount of manual effort and a very much smaller training corpus. This is because the manually disambiguated parse histories contain little 'noise' and usefully reflect the semantically and pragmatically appropriate analysis in the training corpus, and because the number of failures of coverage were reduced to some extent by the ability to add rules specifically motivated by the corpus during supervised training. Unlike Fujisaki *et al.* or Sharman *et al.*, the system does not integrate any information about lexemes into the rule probabilities, or make any use of information concerning the probability of a lexeme occurring as a specific lexical syntactic category. The results were achieved solely on the basis of statistics concerning the conditional probability of syntactic rules in a syntactically-defined (LR) parse context, and therefore a significant number of errors involve incorrect attachment of prepositional phrases, analyses of compounds, coordinations, and so forth, where lexical (semantic) information plays a major role. In many of these cases, the correct analysis is in the three highest ranked analyses, and moreover the structural preference for local attachment exhibited by the system might be overruled in appropriate contexts if lexeme (or better, word sense) information were taken into

account. The slightly worse results (relative to mean definition length) obtained for the unseen data appear to be caused more by the non-existence of a correct analysis in a number of cases, rather than by any significant decline in the usefulness of the rule probabilities.

7. Conclusions

This thesis is about the practical parsing of realistic NL texts with a wide-coverage unification-based grammar. The thesis proposes novel solutions to the problems of ensuring good parser throughput in the face of ambiguity and long input strings, selecting the correct analysis for a particular input from the set licensed by the grammar, and tuning a grammar to a particular corpus or sublanguage. The techniques described have been tested in an implemented computer system, and experiments have been carried out to assess their effectiveness in practice. The thesis does not tackle the related problem of providing reliable analyses of input outside the coverage of the grammar, but possible approaches which build on the framework presented are outlined in section 7.3 below.

This chapter summarises the novel techniques described in the thesis, comments on the results of experiments reported, presents general conclusions, and suggests directions for further work.

7.1 Maximising Parser Throughput

7.1.1 Parsing Techniques and Algorithms

The parsing systems described in the thesis are designed around the framework of the Alvey NL Tools (ANLT) grammar and associated grammatical formalism. The thesis presents a novel unification algorithm which is argued to be more suitable for practical NL parsing with this formalism (and related ones) than existing alternatives. The algorithm is employed in a highly-optimised bottom-up active chart parser. The optimisations include statically indexing strategic parts of the grammar, dynamically indexing certain run-time data-structures, and methods for minimising the use of dynamic storage; these techniques are straightforwardly implementable and fairly generally applicable to other parsers. In this parser they result in a speed gain of a factor of more than three. In experiments with a corpus, the parser with the ANLT grammar is shown to have significantly better practical performance than the CLARE system (Alshawi, 1992; Alshawi *et al.*, 1992), a comparable, state-of-the-art unification-based analysis system.

One surprising outcome of the experiments is that when the ANLT grammar is used with the CLARE system parser, throughput is very poor. Although it is likely that no explicit or formal process of 'tuning' parsers and grammars to perform well with each other has been carried out in

either of the two systems, the parallel development of the software and grammars that has occurred nevertheless appears to have caused this to happen automatically. It seems plausible that such a marked difference would not occur with systems using formalisms that were merely context-free (rather than unification-based), and that the more formally powerful or expressive the type of formalism, the more likely these types of difference would be to occur. For instance, in the context of ALE (Carpenter, 1992), research to integrate constraint-based grammar with chart parsing is likely to produce a parser with a control strategy that is specific to HPSG (Penn, personal communication); thus grammars written in the same formalism, but diverging from this particular theory, would be unlikely to perform as well.

In further experiments with the chart parser, the integration of top-down prediction by means of a left-corner reachability table results in no improvement in practical performance. This conclusion is also somewhat unexpected, and runs counter to (the limited number of) results reported in the literature for other systems.

Although the parsing systems described in the thesis produce parse forests, rather than exhaustively enumerated sets of parse trees (so that portions of sub-analyses and the work required to extend them are shared wherever possible), the systems are nevertheless still unable to process inputs more than about thirty words long, given reasonable time and space bounds. Realistic NL texts often contain sentences significantly longer than this. Text is also expected to have been tokenised ready for lexical access, and all punctuation to have been removed. (Since many aspects of punctuation do not reflect syntactic distinctions (e.g. Nunberg, 1990), this is appropriate in general). The text preprocessor that is thus required could 'package up' smaller units than whole sentences for parsing individually, when it encounters parentheticals and other sub-sentential constituents, for example, which are related by discourse relations (such as elaboration) rather than by syntactic relations. The practical restriction on length of input string would then be likely to have much less of an impact.

The thesis presents a novel methodology for constructing LR (and LR-like) parsers for unification grammars, *via* the automatic construction of a CF *backbone* grammar, from which an appropriate table is constructed that is used to drive a parser. Unifications are performed during each reduce action when parsing. The suitability of various types of LR parse table are compared for realistically-sized, ambiguous NL grammars, and a tractable approach to computing LALR(1) tables for such grammars is outlined, together with a scheme for representing tables compactly. Although the backbone grammar computation is automatic, its constraining power can be improved by manual intervention. To investigate these issues systematically, software tools would need to be built; a backbone grammar could then be reliably 'tuned' with respect to the particular phenomena that occur in the unification grammar. This process would be analogous to Ingria's (personal communication) efficiency modifications to the Delphi system (Boisen *et al.*, 1989), involving the (manual) 'compilation' of some combinations of categories and features

into new categories that are able to be more efficiently processed. Maxwell & Kaplan (in press) also describe a manual process of this type (but driven by statistics collected automatically), in which functional constraints in an LFG grammar are converted into equivalent phrasal constraints—thus splitting rules into more specialised versions—based on information about which constraints frequently caused constituents to be locally unsatisfiable when parsing.

A novel, efficient non-deterministic LR parsing algorithm for CF grammar is presented, which has better worst-case time complexity than any algorithm previously reported in the literature based on shift-reduce parsing with a graph-structured stack. The space complexity is also better, if grammar dependencies are ignored. A unification-based implementation of a version of this parser is described, and its practical throughput is empirically compared to the chart parser, and also to the implementation of another efficient LR-like algorithm that has recently been proposed in the literature. The results show that the LR parser is approximately 35% faster than the chart parser, and twice as fast as the LR-like parser. The latter difference seems to be due to smaller overheads and a more deterministic parse table.

In the adaption of the CF algorithm for unification grammar, the algorithm is augmented to cache the results of unifications (otherwise its performance is significantly degraded); æsthetically, this solution is not pleasing, and in this respect an algorithm specifically designed for unification grammar might be more satisfactory.

7.1.2 Parsing and Complexity

The thesis presents worst-case time complexity results for the parsing algorithms described. For a CF version of the unification-based chart parser, in the case where the grammar does not contain any rules whose daughters can iterate, the complexity is $O(|C|^2 |R|^3 n^{\rho+1})$, where n is the length of the input string, $|C|$ is the number of categories in the grammar, $|R|$, the number of rules, and ρ the maximum number of daughters in a rule. For grammars with at most binary-branching rules, the factor dependent on the length of the input is no worse than that of the CKY algorithm; moreover the chart parser accepts an arbitrary CF grammar and is not restricted to grammars expressed in Chomsky Normal Form (CNF), as is the case with CKY. The unification-based chart parser has the same complexity as the CF version, but if any rule daughter can iterate, the input length factor worsens to n^n (i.e. exponential). The ANLT grammar contains several rules of this type (in its analysis of co-ordination constructions), but the exponential theoretical complexity does not seem to be a problem in practice (even in sentences containing co-ordination).

A further potential problem suggested by the complexity result is that the latter is related to the square of the number of categories in the grammar, and this number can be very large in a

realistic unification-based grammar. However, this also seems not to be a practical problem. A similar type of grammar dependency is evident in the time complexity of the non-deterministic LR parsing algorithm: the complexity is exponential on the size of the grammar (i.e. the total number of daughters in all the rules), since the number of LR(0) states is exponentially related to the grammar size (Johnson, 1989). Again, empirical results show that, although the parse table for the ANLT grammar is large and requires special techniques to represent it in a reasonable amount of space, parser throughput is nevertheless good.

Schabes (1991) presents a LR-like parse table for CF grammar which is used by a predictive parsing algorithm whose complexity is only linearly dependent on grammar size. This algorithm is intended to solve the problem of a non-deterministic LR parser theoretically behaving in the worst case in exponential time with respect to the grammar size for some ambiguous grammars and inputs. Schabes (1991:107) claims that "this problem is particularly acute for natural language processing since in this context the input length is typically small (10–20 words) and the grammar size very large (hundreds or thousands of rules and symbols)." The experiments reported in this thesis indicate that, with a wide-coverage NL grammar, inputs of this length can be parsed quite quickly (especially with an LR parser), but longer inputs (of more than about 30 words in length)—which occur relatively frequently in written text—are a problem. Unless grammar size takes on proportionately much more significance for such longer inputs, which seems unlikely, it appears that in fact the major problems do not lie in the area of grammar size, but in input length.

Thus, although the analysis of complexity may be useful for identifying potential inefficiencies in NL formalisms and algorithms, proposals for their improvement should be informed by empirical data derived from realistic implementations.

7.2 Parse Selection and Tuning Grammars to Corpora

A wide-coverage NL grammar can often license hundreds or thousands of analyses for a single sentence. The thesis describes a probabilistic system for parsing text taken from a specific corpus, for each sentence, returning the set of highest scoring syntactic analyses. The system is based around the LR parsing technique, and incorporates a novel, resource-efficient, and incremental approach to semi-supervised training, in which a grammar is 'tuned' to the corpus. Other new techniques include a method of probabilistically scoring sub-analyses that is sensitive to context, the accurate manipulation of probabilities during the course of an LR parse, and the identification of the highest ranked analyses in a unification-based parse forest without a full exploration of the search space. A preliminary experiment was carried out on a corpus of definitions from the *Longman Dictionary of Contemporary English* (Procter, 1978) to evaluate

the system's accuracy in selecting the correct analysis. The system identified the correct analysis for 76% of the 89 definitions in the training data between two and ten words in length (inclusive); for 91% of the definitions, the correct analysis was amongst the three highest ranked. The results for a further 55 unseen definitions were similar.

The probabilistic framework adopted by the system constitutes a *decision procedure*, rather than a *language model* (see section 6.1.3 for a discussion of this distinction with respect to parse selection). Although the system currently contains only syntactic knowledge, the framework will allow other sources of knowledge, such as lexical and semantic to be integrated relatively straightforwardly (see sections 7.2.2 and 7.2.3 below). Factoring such different kinds of knowledge into a language model would be much more difficult. However, the type of re-estimation used in the training of language models might be of benefit to this system when parsing unseen input, since the all-paths parsing that takes place during the re-estimation process generates more data points for a given corpus, and thus tends to smooth the probabilities. This is an area which warrants further study and experimentation.

7.2.2 Acquisition of Lexical Entries and Word Sense Frequencies

The probabilistic system described in the thesis does not take account of differing frequencies of occurrence of lexical entries; for example, in the LOB corpus the verb *believe* occurs with a finite sentential complement in 90% of citations, although it is grammatical with at least a further five patterns of complementation (Briscoe, personal communication). This type of lexical information, which is very likely to vary between sublanguages, should be integrated into a practical system. This would be straightforward in this system, since probabilities could be associated with each distinct lexical entry for a lexeme and be carried forward in the computation of the likelihood of each parse. However, acquiring the statistical information from which these probabilities may be derived is more difficult. Existing lexical taggers are unable to assign tags which reliably encode subcategorisation information. It seems likely that automatic acquisition of such information must await successful techniques for robust parsing of, at least, phrases in corpora[50].

In the experiment reported, syntactic peculiarities of the corpus surfaced quite rapidly, so that the manual addition of rules to the grammar during the training phase was practical (forming part of the 'tuning' process). Lexical idiosyncrasies are far less likely to be exhausted during training, though, suggesting that it will be necessary to develop an automatic method of dealing with them. One promising approach to this problem is to apply the same probabilistic techniques

[50] Brent (1991) claims to be able to recognise some subcategorisation patterns from large quantities of untagged text.

described in the thesis to a grammar of sequences of lexical tags, rather than of word strings, to form a robust phrasal parser. No lexical entries would then be needed, although the resulting analyses would necessarily be much less detailed than with the full ANLT grammar, and would not support subsequent semantic interpretation. However, the analyses could be used to help construct a more reliable and exhaustive lexicon for the full grammar. Fordham (1992) describes a system of this type, consisting of a relatively simple unification-based grammar designed to parse tag sequences drawn from the Spoken English Corpus (Taylor & Knowles, 1988), and produce analyses which are reasonably accurate at the phrasal level, but with any attachment ambiguity reduced to a single canonical binary-branching possibility. Fordham's system achieves a success rate with respect to correctness of rule application of approximately 96% on unseen data. An example of a tag sequence for which it returns the correct phrasal analysis (Fordham, personal communication) is:

> *The_AT government_NN is_VBZ comprised_VVN of_IO six_MC of_IO the_AT multitude_NN1 of_IO ethnic_JJ cross-ethnic_JJ and_CC political_JJ parties_NN2.*

A system based on the ANLT lexicon would be unlikely to analyse this sentence successfully if it occurred in unseen data as there would probably be no lexical entry for *cross-ethnic*, but this type of phrasal analysis of the corpus might allow a more adequate (corpus-specific) lexicon to be 'bootstrapped'.

7.2.3 Making Use of Collocational Information

The system described in the thesis provides a partial solution to the parse selection task. The results of the preliminary experiment show that the system is often able to rank highly the semantically and pragmatically correct analysis, just by exploiting the frequency of occurrence of the syntactic rules in the training data, and the syntactic contexts in which they appear. However, this approach will not succeed in all cases; for example, in the experiment, the system appears to develop a preference for local attachment of prepositional phrases (PPs) which is inappropriate in a significant number of cases. It is not surprising that probabilities based solely on syntax are not capable of resolving this type of ambiguity; in an input like *John saw the man on Monday again* it is the temporal interpretation of *Monday* which favours the adverbial interpretation (and thus non-local attachment). Such inputs are syntactically identical to ones like *John saw the man on the bus again* in which there is a mild preference for local attachment. Selecting the correct analysis in such cases would involve integrating information concerning word sense collocations into the probabilistic analysis. In this case, the applicable collocations are between the head of a PP complement, a preposition and the head of the phrase being postmodified. (The words will

not in general be adjacent, so approaches such as that of Church & Hanks (1989) cannot be used).

Hindle & Rooth (1991) report good results using a mutual information measure of collocation applied within a structurally-defined context, and their approach could be used in the system described here. Another way of integrating such 'structural' collocational information into the system would be to make use of the semantic component of the ANLT grammar. This component pairs with each distinct syntactic analysis a set of logical forms which represent, amongst other things, the predicate-argument structure of the input. In the resolution of PP attachment and similar ambiguities, it is 'collocation' at this level of representation which appears to be most relevant. In principle, integrating a probabilistic ranking of the resultant logical forms with the probabilistic ranking of the distinct syntactic analyses presents no problems. However, once again, the acquisition of the relevant statistical information would be difficult, requiring considerable quantities of analysed text as training material. One way to ameliorate the problem might be to reduce the size of the 'vocabulary' for which statistics need to be gathered by replacing lexical items with their superordinate terms (or a disjunction of such terms in the case of ambiguity). Copestake (1990, 1992) describes a program capable of deriving hierarchical taxonomies of word senses from a machine-readable dictionary. Taxonomies of this form might be used to replace PP complement heads and postmodified heads in corpus data with a smaller number of superordinate concepts. This would make the statistical data concerning trigrams of head-preposition-head less sparse (c.f. Gale & Church, 1990) and easier to gather from a corpus. Nevertheless, it is only possible to gather such data from determinately syntactically analysed material.

7.3 Dealing with Undergeneration

Although many missing rules can be identified during the interactive training phase, the probabilistic framework set out in the thesis does not completely solve the problem of gaps in coverage remaining when unseen corpus material is being analysed. A large part of the problem is detecting automatically when an input is not covered by the grammar. Some such inputs will not receive any analysis and will, therefore, be easy to spot. Many, though, will receive incorrect analyses (one of which will be automatically ranked as the most probable) and can, therefore, only be identified manually or perhaps on the basis of relative improbability. Jensen *et al.* (1983) describe an approach to parsing such inputs based on *parse fitting* or *rule relaxation* to deal with 'ill-formed' input. An approach of this type might work with inputs which receive no parse, but cannot help with the identification of those which only receive an incorrect one. In addition, it involves annotating each grammar rule about what should be relaxed and requires that semantic

interpretation is able to be extended to fitted or partial parses.

A technique with similar potential robustness to that described in section 7.2.2 above would be to infer a PCFG using Baum-Welch re-estimation from a given training corpus and predefined category set, following Lari & Young (1990) and Pereira & Schabes (1992). This approach has the advantage that the resulting grammar defines a well-defined set of analyses for which rules of compositional interpretation might be developed. However, the technique is limited in two main ways; firstly, such grammars are restricted to small (maximum about 15 non-terminal) CNF PCFGs because of the computational cost of iterative re-estimation with an algorithm polynomial in sentence length and non-terminal category size; and secondly, because some form of supervised training will be essential if the analyses assigned by the grammar are to be linguistically-motivated. Immediate prospects for applying such techniques to realistic NL grammars do not seem promising—the ANLT backbone grammar discussed in chapter 4 contains more than 550 categories. However, Briscoe & Waegner (1992) describe an experiment in which, firstly, Baum-Welch re-estimation was used in conjunction with other more linguistically-motivated constraints (such as 'headedness') on the class of grammars which could be inferred; and secondly, initial probabilities were heavily biased in favour of manually-coded, linguistically highly-plausible rules. This approach resulted in a simple tag sequence grammar often able to assign coherent and semantically and pragmatically plausible analyses to tag sequences drawn from the Spoken English Corpus.

An alternative approach, based in the framework described in the thesis, which can be considered to be the converse of Briscoe & Waegner's, is to start with an entirely hand-crafted 'core' grammar (such as the ANLT) and in cases in which a parse fails (which will be comparatively rare with a comprehensive lexicon), to automatically construct and add to the grammar one or more new rules, consistent with those extant, that will allow an analysis to be built[51]. With the LR technique, during the course of a parse, an inability to form a complete analysis of the input string is detected as soon as it possibly could be; in a non-deterministic LR parser of the type described in chapter 5, the states on the top of the graph-structured stack, in conjunction with the lookahead symbol, provide a constraining environment in which to conjecture a new rule. In addition, the probabilities attached to LR transitions give further information about the relative probabilities of valid transitions. For example, if the grammar did not contain a rule to analyse 'heavy NP movement' in English, i.e. in the ANLT framework the phrase structure rule,

```
Heavy_NP_Shift: VP --> H[SUBCAT NP_PP] PP NP.
```

and an instance of this construction was encountered in a corpus, the parser, after finding the

[51] This approach is related to the stochastic error-correcting parsing techniques described by Tanaka & Fu (1978), Fu (1982) and others.

head verb of a verb phrase subcategorising for an NP followed by a PP, will unexpectedly encounter a preposition (P) as the lookahead. At this point, all transitions in the table for lookahead P could be taken, a marker having been placed on the stack to indicate that a new rule to cover this discontinuity will be required later. The parse would then continue, but on each reduction that attempted to pop this marker off the stack, a new rule would be conjectured to analyse the constituents that were actually present. An additional source of constraint on the form of a new rule is provided by grammatical principles, for example, headedness, and also the 'shape' of propagation and default rules in the metagrammar (section 2.1): in this case, feature percolation will then also be consistent with rest of grammar. If several new rules were posited for a single failure, they could be ranked with respect to the probabilities assigned to the complete analyses in which they appeared.

False positives could be dealt with in a similar way by conjecturing a new rule when a transition about to be taken had an unusually low probability. New rules could perhaps be added to the system 'on the fly' by incrementally updating the parse table (e.g. Heering, Klint & Rekers, 1989)[52], thus making them available immediately for the analysis of subsequent sentences. This form of grammar 'learning' is incremental, in that it operates at exactly the point at which the parse has gone wrong and the repair is limited to that particular context, so the possibilities lie in a similarly constrained space to that of Berwick's (1985) approach. The alternative, post-hoc approach, operating over the whole of a parse forest or chart (e.g. Mellish's (1989) technique of hypothesising possible complete parses from the fragments in a chart built bottom-up) is much less constrained. If probabilities continue to be updated dynamically after training, the probabilistic framework provides a natural mechanism for determining whether a new rule should be kept or eventually discarded: a systematic extension will acquire a higher probability, but a one-off idiosyncrasy will retain its low initial probability.

7.4 Summary

This section reports very general conclusions arising from the research described in this thesis. The research concentrates on two of the most important issues for practical parsing of NL with wide-coverage unification-based grammars: parser throughput, and disambiguation.

When trying to maximise parser throughput, a purely bottom-up parser using a well-designed, appropriate unification algorithm and techniques for minimising processing overheads can perform significantly better in practice than comparable parsers employing simple types of top-down information. The standard LR technique can be applied in this area, and does provide

[52] Incremental parse table updating for grammars of the size under consideration might, however, prove to be intractable.

improved performance (although a precomputed Earley parser returns worse performance). Exponential worst-case complexity on the length of the input string does not prevent sentences of realistic text of up to about 25 words in length being parsed in reasonable time on conventional computer hardware. Exponential complexity on grammar size in the LR case does not appear to be an important factor.

Statistical methods can successfully be applied to the task of syntactic disambiguation to select analyses from which the semantic (and thus pragmatically) most plausible interpretations can straightforwardly be derived. The approach used, based on associating probabilities with transitions in an LR parse table, can encode information about the syntactic context in which rules are applied, and is efficiently implementable. In a limited-scale experiment involving supervised training, a system using this approach achieves an accuracy comparable to results reported for probabilistic CF grammar, but on the basis of a much smaller training corpus, and returning analyses that are more suitable for subsequent semantic interpretation.

Bibliography

Aho, A. (1968) "Indexed grammars – an extension to context-free grammars." *Journal of the ACM*, 15(4), 647–671.

Aho, A., R. Sethi & J. Ullman (1986) *Compilers: principles, techniques and tools*. Reading, MA: Addison-Wesley.

Aït Kaci, H. (1984) *A lattice-theoretic approach to computation based on a calculus of partially ordered types*. Ph.D. thesis, University of Pennsylvania, PA.

Alshawi, H. (1989) "Analysing the dictionary definitions." In B. Boguraev & E. Briscoe (eds.), *Computational Lexicography for Natural Language Processing*, Harlow, UK: Longman, 153–169.

Alshawi, H. (ed.) (1992) *The Core Language Engine*. Cambridge, MA: MIT Press.

Alshawi, H., D. Carter, R. Crouch, S. Pulman, M. Rayner & A. Smith (1992) *CLARE: a contextual reasoning and cooperative response framework for the Core Language Engine*. SRI International, Cambridge, UK.

Amsler, R. (1981) "A taxonomy for English nouns and verbs." In *Proceedings of the 19th Meeting of the Association for Computational Linguistics*, Stanford, CA, 133–138.

Ancona, M., G. Dodero, V. Gianuzzi & M. Morgavi (1991) "Efficient construction of LR(*k*) states and tables." *ACM Transactions on Programming Languages and Systems*, 13(1), 150–178.

Anderson, T., J. Eve & J. Horning (1973) "Efficient LR(1) parsers." *Acta Informatica*, 2(1), 12–39.

Baker, J. (1982) "Trainable grammars for speech recognition." In D. Klatt & J. Wolf (eds.), *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, ASA, 547–550.

Barstow, D., E. Shrobe & E. Sandewall (1984) *Interactive programming environments*. New York, NY: McGraw Hill.

Barton, G., R. Berwick & E. Ristad (1987) *Computational complexity and natural language*. Cambridge, MA: MIT Press.

Baum, L. (1972) "An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes." *Inequalities*, III, 1–8.

Bermudez, M. (1991) "A unifying model for lookahead LR parsing." *Computer Languages*, 16(2), 167–178.

Berwick, R. (1985) *The acquisition of syntactic knowledge*. Cambridge, MA: MIT Press.

Billot, S. & B. Lang (1989) "The structure of shared forests in ambiguous parsing." In *Proceedings of the 27th Meeting of the Association for Computational Linguistics*, Vancouver, Canada, 143–151.

Boguraev, B., J. Carroll, E. Briscoe & C. Grover (1988) "Software support for practical grammar development." In *Proceedings of the 12th Conference on Computational Linguistics (COLING)*, Budapest, Hungary, 54–58.

Boisen, S., Y. Chow, A. Haas, R. Ingria, A. Roucos, R. Scha, D. Stallard & M. Vilain (1989) *Integration of speech and natural language: final report*. BBN Systems and Technologies Corporation, Cambridge, MA, Report 6991.

Boyer, R. & J. Moore (1972) "The sharing of structure in theorem-proving programs." In B. Melzer & D. Mitchie (eds.), *Machine Intelligence 7*, New York, NY: Wiley, 101–116.

Brent, M. (1991) "Automatic acquisition of subcategorization frames from untagged text." In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, Berkeley, CA, 209–214.

Briscoe, E. (1987) *Modelling human speech comprehension: a computational approach*. Chichester, UK: Ellis Horwood; New York, NY: Wiley.

Briscoe, E. (1993) Review of Oostdijk (1991). *Computational Linguistics*, 19(1), 210–213.

Briscoe, E., C. Grover, B. Boguraev & J. Carroll (1987a) "A formalism and environment for the development of a large grammar of English." In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, Milan, Italy, 703–708.

Briscoe, E., C. Grover, B. Boguraev & J. Carroll (1987b) "Feature defaults, propagation and reentrancy." In E. Klein & J. van Benthem (eds.), *Categories, Polymorphism and Unification*, Centre for Cognitive Science, Edinburgh University, UK, 19–34.

Briscoe, E. & J. Carroll (1991) *Generalised probabilistic LR parsing of natural language (corpora) with unification-based grammars*. Computer Laboratory, Cambridge University, UK, Technical Report 224.

Briscoe, E. & J. Carroll (1993) "Generalised probabilistic LR parsing of natural language (corpora) with unification-based grammars." *Computational Linguistics*, 19(1), 25–59.

Briscoe, E. & N. Waegner (1992) "Robust stochastic parsing using the inside-outside algorithm." In *Proceedings of the AAAI Workshop on Statistically-based NLP Techniques*, San Jose, CA.

Busemann, S. & C. Hauenschild (1988) "A constructive view of GPSG or how to make it work." In *Proceedings of the 12th Conference on Computational Linguistics (COLING)*, Budapest, Hungary, 77–82.

Carpenter, R. (1992) *The logic of typed feature structures with applications to unification grammars, logic programs and constraint resolution*. Cambridge Tracts in Theoretical Computer Science 32, Cambridge, UK: Cambridge University Press.

Carpenter, R., C. Pollard & A. Franz (1991) "The specification and implementation of constraint-

based unification grammars." In *Proceedings of the 2nd International Workshop on Parsing Technologies*, Cancun, Mexico, 143–153.

Carroll, J. (1983) "An island parsing interpreter for the full augmented transition network formalism." In *Proceedings of the 1st Conference of the European Chapter of the Association for Computational Linguistics*, Pisa, Italy, 101–105.

Carroll, J. & E. Briscoe (1992) "Probabilistic normalisation and unpacking of packed parse forests for unification-based grammars." In *Proceedings of the AAAI Fall Symposium on Probabilistic Approaches to Natural Language*, Cambridge, MA, 33–38.

Carroll, J., E. Briscoe & C. Grover (1991) *A development environment for large natural language grammars*. Computer Laboratory, Cambridge University, UK, Technical Report 233.

Carroll, J. & C. Grover (1989) "The derivation of a large computational lexicon for English from LDOCE." In B. Boguraev & E. Briscoe (eds.), *Computational Lexicography for Natural Language Processing*, Harlow, UK: Longman, 117–134.

Chapman, N. (1987) *LR parsing: theory and practice*. Cambridge, UK: Cambridge University Press.

Charniak, E., C. Riesbeck & D. McDermott (1980) *Artificial intelligence programming*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Chitrao, M. & R. Grishman (1990) "Statistical parsing of messages." In *Proceedings of the DARPA Speech and Natural Language Workshop*, Hidden Valley, PA, 263–266.

Chomsky, N. (1959) "On certain formal properties of grammars." *Information and Control*, 2(2), 137–167.

Church, K. & P. Hanks (1989) "Word association norms, mutual information, and lexicography." In *Proceedings of the 27th Meeting of the Association for Computational Linguistics*, Vancouver, Canada, 76–83.

Church, K. & R. Patil (1982) "Coping with syntactic ambiguity or how to put the block in the box on the table." *Computational Linguistics*, 8, 139–149.

Clocksin, W. & C. Mellish (1984) *Programming in Prolog*. Berlin, Germany: Springer-Verlag.

Copestake, A. (1990) "An approach to building the hierarchical element of a lexical knowledge base from a machine-readable dictionary." In *Proceedings of the ITK International Workshop on Inheritance in Natural Language Processing*, Tilburg, The Netherlands, 19–29.

Copestake, A. (1992) "The ACQUILEX LKB: representation issues in semi-automatic acquisition of large lexicons." In *Proceedings of the 3rd ACL Conference on Applied Natural Language Processing*, Trento, Italy, 88–96.

Culik, K. II & R. Cohen (1973) "LR-regular grammars—an extension of LR(k) grammars." *Journal of Computer and System Sciences*, 7, 66–96.

Cutting, D., J. Kupiec, J. Pedersen & P. Sibun (1992) "A practical part-of-speech tagger." In *Proceedings of the 3rd ACL Conference on Applied Natural Language Processing*, Trento, Italy, 133–140.

de Marcken, C. (1990) "Parsing the LOB corpus." In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, Pittsburgh, PA, 243–251.

DeRemer, F. & T. Pennello (1982) "Efficient computation of LALR(1) look-ahead sets." *ACM Transactions on Programming Languages and Systems*, 4(4), 615–649.

de Rose, S. (1988) "Grammatical category disambiguation by statistical optimization." *Computational Linguistics*, 14(1), 31–39.

Earley, J. (1970) "An efficient context-free parsing algorithm." *Communications of the ACM*, 13(2), 94–102.

Emele, M. (1991) "Unification with lazy non-redundant copying." In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, Berkeley, CA, 323–330.

Estival, D. (1990) *ELU user manual*. ISSCO, Geneva, Switzerland, Technical Report 1.

Evans, R. (1985) "ProGram—a development tool for GPSG grammars." *Linguistics*, 23(2), 213–243.

Fordham, A. (1992) "Probabilistic phrasal parsing of natural language corpora." Unpublished M.Phil. dissertation, Computer Laboratory, Cambridge University, UK.

Frazier, L. (1988) "Grammar and language processing." In F. Newmeyer (ed.), *Linguistics: The Cambridge Survey 2*, Cambridge, UK: Cambridge University Press, 14–45.

Fu, K-S. (1982) "Error-correcting parsing for string languages." In *Syntactic Pattern Recognition and Applications*, London, UK: Prentice-Hall, 246–275.

Fujisaki, T., F. Jelinek, J. Cocke, E. Black & T. Nishino (1989) "A probabilistic method for sentence disambiguation." In *Proceedings of the 1st International Workshop on Parsing Technologies*, Pittsburgh, PA, 105–114.

Gale, W. & K. Church (1990) "Poor estimates of context are worse than none." In *Proceedings of the DARPA Speech and Natural Language Workshop*, Hidden Valley, PA, 283–287.

Garside, R., G. Leech & G. Sampson (eds.) (1987) *The computational analysis of English: a corpus-based approach*. Harlow, UK: Longman.

Gazdar, G., E. Klein, G. Pullum & I. Sag (1985) *Generalized Phrase Structure Grammar*. Oxford, UK: Blackwell.

Gazdar, G. & C. Mellish (1989) *Natural language processing in Lisp*. Wokingham, UK: Addison-Wesley.

Godden, K. (1990) "Lazy unification." In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, Pittsburgh, PA, 180–187.

Good, I. (1953) "The population frequencies of species and the estimation of population parameters." *Biometrica*, 40, 237–264.

Graham, S., M. Harrison & W. Ruzzo (1980) "An improved context-free recognizer." *ACM Transactions on Programming Languages and Systems*, 2(3), 415–462.

Grosch, J. (1990) "Lalr—a generator for efficient parsers." *Software—Practice and Experience*, 20(11), 1115–1135.

Grosz, B., D. Appelt, M. Douglas & F. Pereira (1987) "TEAM: An experiment in the design of transportable natural language interfaces." *Artificial Intelligence*, 32(2), 173–244.

Grover, C., E. Briscoe, J. Carroll, & B. Boguraev (1989) *The Alvey Natural Language Tools grammar (second release)*. Computer Laboratory, Cambridge University, UK, Technical Report 162.

Grover, C., J. Carroll & E. Briscoe (1993) *The Alvey Natural Language Tools grammar (4th release)*. Computer Laboratory, Cambridge University, UK, Technical Report 284.

Heering, J., P. Klint & J. Rekers (1989) "Incremental generation of parsers." In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, Portland, OR, 179–191.

Hektoen, E. (1991) *LAR processing for natural language grammars*. Unpublished M.Phil. Dissertation, Computer Laboratory, Cambridge University, UK.

Hindle, D. (1989) "Acquiring disambiguation rules from text." In *Proceedings of the 27th Meeting of the Association for Computational Linguistics*, Vancouver, Canada, 118–125.

Hindle, D. & M. Rooth (1991) "Structural ambiguity and lexical relations." In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, Berkeley, CA, 229–236.

Hirst, G. (1987) *Semantic interpretation and the resolution of ambiguity*. Cambridge, UK: Cambridge University Press.

Holmes, J. (1988) *Speech synthesis and recognition*. Wokingham, UK: Van Nostrand Reinhold.

Jaffar, J. (1984) "Efficient unification over infinite terms." *New Generation Computing*, 2, 207–219.

Jensen, K., G. Heidorn, L. Miller & Y. Ravin (1983) "Parse fitting and prose fixing: getting a hold on ill-formedness." *Computational Linguistics*, 9, 147–153.

Jensen, K., G. Heidorn, S. Richardson & N. Haas (1986) *PLNLP, PEG and CRITIQUE: three contributions to computing in the humanities*. IBM, Yorktown Heights, NY, Research Report RC 11841.

Johnson, M. (1989) "The computational complexity of Tomita's algorithm." In *Proceedings of the 1st International Workshop on Parsing Technologies*, Pittsburgh, PA, 203–208.

Johnson, R. & C. Rupp (1992) "Evaluating complex constraints in linguistic formalisms." In *Proceedings of Workshop W2 at the 10th European Conference on Artificial Intelligence: Coping with linguistic ambiguity in typed feature formalisms*, Vienna, Austria, 39–48.

Johnson, S. (1975) *Yacc—yet another compiler-compiler*. Bell Telephone Laboratories, Murray

Hill, NJ, Computer Science Technical Report 32.

Kaplan, R. & J. Bresnan (1982) "Lexical-Functional Grammar: a formal system for grammatical representation." In J. Bresnan (ed.), *The Mental Representation of Grammatical Relations*, Cambridge, MA: MIT Press, 173–281.

Karttunen, L. (1986) "D-PATR: A development environment for unification-based grammars." In *Proceedings of the 11th International Congress on Computational Linguistics (COLING)*, Bonn, Germany, 74–80.

Kasami, J. (1965) *An efficient recognition and syntax analysis algorithm for context-free languages*. Air Force Cambridge Research Laboratory, Bedford, MA, Report AFCRL-65-758.

Kay, M. (1985) "Parsing in Functional Unification Grammar." In D. Dowty, L. Karttunen & A. Zwicky (eds.), *Natural Language Parsing*, Cambridge, UK: Cambridge University Press.

Kay, M. (1986) "Algorithm schemata and data structures in syntactic parsing." In B. Grosz, K. Sparck-Jones & B. Webber (eds.), *Readings in Natural Language Processing*, Los Altos, CA: Morgan Kaufmann, 35–70.

Kay, M. (1989) "Head-driven parsing." In *Proceedings of the 1st International Workshop on Parsing Technologies*, Pittsburgh, PA, 52–62.

Kilbury, J. (1986) "Category cooccurence restrictions and the elimination of metarules." In *Proceedings of the 11th Conference on Computational Linguistics (COLING)*, Bonn, Germany, 50–55.

Kimball, J. (1973) "Seven principles of surface structure parsing in natural language." *Cognition*, 2(1), 15–47.

Kiparsky, C. (1985) *LFG manual*. XEROX Palo Alto Research Center, Palo Alto, CA, Ms.

Kipps, J. (1989) "Analysis of Tomita's algorithm for general context-free parsing." In *Proceedings of the 1st International Workshop on Parsing Technologies*, Pittsburgh, PA, 193–202.

Klein, E. & M. Martin (1989) "The parser generating system PGS." *Software—Practice and Experience*, 19(11), 1015–1028.

Knuth, D. (1965) "On the translation of languages from left to right." *Information and Control*, 8, 607–639.

Kogure, K. (1990) "Strategic lazy incremental copy graph unification." In *Proceedings of the 13th Conference on Computational Linguistics (COLING)*, Helsinki, Finland, 223–228.

Koskenniemi, K. (1983) "Two-level model for morphological analysis." In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, Karlsruhe, Germany, 683–685.

Koster, C. (1991) "Affix grammars for natural languages." In *Attribute Grammars, Applications and Systems, Lecture Notes in Computer Science 545*, Berlin, Germany: Springer-Verlag, 358–373.

Kristensen, B. & O. Madsen (1981) "Methods for computing LALR(k) lookahead." *ACM*

Transactions on Programming Languages and Systems, 3(1), 60–82.

Kuno, S. (1965) "The predictive analyzer and a path elimination technique." *Communications of the ACM*, 8(7), 453–462.

Kupiec, J. (1991) "A trellis-based algorithm for estimating the parameters of a hidden stochastic context-free grammar." In *Proceedings of the DARPA Speech and Natural Language Workshop*, Asilomar, CA.

LaLonde, W., E. Lee & J. Horning (1971) "An LALR(k) parser generator." In *Proceedings of the IFIP Congress 71*, New York, NY: Elsevier, 151–153.

Lang, B. (1974) "Deterministic techniques for efficient non-deterministic parsers." In J. Loeckx (ed.), *Automata, Languages and Programming, Lecture Notes in Computer Science 14*, Berlin, Germany: Springer-Verlag, 255–269.

Lang, B. (1992) "Recognition can be harder than parsing." In *Proceedings of the 2nd International Workshop on Tree Adjoining Grammars*, Philadelphia, PA. Also to appear in *Computational Intelligence*, 1994.

Lankhorst, M. (1991) "An empirical comparison of generalized LR tables." In R. Heemels, A. Nijholt & K. Sikkel (eds.), *Proceedings of the 1st Twente Workshop on Language Technology*, 87–93.

Lari, K. & S. Young (1990) "The estimation of stochastic context-free grammars using the inside-outside algorithm." *Computer Speech and Language Processing*, 4, 35–56.

Leech, G. & R. Garside (1991) "Running a grammar factory: the production of syntactically analysed corpora or 'treebanks.'" In S. Johansson & A. Stenström (eds.), *English Computer Corpora: Selected Papers and Research Guide*, Berlin, Germany: Mouton de Gruyter, 15–32.

Leermakers, R. (1989) "How to cover a grammar." In *Proceedings of the 27th Meeting of the Association for Computational Linguistics*, Vancouver, Canada, 135–142.

Leermakers, R. (1991) "Recursive ascent parsing." In R. Heemels, A. Nijholt & K. Sikkel (eds.), *Proceedings of the 1st Twente Workshop on Language Technology*, 9–19.

Magerman, D. & M. Marcus (1991) "Pearl: a probabilistic chart parser." In *Proceedings of the 2nd International Workshop on Parsing Technologies*, Cancun, Mexico, 193–199.

Magerman, D. & C. Weir (1992) "Efficiency, robustness and accuracy in picky chart parsing." In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, Newark, Delaware, 40–47.

Marcus, M. (1980) *A theory of syntactic recognition for natural language*. Cambridge, MA: MIT Press.

Martelli, A. & Montanari, U. (1982) "An efficient unification algorithm." *ACM Transactions on Programming Languages and Systems*, 4(2), 258–282.

Maxwell, J. III & R. Kaplan (in press) "The interface between phrasal and functional

constraints." To appear in *Computational Linguistics*.

Mellish, C. (1989) "Some chart-based techniques for parsing ill-formed input." In *Proceedings of the 27th Meeting of the Association for Computational Linguistics*, Vancouver, Canada, 102–109.

Meteer, M., R. Schwartz & R. Weischedel (1991) "POST: Using probabilities in language processing." In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, Sydney, Australia, 960–965.

Moore, R. & J. Dowding (1991) "Efficient bottom-up parsing." In *Proceedings of the DARPA Speech and Natural Language Workshop*, Asilomar, CA, 200–203.

Nakazawa, T. (1991) "An extended LR parsing algorithm for grammars using feature-based syntactic categories." In *Proceedings of the 5th European Conference of the Association for Computational Linguistics*, Berlin, Germany, 69–74.

Nederhof, M-J. (1992) *Generalized left-corner parsing*. Katholieke Universiteit Nijmegen, Department of Informatics, Technical Report 92-21.

Nederhof, M-J., C. Koster, C. Dekkers & A. van Zwol (1992) "The grammar workbench: a first step towards lingware engineering." In W. ter Stal, A. Nijholt & R. op den Akker (eds.), *Proceedings of the 2nd Twente Workshop on Language Technology*, 103–115.

Newman, P. (1988) "Combinatorial disambiguation." In *Proceedings of the 2nd ACL Conference on Applied Natural Language Processing*, Austin, TX, 243–252.

Ng, S-K. & M. Tomita (1991) "Probabilistic parsing for general context-free grammars." In *Proceedings of the 2nd International Workshop on Parsing Technologies*, Cancun, Mexico, 154–163.

Nunberg, G. (1990) *The linguistics of punctuation*. Chicago, IL: University of Chicago Press.

Oostdijk, N. (1991) *Corpus linguistics and the automatic analysis of English*. Amsterdam, The Netherlands: Rodopi.

Osborne, M. (1990) *Corpus parsing*. Unpublished M.Phil. dissertation, Computer Laboratory, Cambridge University, UK.

Paterson, M. & M. Wegman (1978) "Linear unification." *Journal of Computer and System Sciences*, 16, 158–167.

Pereira, F. (1985) "A structure-sharing representation for unification-based grammar formalisms." In *Proceedings of the 23rd Meeting of the Association for Computational Linguistics*, Chicago, IL, 137–144.

Pereira, F. & Y. Schabes (1992) "Inside-outside re-estimation for partially bracketed corpora." In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, Newark, Delaware, 128–135.

Pereira, F. & S. Shieber (1987) *Prolog and natural language analysis*. Chicago, IL: University of Chicago Press.

Pereira, F. & D. Warren (1980) “Definite clause grammars for language analysis—a survey of the formalism and a comparison with augmented transition networks.” *Artificial Intelligence*, 13(3), 231–278.

Pereira, F. & D. Warren (1983) “Parsing as deduction.” In *Proceedings of the 21st Meeting of the Association for Computational Linguistics*, Cambridge, MA, 137–144.

Pezaris, S. (1989) “Shift-reduce conflicts in LR parsers.” *ACM SIGPLAN Notices*, 24(11), 94–95.

Phillips, J. (1986) “A simple, efficient parser for phrase-structure grammars.” *AISB Quarterly*, 59, 14–18.

Phillips, J. & H. Thompson (1985) “GPSGP—a parser for Generalized Phrase Structure Grammars.” *Linguistics*, 23(2), 245–261.

Phillips, J. & H. Thompson (1987) “A parser and an appropriate computational representation for GPSG.” In N. Haddock, E. Klein & G. Morrill (eds.), *Edinburgh Working Papers in Cognitive Science, 1: Categorial Grammar, Unification Grammar, and Parsing*, Centre for Cognitive Science, Edinburgh University, UK, 115–136.

Pollard, C. & I. Sag (1987) *Information-based syntax and semantics: volume 1—fundamentals*. Chicago, IL: University of Chicago Press.

Pratt, V. (1975) “LINGOL – a progress report.” In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, Tbilisi, USSR, 422–428.

Procter, P. (ed.) (1978) *The Longman dictionary of contemporary English*. Harlow, UK: Longman.

Purdom, P. (1974) “The size of LALR(1) parsers.” *BIT*, 14, 326–337.

Ritchie, G., S. Pulman, G. Russell, & A. Black (1987) “A computational framework for lexical description.” *Computational Linguistics*, 13, 290–307.

Ritchie, G., G. Russell, A. Black & S. Pulman (1992) *Computational morphology*. Cambridge, MA: MIT Press.

Robinson, J. (1982) “DIAGRAM: a grammar for dialogues.” *Communications of the ACM*, 25(1), 27–47.

Sampson, G. (1987) “The grammatical database and parsing scheme.” In R. Garside, G. Leech and G. Sampson (eds.), *The Computational Analysis of English: A Corpus-based Approach*, Harlow, UK: Longman, 82–96.

Sampson, G., R. Haigh, & E. Atwell (1989) “Natural language analysis by stochastic optimization: a progress report on Project APRIL.” *Journal of Experimental and Theoretical Artificial Intelligence*, 1, 271–287.

Santorini, B. (1990) *Penn Treebank tagging and parsing manual*. CIS Department, University of Pennsylvania, Ms.

Schabes, Y. (1991) “Polynomial time and space shift-reduce parsing of arbitrary context-free grammars.” In *Proceedings of the 29th Annual Meeting of the Association for*

Computational Linguistics, Berkeley, CA, 106–113.

Schubert, L. (1986) "Are there preference trade-offs in attachment decisions." In *Proceedings of the 5th National Conference on Artificial Intelligence (AIAI-86)*, Philadelphia, PA, 601–605.

Sedgewick, R. (1988) *Algorithms*. Wokingham, UK: Addison-Wesley.

Sharman, R. (1989) *Probabilistic ID/LP grammars for English*. IBM UK Scientific Centre, Winchester, UK, Report 217.

Sharman, R. (1991) *An introduction to language modelling*. IBM UK Scientific Centre, Winchester, UK, Ms.

Sharman, R., F. Jelinek & R. Mercer (1990) "Generating a grammar for statistical training." In *Proceedings of the DARPA Speech and Natural Language Workshop*, Hidden Valley, PA, 267–274.

Shieber, S. (1983) "Disambiguation by a shift-reduce parsing technique." In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, Cambridge, MA, 113–118.

Shieber, S. (1984) "The design of a computer language for linguistic information." In *Proceedings of the 10th International Conference on Computational Linguistics (COLING)*, Stanford, CA, 362–366.

Shieber, S. (1985) "Using restriction to extend parsing algorithms for complex feature-based formalisms." In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, Chicago, IL, 145–152.

Shieber, S. (1986a) *An introduction to unification-based approaches to grammar*. CSLI, Stanford, CA, Lecture Notes Number 4; also Chicago, IL: University of Chicago Press.

Shieber, S. (1986b) "A simple reconstruction of GPSG." In *Proceedings of the 11th International Conference on Computational Linguistics (COLING)*, Bonn, Germany, 211–215.

Shieber, S. (1987) "Separating linguistic analyses from linguistic theories." In P. Whitelock, M. Wood, H. Somers, R. Johnson & P. Bennet (eds.), *Linguistic Theory and Computer Applications*, London, UK: Academic Press, 1–36.

Shieber, S. (1988) *CL-PATR user's manual*. SRI, Menlo Park, CA, Ms.

Spector, D. (1983) "Lexing and parsing Modula-2." *ACM SIGPLAN Notices*, 18(10), 25–32.

Spector, D. (1988) "Efficient full LR(1) parser generation." *ACM SIGPLAN Notices*, 23(12), 143–149.

Steel, S. & A. De Roeck (1987) "Bi-directional chart parsing." In C. Mellish & J. Hallam (eds.), *Advances in Artificial Intelligence (Proceedings of AISB-87)*, Chichester, UK: Wiley, 223–235.

Steele, G. (1984) *Common Lisp: the language*. Bedford, MA: Digital Press.

Stock O., R. Falcone & P. Insinnamo (1988) "Island parsing and bidirectional charts." In

Proceedings of the 12th Conference on Computational Linguistics (COLING), Budapest, Hungary, 636–641.

Tanaka, E. & K-S. Fu (1978) "Error-correcting parsers for formal languages." *IEEE Transactions on Computers*, c-27, 605–616.

Taylor, L., C. Grover & E. Briscoe (1989) "The syntactic regularity of English noun phrases." In *Proceedings of the 4th European Meeting of the Association for Computational Linguistics*, Manchester, UK, 256–263.

Taylor, L. & G. Knowles (1988) *Manual of information to accompany the SEC corpus: the machine-readable corpus of spoken English*. University of Lancaster, UK, Ms.

Thompson, H. (1983) "MCHART—a flexible, modular chart parsing framework." In *Proceedings of the National Conference on Artificial Intelligence (AIAI-83)*, Washington, DC, 408–410.

Tomabechi, H. (1991) "Quasi-destructive graph unification." In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, Berkeley, CA, 315–322.

Tomita, M. (1984) "Parsers for natural languages." In *Proceedings of the 10th International Conference on Computational Linguistics (COLING)*, Stanford, CA, 354–357.

Tomita, M. (1985) "An efficient context-free parsing algorithm for natural languages." In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, Los Angeles, CA, 756–764.

Tomita, M. (1986) *Efficient parsing for natural language*. Boston, MA: Kluwer.

Tomita, M. (1987) "An efficient augmented-context-free parsing algorithm." *Computational Linguistics*, 13(1), 31–46.

Valiant, L. (1975) "General context free recognition in less than cubic time." *Journal of Computing Systems Science*, 10, 308–315.

Vijay-Shanker, K. & A. Joshi (1985) "Some computational properties of Tree Adjoining Grammars." In *Proceedings of the 23rd Meeting of the Association for Computational Linguistics*, Chicago, IL, 82–93.

Vijay-Shanker, K. & D. Weir (1990) "Polynomial time parsing of Combinatory Categorial Grammars." In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, Pittsburgh, PA, 1–8.

Viterbi, A. (1967) "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm." *IEEE Transactions on Information Theory*, IT-13, 260–269.

White, J. (1987) "The research environment in the METAL project." In S. Nirenburg (ed.), *Machine Translation: Theoretical and Methodological Issues*, Cambridge, UK: Cambridge University Press, 225–246.

Whittemore, G. & K. Ferrara (1990) "Empirical study of predictive powers of simple attachment schemes for post-modifier prepositional phrases." In *Proceedings of the 28th Annual*

Meeting of the Association for Computational Linguistics, Pittsburgh, PA, 23–30.

Wilks, Y., X. Huang & D. Fass (1985) "Syntax, preference, and right attachment." In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, Los Angeles, CA, 779–784.

Woods, W. (1970) "Transition network grammars for natural language analysis." *Communications of the ACM*, 13(8), 591–606.

Wright, J. (1990) "LR parsing of probabilistic grammars with input uncertainty for speech recognition." *Computer Speech and Language*, 4, 297–323.

Wright, J. (1992) "Introduction to probabilistic grammars." In *Proceedings of the ELSNET Workshop on Sublanguage Grammar and Lexicon Acquisition for Speech and Language Processing*, HCRC, University of Edinburgh, UK, 27–32.

Wright, J. & E. Wrigley (1989) "Probabilistic LR parsing for speech recognition." In *Proceedings of the 1st International Workshop on Parsing Technologies*, Pittsburgh, PA, 193–202.

Wright, J., E. Wrigley & R. Sharman (1991) "Adaptive probabilistic generalized LR parsing." In *Proceedings of the 2nd International Workshop on Parsing Technologies*, Cancun, Mexico, 154–163.

Wroblewski, D. (1987) "Nondestructive graph unification." In *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI-87)*, Seattle, WA, 582–587.

Younger, D. (1967) "Recognition and parsing of context-free languages in time n^3." *Information and Control*, 10(2), 189–208.

Zajac, R. (1992) "Inheritance and constraint-based grammar formalisms." *Computational Linguistics*, 18(2), 159–180.

Zeevat, H., E. Klein & J. Calder (1987) "Unification Categorial Grammar." In N. Haddock, E. Klein & G. Morrill (eds.), *Edinburgh Working Papers in Cognitive Science, 1: Categorial Grammar, Unification Grammar, and Parsing*, Centre for Cognitive Science, Edinburgh University, UK, 195–222.

Appendix A

Below are the four sets of sentences used in the experiments reported in chapters 3 and 5.

decl

```
he doesn't help.
he confidently accepted their conditions.
don't help him!
do be more resolute!
help me!
he accepted their conditions confidently.
in the abbey he helped the abbot.
he helped the abbot in the abbey.
in an anxious mood he helped the abbot.
he helped the abbot in an anxious mood.
with some anxiety he helped the abbot.
he helped the abbot with some anxiety.
without a doubt he helped the abbot.
he helped the abbot without a doubt.
he helped the abbot without a doubt but with some anxiety.
he helped them down at the abbey.
this mood of lee's is not very characteristic.
he couldn't help hearing that admission of the abbot's.
she anticipates that he will help.
she anticipates he will help.
it appears that she knows him.
it appears she knows him.
that he didn't help mattered.
it mattered that he didn't help.
it mattered he didn't help.
he promised her that he would help.
he promised her he would help.
that he apologized amuses her.
she agrees with him that he should help.
that lee helps matters to her.
she is busy and in the mood.
she is not busy and in the mood.
there is an abbot in the abbey.
it is to the abbot that he gives the message.
he is agreed with.
she helps busily.
```

she busily helps.
she helps in the abbey.
she is busily helping.
she helps busily and with abandon.
she does not help.
whose abacus is this?
which of the abbots is here?
which abbot did you see?
the abbot my doubts about whom they are aware of is here.

wh/slash

which abbot do I have my doubts about?
which abbot do I agree with lee's doubts about?
which abbot do I have some doubts about?
my doubts all of which are crazy don't help me.
who did he have a desire to help?
who did he have a characteristic desire to help?
how confident is he?
who is he eager to help?
who is he resolutely eager to help?
to whom was it clear that he wouldn't apologize?
who is he certain to help?
who is he busy helping?
who was he abandoned by?
who does she know to be crazy?
what did she anticipate him doing?
who did she anticipate helping?
who could she hear apologizing?
who did she see help the abbot?
who did she see accepted by them?
which abbey is he in?
in which abbey is he?
how crazy is he?
what was there?
who was it that he helped?
who was it who he helped?
with whom is it that he agrees?
what was she not appearing to do?
what was she appearing not to do?
how can she help him?
how easily can she do it?
in which abbey did he see her?
how easily was he being abandoned?
who will help and who won't help?
the abbot lee knows but kim doesn't know.

which abbot does lee know but kim doesn't know?
which abbot and which abbey did you see?
what do you have a desire to do but anxieties about?
what is he scared to do but resolute that he will do?
in which abbey and with which abbot did he see them?
what does lee see and hear?

passive

he is abandoned.
he is abandoned by her.
he is given back his abacus.
he is given back his abacus by kim.
he is allowed into the abbey.
he is allowed into the abbey by them.
he is allowed by them into the abbey.
she was promised that he would help.
she was promised by him that he would help.
she was promised that he would help by him.
she was bet an abacus that he could do it.
she was bet an abacus by him that he could do it.
he was asked what he would do.
he was asked by them what he would do.
he was asked whether he would help.
he was asked by them whether he would help.
he was asked what to do.
he was asked by them what to do.
he was asked whether to help.
he was asked by them whether to help.

conj

lee won't help but he will allow kim to help.
lee won't help and kim won't apologize.
either kim helps or lee helps.
he is crazy and anxious.
he is crazy or anxious.
he is crazy and in the abbey.
kim and lee are here.
kim and lee and sandy are here.
kim lee and sandy are here.
both kim and sandy are here.
either kim or lee is here.
the abbot or his host has apologized.
the abbot or his host have apologized.

the abbots or lee have apologized.
lee or the abbots have apologized.
the abbots or their hosts have apologized.
neither the abbots nor their hosts are here.
neither the abbot nor his host are here.
neither the abbots nor their host are here.
neither their host nor the abbots are here.
either the abbots or the hosts are here.
either the abbot or his host are here.
his crazy doubt or anxiety is not helping him.
his crazy doubts or anxieties are not helping him.

Appendix B

Below are the full results of the experiment reported in chapter 6 with the LDOCE noun definition corpus. Column one: ranking of the correct parse (0 if the correct parse is not ranked in the top three); column two: total number of analyses.

Definitions of Length 2–10 Words Reparsed from the Training Corpus

```
1   3  freedom from control
1   1  monastery or convent
1   6  the group of people living in such a building
3   3  the alphabet as taught to children
1   1  sudden forgetfulness
1   1  an example of this
1   1  the act of washing oneself
1   2  an australian aborigine
1   2  place where one lives
1   1  the act of putting an end to something
2   4  great hatred
1  13  a very hateful or nasty thing or act
1   8  an example of this by accident or intention
3   4  a person esp a doctor who intentionally causes an abortion
0 144  a complete change to the opposite position direction or opinion
0   6  loss of surface by rubbing
1   1  the act of making shorter
3   4  the state of being away or of not being present
1   6  loss of attention to what one is doing
1   2  a person who stays away
0  70  regular absence without good cause esp from work or duty
2   3  a bitter green very strong alcoholic drink
1   1  a glass of this
1   4  a shortened form of a statement speech etc
1   4  an idea of a quality as separate from any object
1   2  a great quantity
1   4  wrong use
1   4  a great hole which appears to have no bottom
1   1  a member of an academy
1  32  a school for training in a special art or skill
1   1  the act of increasing speed
2   3  the act of accepting or of being accepted
3   3  in business an agreement to pay
```

```
1   2  means of entering
1   1  way in
1  15  the act of coming to a position office state etc
1   1  something added
1   1  an addition
1   3  agreement as to a demand
1   4  strong expressions of approval and praise
1   4  loud shouts of welcome and honour
1   1  an upward slope
1   6  strong praise and approval
0  44  room flat house hotel room etc
1   1  something that helps or makes an action easier
1   5  something which is usually or often found with something else
1   2  a person who plays a musical accompaniment
1   4  a person who helps one to do wrong
1   1  the act of finishing work completely and successfully
1   1  something completely and successfully done
1   1  a lady-like art
1   4  an agreement between countries businesses etc
3   6  matter material etc that has come together or grown
2   3  a box-like apparatus in which electrical power can be stored
1   1  the quality of being accurate
2   5  a person of the highest class or skill in something
1  11  bitterness of speech temper or manner
1   1  an example of this
1   2  a chemical made from acetic acid
1   1  a continuous pain
1   5  the quality of being acid
1   4  something esp a liquid with an acid taste
0   6  an antiaircraft gun or fire from such a gun
0   6  a statement letter etc saying that something has been received
2  17  the highest or greatest point of development ability success etc
1   4  a person who helps a priest to perform religious ceremonies
1   1  an attendant
1   3  a medicine made from one of these plants
0  12  the state when one's feelings and actions are uncontrolled
1   6  the group of people living in such a building
1   1  the act of making shorter
1   3  powers and skills esp of the mind
1  12  the lowest temperature that is thought to be possible
1   1  the wood of this tree
1   1  a jew
1  11  two animals joined together by such a bar
1   4  power control etc
1  10  a simple or foolish country man
1   2  the yellow central part of an egg
1   4  a very long time
```

```
0  12  a type of short dog with fairly long hair
1   6  young people considered as a group
1   2  young animals
1   4  a young person esp a boy
1   3  early life
2  16  young men and women considered as a group
2   2  a long loud cry
1   1  something that binds people or things together
```

Parses of Unseen Definitions

```
1   1  something needing action
1   1  a happening
1   4  behaviour which is not one's natural manner
1   7  a feeling or manner that is pretended
1   1  something which causes suffering
1   1  a person from africa
2   6  the power or force which causes a result
1   3  one of the periods of life
1   2  a long time
1   1  something that annoys one
1   2  thinking in the same way
0   7  a person or thing that supports or helps
3   5  the act of directing a weapon remark etc
1   2  a dog with a rough coat
1   1  type of large terrier
1   4  the act of allowing fresh air into a room
0   3  any letter that is sent by air
1   1  a large passenger aircraft
0   6  letters parcels etc sent by air
1   4  the system of sending things by air
0   8  the speed at which an aircraft travels through the air
1  10  the sky or the space above the ground
1   3  a very gentle wind
1   2  an attack by military aircraft
1   4  a case or example of this work
1   2  a female airman
3   4  an animal or plant that lacks the typical colouring
1  10  the white or colourless part of an egg
0   1  a person who studied or practised alchemy
1   1  the drinks containing this
1   1  false name
1   3  the muslim name for god
1   7  the style of such a story poem etc
2   2  a union of families by marriage
1   4  a share as of money or space
```

```
1   4  money provided for a special purpose
1   1  a person who helps or supports one
1   2  something that attracts or charms
1   2  quantity or sum
0   7  total quantity or sum
2   2  a sexual relationship esp one that is secret
1   1  large amount
1   3  the state of being amused
1   6  the act or action of producing this state
0   3  a person who believes in anarchism
1   6  absence of government or control
1   2  absence of order
1   1  something hated
1   1  a person skilled in anatomy
1   3  a place where ships may anchor
2   2  the money charged to anchor in a harbour
1   1  a means of making firm
1   1  an old man
0   3  a person who helps another in work
1   2  a machine for measuring the strength of wind
```