June 1997

Complexity of Lexical Descriptions and its Relevance to Partial Parsing

Srinivas Bangalore
University of Pennsylvania

University of Pennsylvania Institute for Research in Cognitive Science Technical Report No. IRCS-97-10.

Complexity of Lexical Descriptions and its Relevance to Partial Parsing

Abstract

In this dissertation, we have proposed novel methods for robust parsing that integrate the flexibility of linguistically motivated lexical descriptions with the robustness of statistical techniques. Our thesis is that the computation of linguistic structure can be localized if lexical items are associated with rich descriptions (*supertags*) that impose complex constraints in a local context. However, increasing the complexity of descriptions makes the number of different descriptions for each lexical item much larger and hence increases the local ambiguity for a parser. This local ambiguity can be resolved by using supertag co-occurrence statistics collected from parsed corpora. We have explored these ideas in the context of Lexicalized Tree-Adjoining Grammar (LTAG) framework wherein supertag disambiguation provides a representation that is an *almost parse*. We have used the disambiguated supertag sequence in conjunction with a lightweight dependency analyzer to compute noun groups, verb groups, dependency linkages and even partial parses. We have shown that a trigram-based supertagger achieves an accuracy of 92.1‰ on Wall Street Journal (WSJ) texts. Furthermore, we have shown that the lightweight dependency analysis on the output of the supertagger identifies 83‰ of the dependency links accurately. We have exploited the representation of supertags with Explanation-Based Learning to improve parsing effciency. In this approach, parsing in limited domains can be modeled as a Finite-State Transduction. We have implemented such a system for the ATIS domain which improves parsing eciency by a factor of 15. We have used the supertagger in a variety of applications to provide lexical descriptions at an appropriate granularity. In an *information retrieval* application, we show that the supertag based system performs at higher levels of precision compared to a system based on part-of-speech tags. In an *information extraction* task, supertags are used in specifying extraction patterns. For *language modeling* applications, we view supertags as syntactically motivated class labels in a class-based language model. The distinction between recursive and non-recursive supertags is exploited in a *sentence simplification* application.

Comments

University of Pennsylvania Institute for Research in Cognitive Science Technical Report No. IRCS-97-10.

Institute for Research in Cognitive Science

Complexity of Lexical Descriptions and Its Relevance to Partial Parsing

Srinivas Bangalore

University of Pennsylvania
3401 Walnut Street, Suite 400A
Philadelphia, PA 19104-6228

June 1997

Site of the NSF Science and Technology Center for Research in Cognitive Science

IRCS Report 97--10

COMPLEXITY OF LEXICAL DESCRIPTIONS AND ITS RELEVANCE TO PARTIAL PARSING

SRINIVAS BANGALORE

A DISSERTATION

in

COMPUTER AND INFORMATION SCIENCE

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy.

1997

Aravind K. Joshi

Supervisor of Dissertation

Mark Steedman

Graduate Group Chairperson

Acknowledgments

I owe my thanks to a number of people, each of whom contributed in their own way towards this research and in the preparation of this document. First of all, I thank Prof. Aravind Joshi for his continued support during the period of this research. I have benefited significantly from his deep insights and his passion for subtle details which have made a significant impact on this research. I thank Prof. Mitch Marcus with whom I have had numerous "corridor conversations" that have helped shape this research. I deeply appreciate his support in the absence of Prof. Joshi, during Fall 1994. My thanks are also due to Prof. Mark Steedman whose seminar course helped me to extend the applicability of this research. I thank Steve Abney, Mark Liberman and John Trueswell, who as part of my committee provided insightful comments and suggestions that has improved the quality of this research.

Without Yves Schabes and his dissertation, this research would not have been possible at all. I have benefited a great deal from my discussions with him on many occasions. I have also used his code developed for XTAG, extensively in this research.

There would be no greater injustice if I do not acknowledge the influence the XTAG group has had on this research, besides keeping me on my toes at all times. Beth Ann Hockey and Christine Doran deserve my special thanks for putting up with my linguistic ignorance and cheering me along with my often silly ideas. Without their critical remarks, this research would have taken much longer than it has. I thank Dania and Martin for providing the many tools used in this research. I also would like to thank the XTAG elves – Heather, Laura, Susan and young Tim who cleaned up the XTAG corpus over the summer of 1995 and 1996.

My special thanks to R. Chandrasekar (Mickey) for several invigorating discussions

about my work, in particular and the field of computer science, in general. I also thank him for giving me a chance to apply supertagging to his work and for his warm fraternal affection that saw me through the final stages of this work. Significant improvement in supertagging performance came about during the Summer of 1996, thanks to the "summer camp" team – Breck Baldwin, Christine Doran, Michael Niv and Jeff Reynar. I am deeply indebted to them for providing constructive criticism and setting up high standards of performance.

My thanks to Martha Palmer, Tilman Becker, James Rogers for providing constructive comments during practice talks and in student-lounge discussions. I would also like to acknowledge Rajesh Bhatt, Eric Brill, Ted Briscoe, Mike Collins, Jason Eisner, Seth Kulick, Dan Melamed, Adwait Ratnaparkhi, Anoop Sarkar, David Yarowsky and the innumerable IRCS visitors for useful discussions.

I would like to thank the efforts of the administrative staff of CIS Department and IRCS, including Mike Felker, for dealing with my course issues; Gail Shannon, for many money matters; Trisha Yannuzzi and Chris Sandy for accomodating my wierd requests for various kinds of letters and certifications.

Finally, I owe everything I am to my parents. In particular, I thank them for their unflagging encouragement to pursue my research career and for their confidence in me to succeed. My special thanks to Purnima for standing by me at all times and giving me a reason to smile during the frustrating times of this research. I also acknowledge the support from my friends Bharathi, Esther, Raghava, Tikkana, and Vijay during the early stages of this research.

Abstract

Complexity of Lexical Descriptions and its Relevance to Partial Parsing

Srinivas Bangalore

Supervisor: Aravind K. Joshi

In this dissertation, we have proposed novel methods for robust parsing that integrate the flexibility of linguistically motivated lexical descriptions with the robustness of statistical techniques. Our thesis is that the computation of linguistic structure can be localized if lexical items are associated with rich descriptions (*supertags*) that impose complex constraints in a local context. However, increasing the complexity of descriptions makes the number of different descriptions for each lexical item much larger and hence increases the local ambiguity for a parser. This local ambiguity can be resolved by using supertag co-occurrence statistics collected from parsed corpora. We have explored these ideas in the context of Lexicalized Tree-Adjoining Grammar (LTAG) framework wherein supertag disambiguation provides a representation that is an *almost parse*. We have used the disambiguated supertag sequence in conjunction with a lightweight dependency analyzer to compute noun groups, verb groups, dependency linkages and even partial parses. We have shown that a trigram-based supertagger achieves an accuracy of 92.1% on Wall Street Journal (WSJ) texts. Furthermore, we have shown that the lightweight dependency analysis on the output of the supertagger identifies 83% of the dependency links accurately. We have exploited the representation of supertags with Explanation-Based Learning to improve parsing efficiency. In this approach, parsing in limited domains can be modeled as a Finite-State Transduction. We have implemented such a system for the ATIS domain which improves parsing efficiency by a factor of 15. We have used the supertagger in a

variety of applications to provide lexical descriptions at an appropriate granularity. In an *information retrieval* application, we show that the supertag based system performs at higher levels of precision compared to a system based on part-of-speech tags. In an *information extraction* task, supertags are used in specifying extraction patterns. For *language modeling* applications, we view supertags as syntactically motivated class labels in a class-based language model. The distinction between recursive and non-recursive supertags is exploited in a *sentence simplification* application.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Although parsers have proved uncontroversially useful in the domain of processing Programming Languages, the issue of parsing, in the domain of Natural Languages (NL) processing, has been a cause for tension between the computational and linguistic perspectives for a long time. In the past, the controversy about parsing was due to the divergence of objectives between natural language application developers who were oriented to developing practical parsers and psycholinguists who were concerned with the psychological process of language comprehension. In recent times, however, developers of natural language applications have questioned the usefulness for parsing in practical NLP systems. This is primarily because there are no grammars that have complete coverage of freely occurring natural language texts and there are no parsers that are robust enough to deal with that inadequacy. This limitation is further compounded by the fact that the inherent ambiguity of Natural Languages forces parsers to operate at speeds far from real-time requirements. In this dissertation, we address the issue of robust parsing by exploring various novel methods of using linguistically motivated, rich and complex lexical descriptions for partial parsing.

We review some of the issues faced by NL parsers in Section 1.1 and summarize two approaches to robust parsing in Section 1.2. In Section 1.3 we motivate our approach of using rich and complex lexical descriptions for partial parsing and introduce two methods pursued in this dissertation.

Traditionally, a *parser* is defined as a device that assigns a structural description to

an input string given a set of elementary structures (rules, trees or graphs) defined in a grammar and the operations for combining those structures.[1] The output of a parser is a *parse* that serves as a proof of the well-formedness of the input string given the elementary structures of the grammar. A parse is the result of searching, selecting and combining appropriate structures from the set of elementary structures defined in the grammar, so as to span the input string. The process of combining elementary structures is represented by the *derivation structure* for the input string. In the past few years, however, statistically-induced grammarless parsing methodology [Jelinek *et al.*, 1994; Magerman, 1995; Collins, 1996] has redefined the parsing enterprise. Although the task of parsing is still to assign a structural description to an input, it is no longer the case that the description be arrived by combining elementary structures of a grammar. In fact, there is, no notion of an explicit grammar in this paradigm of parsing. A structural description to a string is assigned based on the likelihood of that structure in a context similar to that in the string. The contexts and likelihoods are collected from a corpus of sentences that are annotated with the desired structural descriptions.

1.1 Issues in Natural Language Parsing

In order to situate our work, we briefly review the issues involved in Natural Language (NL) parsing. Since parsing is the computation of assigning an interpretation to a sentence via a grammar, NL parsing enterprise has been influenced by computational, linguistic and psycholinguistic considerations [Robinson, 1981]. In this section, we review the issues in NL parsing from computational, psycholinguistic and linguistic perspectives in the context of a purely grammar-based parsing approach. In the next few sections, we discuss the emerging approaches to NL parsing that address these issues to some extent.

1.1.1 Computational Issues

Natural Languages are both lexically and structurally ambiguous. This results in both local and global ambiguity for a natural language parser. This fact is further compounded

[1] Parsers for context-free grammars can be viewed this way with rules as one level trees that are combined using substitution operation.

by the high performance requirements on NL parsers in terms of speed,[2] accuracy and robustness which makes NLs hard to parse.

Robustness: The syntactic structure of natural language is very complex and no grammar has been written that has complete coverage of any natural language. This is further compounded by the fact that unrestricted texts contain ungrammatical and fragmented sentences. Natural language parsers are expected to be robust in the face of incomplete lexicon and grammar coverage, and even ungrammatical input. They are expected to rapidly parse simple sentences and yet degrade gracefully in the face of extragrammatical and ungrammatical input.

Speed: NL parsers that attempt to parse sentences with 25 to 30 words are often faced with a combinatorially intractable task due to ambiguity inherent in natural language descriptions. Parsers arrive at an analysis by systematically searching, selecting and combining appropriate elementary structures from the set of possible structures defined in the grammar, so as to span the input string. This aspect of searching through the space of elementary structures of the grammar to determine the appropriate structures to combine is the single most time-expensive component of the parsing process. The space of elementary structures to be searched and combined grows rapidly as the degree of ambiguity of words in the input string increases. This fact is further compounded in wide-coverage grammars where a variety of elementary structures are included in the grammar to deal with a range of linguistic constructions. Due to the enormous size of the search space, parsers for wide-coverage grammars are excruciatingly slow and often perform at speeds that make them impractical to use.

Selecting Analysis: Assuming that an NL parser is fast enough to find a set of analyses in a reasonable amount of time, it is faced with the problem of selecting a correct analysis. As a result of lexical and syntactic ambiguity in NLs, parsers invariably produce multiple parses for a given input. However, since the grammar itself does not directly encode any preference metric with each analysis, parsers for NLs in general do not have a method of choosing the correct analysis among competing analyses.

[2]We consider the speed of the parser by itself and not when the parser is embedded in an application and can have access to other sources of information such as semantic, pragmatic, world knowledge or preference factors.

Global Structure: The objective of a parser is to produce a structure that spans the entire input string. However, due to lack of preference metrics combined with the incompleteness of the lexicon and grammar, and possibly ungrammatical input, parsers, in their attempt to assign a global structure to the complete input, often produce disprefered local structures.

1.1.2 Linguistic Issues

It is quite evident that there is a close relationship between a parser and the representation the parser manipulates. However, in recent times there has been increasing debate on such issues as what should the representation be, how linguistically detailed should the representation be and how does one go about constructing such a representation.[3]

An active line of research has been to develop wide-coverage grammars in well-studied, mathematically rigorous grammar frameworks that are linguistically adequate. A parser based on a linguistically motivated wide-coverage grammar has many advantages. A wide-coverage grammar that is domain independent encodes the invariances of the language in general and is not specialized to any particular domain. As a result, such a grammar is more portable across domains than a grammar developed for a particular domain. However, for a parser to better exploit the idiosyncrasies of a domain, a wide-coverage grammar can be specialized to that domain; thus making parsing efficient in limited domains. Also, depending on how directly a grammar framework encodes linguistic facts, a linguistically motivated grammar developed in that framework could produce output that is quite detailed and directly amenable to further processing. Furthermore, if a grammar for one language is created in detail and the structures of the grammar are organized systematically then it is conceivable that grammars for closely related languages could be automatically generated by abstracting away from language specific features.

1.1.3 Psycholinguistic Issues

Humans parse natural language expressions rapidly, robustly, efficiently and effectively even in noisy environments with degraded inputs. This suggests that parsing mechanisms

[3]We are hesitant to limit the notion of representation to the traditional sense of grammar, since these issues are applicable to statistical "grammar-less" parsing paradigms as well.

that build on what we know of human sentence processing might do better in terms of speed and robustness.

Lexicalist Approach: Humans seem to bring in vast amounts of lexical information and rapidly commit to a single structure on the basis of probabilistic and local contextual information [Trueswell and Tanenhaus, 1994; MacDonald *et al.*, 1994]. In view of this, recent theories of sentence processing in psycholinguistics have emphasized the role of lexical mechanisms, a fact independently motivated in computational formalisms as well.

Local Constaints: An important aspect of human sentence processing is use of local constraints and rapid commitment to a single structure, even at the cost of processing errors. Not many parsers in the past have focussed on this issue, and instead attempted at a globally consistent parse while entertaining many possible structure in parallel instead of risking a total failure to parse. A primary reason for this distinction is that the cost of processing errors in humans is relatively low since they are endowed with quick error detection and error recovery mechanism that works in conjunction with the on-line, incremental processing mechanism. Recent trends in parsing can be viewed as addressing some of these issues.

1.2 Methods for Robust Parsing

In this section, we discuss some of the robust parsing methods that have been adopted to parse NLs that either address or circumvent the issues that were discussed in the previous section. The main emphasis of these techniques has been on speed and coverage of parsing while trading the depth of analysis to robustness. There are two main approaches for dealing with robust parsing of natural languages. We summarize the similarities and differences between these approaches.

1.2.1 Finite State Grammar-based Approaches

Finite State Grammar-based approaches to parsing is characterized by, for example, [Joshi, 1960; Joshi and Hopely, 1997; Abney, 1990a; Appelt *et al.*, 1993; Roche, 1993; Grishman, 1995] parsing systems. These systems use grammars that are represented as cascaded finite-state regular expression recognizers. The regular expressions are usually hand-crafted.

Each recognizer in the cascade provides a locally optimal output. The output of these systems is mostly in the form of noun groups and verb groups rather than constituent structure, often called as a *shallow parse*. There are no clause-level attachments or modifier attachments in the shallow parse. These parsers always produce one output, since they use the longest match heuristic to resolve cases of ambiguity when more than one regular expression matches the input string at a given position. At present none of these systems use any statistical information to resolve ambiguity. The grammar itself can be partitioned into domain independent and domain specific regular expressions which implies that porting to a new domain would involve rewriting the domain dependent expressions. This approach has proved to be quite successful as a preprocessor in information extraction systems [Hobbs *et al.*, 1995; Grishman, 1995].

1.2.2 Statistical Parsers

Pioneered by the IBM natural language group [Fujisaki *et al.*, 1989] and later pursued by, for example [Schabes *et al.*, 1993; Jelinek *et al.*, 1994; Magerman, 1995; Collins, 1996]; this approach decouples the issue of well-formedness of an input string from the problem of assigning a structure to it. These systems attempt to assign some structure to every input string. The rules to assign a structure to an input is extracted automatically from hand-annotated parses of large corpora, which are then subjected to smoothing to obtain reasonable coverage of the language. The resultant set of rules are not linguistically transparent and are not easily modifiable. Lexical and structural ambiguity is resolved using probability information that is encoded in the rules. This allows the system to assign the most-likely structure to each input. The output of these systems consists of constituent analysis, the degree of detail of which is dependent on the detail of annotation present in the treebank that is used to train the system. However, since the system attempts a globally optimal parse, it may produce locally disprefered analysis. Also, since there is no distinction between the domain dependent and domain independent rule sets, the system has to be retrained for each new domain, which in turn requires an annotated corpus for that domain.

There are also parsers that use probabilistic (weighting) information in conjunction

with hand-crafted grammars, for example, [Black *et al.*, 1993b; Nagao, 1994; Alshawi and Carter, 1994; Srinivas *et al.*, 1995]. In these cases the probabilistic information is primarily used to rank the parses produced by the parser and not so much for the purpose of robustness of the system.

1.3 Our Approach

In this dissertation, we have proposed novel methods for robust parsing that integrate the flexibility of linguistically motivated lexical descriptions with the robustness of statistical techniques. We pursue the idea that the computation of linguistic structure can be localized if lexical items are associated with rich descriptions (*Supertags*) that impose complex constraints in a local context. This makes the number of different descriptions for each lexical item much larger, than when the descriptions are less complex; thus increasing the local ambiguity for a parser. However, this local ambiguity can be resolved by using statistical distributions of supertag co-occurrences collected from a corpus of parses. Supertag disambiguation results in a representation that is effectively a parse (*almost parse*).

The idea of using complex descriptions for primitives to capture constraints locally has some precursors in AI. For example, the Waltz algorithm [Waltz, 1975] for labeling vertices of polygonal solid objects can be thought of in these terms. The idea of the algorithm is to include the various possibilities of labeling vertices as primitives thus localizing ambiguity. The labeling constraints associated with the primitives operate locally and are used to filter out mutually incompatible primitives, thus leaving only combinations of compatible ones as solutions. However, as far as we know, there is no rendering of Waltz's algorithm that exploits the locality of constraints using statistical techniques.

1.3.1 Localizing Ambiguity

In the linguistic context, there can be many ways of increasing the complexity of descriptions of lexical items. The idea is to associate lexical items with descriptions that allow for localization of all and only those elements on which the lexical item imposes constraints to be with in the same description. Further, it is required to associate each lexical item with as many descriptions as the number of different syntactic contexts in which the lexical item

can appear. This, of course, increases the local ambiguity for the parser. The parser has to decide which complex description out of the set of descriptions associated with each lexical item is to be used for a given reading of a sentence, even before combining the descriptions together. The obvious solution is to put the burden of this job entirely on the parser. The parser will eventually disambiguate all the descriptions and pick one per lexical item, for a given reading of the sentence. However, there is an alternate method of parsing which reduces the amount of disambiguation done by the parser. Just as in Waltz's algorithm, the idea is to locally check the constraints that are associated with the descriptions of lexical items to filter out incompatible descriptions. During this disambiguation, the system can also exploit statistical information that can be associated with the descriptions based on their distribution in a corpus of parses.

We employ these ideas in the context of LTAG where each lexical item is associated with one supertag for each syntactic context that the lexical item appears in. The number of supertags associated with each lexical item is much larger than the number of standard parts-of-speech (POS) associated with that item. Even when the POS ambiguity is removed the number of supertags associated with each item can be large. We show that disambiguating supertags prior to parsing speeds-up a parser by a factor of 30. More interesting, is the fact that since supertags combine both phrase structure information and dependency information in a single representation, a disambiguated supertag sequence in conjunction with a lightweight dependency analyzer can be used to compute noun groups, verb groups, dependency linkages and even partial parses. We have shown that training on one million supertag annotated words from Wall Street Journal corpus and testing on 47,000 words of held-out data, a trigram-based supertagger assigns 92.1% of the words the same supertag as they would be assigned in the correct parse of the sentences. Furthermore, we have shown that the lightweight dependency analysis on the output of the supertagger for the 47,000 words identifies 83% of the dependency links accurately.

1.3.2 Using Explanation-based Learning Technique

Although hand-crafted wide-coverage grammars are portable, they can be made more efficient if it is recognized that language in limited domains is usually well constrained and

certain linguistic constructions are more frequent than others. Hence, not all the structures of the domain-independent grammar nor all the combinations of those structures would be used in limited domains. In this paper, we view a domain-independent grammar as a repository of portable grammatical structures whose combinations are to be specialized for a given domain. We use Explanation-based Learning mechanism to identify the relevant subset of a hand-crafted general purpose grammar needed to parse in a given domain. The use of EBL for natural language parsing involves two phases – training phase and application phase. In the training phase, generalized parses of sentences are stored under a suitable index that is computed from the training sentences. In the application phase, an index for the test sentence is used to retrieve a generalized parse which is then instantiated to that sentence. This approach improves the efficiency of a parsing system by exploiting the domain specific sentence structures. The success of this approach depends heavily on efficacy to generalize parses. Rayner [1988], Samuelsson [1991] and Neumann [1994] have used the EBL methodology to specialize CFG-based grammars for improving the efficiency of their systems.

We show that the idea of increased complexity of lexical descriptions as instantiated in LTAGs allows us to represent certain generalizations that are not possible in CFG-based approaches. Our method exploits some of the key aspects of LTAGs, to (a) achieve an *immediate* generalization of parses for the training set of sentences (b) achieve generalization over recursive substructures of the parses in the training set and (c) allow for a finite state transducer (FST) representation of the set of generalized parses.

1.4 Chapter Summaries

The outline of the dissertation is as follows.

Chapter 2: In this chapter, we review the previous work in robust natural language parsing. The chapter contains reviews of three class of parsing methodologies that incorporate robustness. First, the finite-state based partial parsing systems employ simple mechanisms to group sequences of words and indicate the grammatical relations among word groups. Second, statistical parsing systems use annotated treebanks to induce probability distributions pertaining to parsing decisions and employ smoothing techniques to

achieve robustness. Third, parsing systems that extend conventional parsing techniques to achieve robustness.

Chapter 3: In this chapter, we introduce the LTAG formalism and review the three central properties of LTAGs: Lexicalization, Extended Domain of Locality and Factoring of Recursion from the domain of dependencies. We discuss the relevance of these properties for partial parsing and present an overview of a grammar development environment and a wide-coverage grammar for English that is being developed in LTAG.

Chapter 4: In this chapter, we present the idea of supertag disambiguation and discuss its relationship to partial parsing. We present several models for supertag disambiguation and their performance evaluation results on various corpora. We also present a novel lightweight dependency analyzer that exploits the information encoded in the supertags to compute a dependency analysis of a sentence.

Chapter 5: Although hand-crafted wide coverage grammars are portable, they can be made more efficient when applied to limited domains, if it is recognized that language in limited domains is usually well constrained and certain linguistic constructions are more frequent than others. In this chapter, we present the Explanation-based Learning technique of specializing a wide-coverage LTAG grammar to a limited domain. We show that by exploiting the central properties of LTAGs, parsing in limited domains can be seen as Finite State Transduction from strings to their dependency structures.

Chapter 6: In this chapter, we present an impoverished parser called *Stapler* that takes as input the *almost parse* structure resulting from an FST induced by the EBL mechanism and computes all possible parses and instantiates it to the particular sentence. Using this technique, the specialized grammar obtains a speed up of a factor of 15 over the unspecialized grammar on the Air Travel Information Service (ATIS) domain.

Chapter 7: This chapter addresses the issue of evaluating partial parsing systems. It consists of two parts. In the first part, we review the currently prevalent evaluation metrics for parsers and indicate their limitations for comparing parsers that produce different representations. We propose an alternate evaluation metric based on dependency relations that overcomes the limitations of the existing evaluation metrics. In the second part of this chapter, we provide results from extensive evaluation of the supertagger and the LDA system on a range of tasks and compare its performance against the performance of other

systems on those tasks.

Chapter 8: In this chapter, we discuss several applications of the supertagger and the LDA system. Supertagging has been used in a variety of applications including information retrieval and information extraction, text simplification and language modeling. Our goal in using a supertagger is to exploit the strengths of supertags as the appropriate level of lexical description needed for most applications. In an *information retrieval* application, we compare the retrieval efficiency of a system based on part-of-speech tags against a system based on supertags and show that the supertag-based system performs at higher levels of precision. In an *information extraction* task, supertags are used in specifying extraction patterns. For *language modeling* applications, we view supertags as syntactically motivated class labels in a class-based language model. The distinction between recursive and non-recursive supertags is exploited in *a sentence simplification application.* The ability to bring to bear document-level and domain-specific constraints during supertagging of a sentence is explored in another application.

Chapter 2

Literature Survey

In recent times, robust Natural Language(NL) parsing has received renewed attention. This interest in robust parsing has been driven by the intuition that exploiting sophisticated linguistic information present in NL texts should improve the performance of NL-based applications. However, although there has been significant progress in both grammar-based and stochastic robust parsing, general purpose parsers are far from being readily deployable in practical applications.

Robustness in parsing refers to that property of the parser that irrespective of the grammaticality of the input string allows the parser to produce a parse that may or may not span the entire input string. The stumbling blocks for grammar-based parsing techniques from achieving robustness are: incomplete lexicons, inadequate grammatical coverage of systems, extremely long sentences that take unreasonable amount of time to complete a parse, and ungrammatical texts. The stochastic parsers, although more robust than purely grammar-based approaches, have their share of limitations. Being heavily data driven, stochastic approaches need vast amounts of data which need to be annotated; a fairly expensive proposition. Further, the richness of the output produced by these parsers is limited by the detail of the annotation.

In this chapter, we will review some of the previous approaches that have resorted to partial parsing in an attempt to achieve robustness in parsing. By partial parsing, we mean that these systems either always or on failure to provide a single tree spanning the input, produce a forest of trees that span most of the input. The reviews in this chapter

discuss the methodologies used by the various partial parsers without providing much description of their performance. However, cross comparison of the performance of these parsers, which would be invaluable, is extremely difficult to achieve for obvious reasons. We also review a few data-driven, stochastic-based parsers that use smoothing techniques to achieve robustness.

The particular parsers being reviewed and the main reasons for including them in this particular set are listed below:

- Type I : Extended Finite State Transducer-based Partial Parsers

  This approach to parsing use grammars that are represented as cascaded finite-state regular expression recognizers. The regular expressions are usually hand-crafted. Each recognizer in the cascade provides a locally optimal output. The output of these systems is mostly in the form of noun groups and verb groups rather than constituent structure, often called as a *shallow parse* or a *text chunk*. There are no clause-level attachments or modifier attachments in the shallow parse. These parsers always produce one output, since they use the longest match heuristic to resolve cases of ambiguity when more than one regular expression matches the input string at a given position. At present none of these systems use any statistical information to resolve ambiguity. The grammar itself can be partitioned into domain independent and domain specific regular expressions which implies that porting to a new domain would involve rewriting the domain dependent expressions. This approach has proved to be quite successful as a preprocessor in information extraction systems

  - Fidditch [Hindle, 1983]: Industrial strength version of Marcus' parser [Marcus, 1983] that is used extensively in the Treebank project at the University of Pennsylvania.
  - CASS [Abney, 1990b]: A multi-stage parser that employs simple and computationally inexpensive set of tools at each stage to arrive at a parse.
  - UNIVAC-1 Parser (now called Uniparse) [Joshi, 1960; Joshi and Hopely, 1997][1]:

[1]This paper describes the parser designed and implemented on UNIVAC-1 during the period 1958-59. This project was directed by Zellig Harris. The members of the project were Carol Chomsky, Lila Gleitman, Aravind Joshi, Bruria Kauffman and Naomi Sager. See also [Harris, 1962]. This parser has been

A parser of historical significance that employed many of the so-called current state-of-the-art techniques in parsing.

- Helsinki Parser [Karlsson *et al.*, 1994]: A parser (more of a tagger) that uses finite-state expressible constraints to reduce the ambiguity of morphosyntactic tags.

- Type II: Stochastic Parsers

  This approach decouples the issue of well-formedness of an input string from the problem of assigning a structure to it. These systems attempt to assign some structure to every input string. The rules to assign a structure to an input is extracted automatically from hand-annotated parses of large corpora, which are then subjected to smoothing to obtain reasonable coverage of the language. The resultant set of rules are not linguistically transparent and are not easily modifiable. Lexical and structural ambiguity is resolved using probability information that is encoded in the rules. This allows the system to assign the most-likely structure to each input. The output of these systems consists of constituent analysis, the degree of detail of which is dependent on the detail of annotation present in the treebank that is used to train the system. However, since the system attempts a globally optimal parse, it may produce locally disprefered analysis. Also, since there is no distinction between the domain dependent and domain independent rule sets, the system has to be retrained for each new domain, which in turn requires an annotated corpus for that domain. Some of the parsers that belong to this class are:

  - Decision Tree Parsers: This technique was first introduced in [Jelinek *et al.*, 1994] and was further developed in [Magerman, 1995]. In this parsing paradigm, parsing decisions are guided by the knowledge implicitly encoded in the probability distributions of a set of features. The probability distributions are estimated from a training corpus of parses using statistical decision tree models.

  - Bilder[Collins, 1996]: Bilder is a bigram lexical dependency based statistical parser that employs lexical information directly to model dependency relations

reconstructed from the original complete documentation (Papers # 15 – # 19 of the Transformation and Discourse Analysis Project (TDAP)) by Phil Hopely and Aravind Joshi.

between pairs of words. The statistical model is far simpler than the statistical decision tree parsing model but yet has been shown to outperform decision tree based parser.

- Data Oriented Parsing: Data Oriented Parsing (DOP) suggested by Scha [Scha, 1990] and developed in Bod ([Bod, 1995]) is a probabilistic parsing method that views parsing as a process of combining tree fragments. Each tree fragment is associated with a probability estimate based on its frequency in the training corpus. The frequency estimates associated with the tree fragments are used to rank order derivation steps and to retrieve the most probable parse tree.

- Type III: Extended Chart Parsers:

  This class of robust parsers are a result of augmenting the chart parsing technique. Most of these parsers use a hand crafted grammar, usually specific to a domain, written in context-free grammar formalism. Robustness in these parsers is usually achieved by using techniques for retrieving partial constituents from the chart even if the parser fails to retrieve a complete parse. However, as discussed below, a notable exception to this approach is [Black *et al.*, 1993b].

  - De Marcken's Parser [Marcken, 1990]: Agenda-based parser with heuristics to order the agenda. This parser is being used as a front-end to an information extraction system, PLUM [Weischedel *et al.*, 1992].

  - Seneff's Extended Chart Parser [Seneff, 1992]: A chart parser that has been extended to permit the recovery of parsable phrases and clauses within a sentence. This parser is presently being used in conjunction with the MIT Speech Recognition System in the ATIS domain.

  - The PLNLP System [Jensen *et al.*, 1993]: An integrated, incremental system for broad-coverage syntactic and semantic analysis and synthesis of natural language.

  - Sparser [McDonald, 1993], TACITUS [Hobbs *et al.*, 1991] and FASTUS [Appelt *et al.*, 1993]: Multi-stage domain-specific partial parsers that have been

employed with good success in information extraction tasks aimed at template-filling.

- IBM's P-CFG and HBG model [Black *et al.*, 1993b]: Hand-crafted context-free grammar that achieved robustness using probabilities obtained from a parsed corpus.

The partial parsers that are reviewed here have been intrinsically designed to provide partial output either always or on failure to provide a complete parse. However, there are other parsers that are not included here that may provide some partial parses but do so as a byproduct rather than being intrinsically designed that way.

2.1 Fidditch

The Fidditch parser[Hindle, 1983] is a large-scale version of Mitch Marcus' parser, PARSIFAL[Marcus, 1983]. PARSIFAL is based on the "determinism hypothesis" that claims that natural language can be parsed by a computationally simple mechanism that does not involve non-determinism and in which all grammatical structure created by the parser is "indelible" in that it must all be output as part of the structural analysis of the parser's input. Once built, no grammatical structure may be altered or discarded in the course of the parsing process.

The design goal of the Fidditch parser was to process transcripts of spontaneous speech and produce labeled bracketed trees, partial if necessary, that represented the syntactic structure of the input. Since the parser is based on the determinism hypothesis, it produces a single analysis for each input.

The input to the parser is a sequence of words, each with one or more lexical categories selected from a 90,000 word lexicon and morphological analyzer. The lexicon, for each word, contains the lexical category, the subcategorization frame, and in some cases, information on compound words. Once the lexical analysis is complete, the phrase structure is constructed using pattern-action rules.

The parser maintains two data structures – the stack and the constituent buffer. The stack is used to keep track of the incomplete nodes and the constituent buffer is used

to keep track of the completed constituents. The size of the constituent buffer is fixed to three, which allows the parser to look through a window of three constituents before selecting a grammar rule to apply. The grammar for the parser is a set of pattern action rules. Each pattern in a rule is matched against some subset of the constituent buffer and the top node of the stack. The actions of the parser include

- Create: build a new node by pushing a category onto the stack.
- Attach: attach the first element of the constituent buffer to the element on the top of stack.
- Drop: dropping the completed constituent from the stack into the first position of the buffer.
- Switch: swap the contents of the first and the second elements of the buffer. This move is to accommodate subject-aux inversion.
- Insert: add a new element into the buffer at some position and shift the contents one cell to the right. This move is to accommodate an empty category.
- Attention-shift: shift the attention from the first constituent in the buffer to some later constituent. This move is to predict a leading edge of Noun Phrases.
- Punt: If attachment is not possible, then move on to the next constituent.

The part-of-speech disambiguation in Fidditch is done during the parse using certain context sensitive rules. The parser does not attach most modifiers, adjuncts and relative clauses. Fidditch is the fastest partial parser, according to Abney [1994b].

2.2 CASS

CASS (Cascaded Analysis of Syntactic Structure) [Abney, 1990b] is a multi-stage parser that uses simple and inexpensive analyzers at each stage of the parsing process. The input sentence is tagged for POS using a trigram tagger. A stochastic NP-recognizer [Church, 1988] is used to identify non-recursive noun phrases in the input. The output after applying the above stages on the sentence 2.1 is shown in 2.2.

(2.1) In South Australia beds of boulders were deposited by melting icebergs in a gulf that marked the position of the Adelaide geosyncline, an elongated, sediment-filled depression in the crust.

(2.2) EOS In$_p$ [South$_{pn}$ Australia$_{pn}$ beds$_{npl}$] of$_p$ [boulders$_{npl}$] were$_{bed}$ deposited$_{vbn}$ by$_p$ [melting$_{vbg}$ icebergs$_{npl}$] in$_p$[a$_{det}$ gulf$_n$] [that$_{wps}$] marked$_{vbd}$ [the$_{det}$ position$_n$] of$_p$ [the$_{det}$ Adelaide$_{pnoun}$ geosyncline$_n$], [an$_{det}$ elongated$_{adj}$], [sediment-filled$_{adj}$ depression$_n$] in$_p$ [the$_{det}$ crust$_n$]. EOS

- Chunker: The chunker consists of a NP corrector and a chunk recognizer. The NP corrector fixes the errors made by the stochastic NP recognizer such as NP fragmentation resulting from conjunction of prenominal adjectives or modification of adjectives/cardinals by adverbs and qualifiers. This stage uses a regular expression recognizer to assemble noun phrases.

  The chunk recognizer also uses a small regular expression grammar to identify chunks. These chunks usually include the NPs recognized in the previous stages. The output after this stage on the input sentence is shown in 2.3.

  (2.3) EOS [$_{PP}$ In [$_{NP}$ South Australia beds]] [$_{PP}$ of [$_{NP}$ boulders]] [$_{VP}$ were deposited] [$_{PP}$ by [$_{NP}$ melting icebergs]] [$_{PP}$ in [$_{NP}$ a gulf]] [$_{WHNP}$ that] [$_{VP}$ marked] [$_{NP}$ the position] [$_{PP}$ of [$_{NP}$ the Adelaide geosyncline]], [$_{NP}$ an elongated, sediment-filled depression] [$_{PP}$ in [$_{NP}$ the crust]]. EOS

- Clause Recognizer: The clause recognizer consists of two modules: the simplex clause recognizer and the clause repair module.

  - Simplex Clauses: The simplex clause recognizer attempts to identify the beginning and end of each simplex clause and marks the subject and predicate of the clause. This is done using regular expressions that leverages off of clause markers such as complementizers, conjunctions and periods. If no subject and predicate can be identified, the type of error encountered is classified as either NO-SUBJ (no subject) or EXTRA-VP (too many VPs).

– Clause Repair: The parser attempts to repair NO-SUBJ and EXTRA-VP errors using predefined templates for each type of error. NO-SUBJ errors are caused due to unrecognized partitives, run-on NPs and complementizers mistaken to be prepositions. EXTRA-VP errors are caused due to empty relative pronouns, empty complementizer and misanalyzed complementizers. If the error type does not match any of the repair templates, then any NP followed by VP is regarded as a clause and a free standing VP a predicate.

Using the information in this module, the example in 2.3 is correctly chunked as follows:

(2.4) EOS [$_{PP}$ In [$_{NP}$ South Australia]] [$_{SUBJ}$ [$_{NP}$ beds]] [$_{PP}$ of [$_{NP}$ boulders]] [$_{VP}$ were deposited] [$_{PP}$ by [$_{NP}$ melting icebergs]] [$_{PP}$ in [$_{NP}$ a gulf]] [$_{WHNP}$ that] [$_{VP}$ marked] [$_{NP}$ the position] [$_{PP}$ of [$_{NP}$ the Adelaide geosyncline]], [$_{NP}$ an elongated, sediment-filled depression] [$_{PP}$ in [$_{NP}$ the crust]]. EOS

- Attachment: This stage, unlike the previous stages, is not a regular expression recognizer. It assembles recursive structures by attaching nodes one to another based on information about the arguments and modifiers licensed by the head words. At each point, the parser considers attaching the current chunk to either the preceding chunk or to the most recent chunk with a verb. There are also templates to handle constructions that involve conjunctions and sentence initial adjuncts. The possible actions are ordered by heuristics and the most appropriate action is applied. If no action is applicable, then the attachment site of the chunk is left indeterminate.

The parser has been evaluated for speed and accuracy of chunking. When used with the part-of-speech tagger the CASS, parser can parse a million word corpus in under six hours with a chunk segmentation accuracy of 95%. The design architecture of this parser is to make local decisions within each level and to make repairs, if any, downstream. Each stage outputs a single best answer. This way, the problem of exponential global ambiguity is overcome by making a sequence of independent small sets of choices. However, if the stages were to produce N-best output, the repair modules may not be required.

2.3 UNIVAC-1 Parser (now called Uniparse)

One of the foremost attempts at parsing language on a large-scale was done in the Discourse Analysis Project [Joshi, 1960; Joshi and Hopely, 1997; Harris, 1962] in the Department of Linguistics at the University of Pennsylvania. This parser was designed and implemented on UNIVAC-1 during the period 1958-59. This project was directed by Zellig Harris and included Carol Chomsky, Lila Gleitman, Aravind Joshi, Bruria Kauffman and Naomi Sager as members of the project. This parser, now called Uniparse [Joshi and Hopely, 1997], has been reconstructed from the original complete documentation (Papers # 15 – # 19 of the Transformation and Discourse Analysis Project (TDAP)) by Phil Hopely and Aravind Joshi. This parser is not only historically important but it also incorporated what are currently regarded as the state-of-the-art techniques for parsing large-scale texts.

The parser was designed as a multi-stage system. It is a cascade of Finite State Transducers (FST), except for the last stage, which technically is not an FST, but more like a Push Down Transducer (PDT). Each word in the input string is "tagged" with the class or classes to which the word belongs. If a word is tagged with multiple classes, then a series of tests are applied to identify environments in which a class definitely cannot hold. However, these negative tests may still leave the class ambiguity unresolved.

The input string is then segmented into *first order* strings based on the class marks associated with the words of the input. First order strings are typically noun phrases, sequence of verbs and adjunct phrase and are only minimal structures that do not include any nesting of other first order strings. Once the first-order strings are identified, their internal structure is not analyzed. The first-order strings behave like chunks in which there is a principal word and all the other words bear relation to this word. The domain of their relationship is "local" and does not extend beyond the substring concerned. The first-order strings can be recognized by a finite state computation since there is no nesting of first-order strings.

The input is scanned either left to right or right to left depending on the type of first-order string being recognized. Elementary noun sequences, adjunct phrases and verb sequences are recognized in that order. The chunks recognized in one scan are treated as frozen for the subsequent scans. Examples with the first-order strings marked off is shown

below. [ ] indicate noun phrases, { } indicate verb sequences and ( ) indicate adjuncts.

(2.5) [Those papers] {may have been published} (in [a hurry]).

(2.6) [I] {may (soon) go}.

Second order substrings are clausal in nature. Second order substrings are fixed sequences of first-order strings and can include other second order substrings. Recognition of a second order string begins with a left to right scan on the input in which the first-order substrings have been replaced by single characters. The input contains the second order heads that are not part of any of the first order strings (for example, complementizers and conjunctions or sentential subjects). Recognition of these second-order strings is done using a push down transducer like automaton. The well-formedness of the sentence is determined if the subcategorization requirements of the verb are satisfied. The following examples illustrate the clausal annotation. The markers < and > include a clause, *that*-clauses, sentential subjects and objects are included in between / and \ and + marks the end of a complement.

(2.7) [Those] <who read [newspapers] > {waste} [their time].

(2.8) (Under [conditions]) (of [dual induction]) , / decreasing: G [the amino - acid concentration]: + \ : N (from [0.2]) (to [0.1 per cent]) {suppressed}: W [beta - galactosidase synthesis]: + (to [a greater extent]) than [nitrate reductase]

At places where there are multiple possibilities in each stage, one choice is pursued but the alternatives are kept track of. The search is similar to a chart-based, depth-first preference-driven search with the possibility of backtracking.

2.4 FASTUS

The FASTUS system [Appelt *et al.*, 1993] was developed in the context of the Fourth Message Understanding Conference (MUC) in 1992 as a successor to the TACITUS system. The main motivation for developing FASTUS was that the TACITUS system was extremely slow (taking 36hours to process 100 messages) in parsing the text in the MUC messages.

The FASTUS system processed the same set of 100 messages in 12 minutes. The crucial difference between FASTUS over TASITUS is that FASTUS used a Finite State Mechanism to extract partial parses instead of a full parser.

Processing in FASTUS is driven by pattern matching. Patterns are used for name recognition, to determine the relevance of the sentence for the task at hand, and for syntactic and semantic processing. Each pattern is associated with at least one trigger word. The lexicon used for syntactic recognition contains 20000 lexical items with a total of 43000 morphologically inflected forms. The syntactic component consists of non-deterministic finite-state automata that recognize noun groups and verb groups. The noun group recognizer has 37 states and recognizes noun phrases, each with a head noun and left modifiers and determiners. The verb group recognizer has 18 states and includes the verb, the auxiliary and any intervening adverbs. The noun and verb groups are used to recognize patterns and output an incident structure. A series of pattern-incident structure pairs have been written for the MUC task. There are patterns that skip over relative clauses and preposition phrases on nouns. The incident structures are merged with other incident structures found in the same sentence. Merging is blocked if the types of the incident structures are incompatible.

The success of this approach lies in identifying rules for chunking and associating the interpretations to chunks. It would be interesting if these rules could be derived automatically for some domain given some initial training material in that domain.

2.5 ENGCG Parser

The ENGCG grammar representation [Karlsson *et al.*, 1994; Anttila, 1994] imposes a shallow analysis based on surface order resulting in a syntactic functional description for each word of the input string. The grammatical structure is encoded in terms of tags associated with each word rather than in terms of phrase structure bracketings. The tags incorporate morphological information and partial-dependency relations that encode information about the syntactic function of the word the tag is associated with. These tags leave a number of decisions about the dependency relations ambiguous.

The parsing grammar in this framework is a set of unordered rules that specify negative

constraints based on the linear order of words and tags to eliminate alternate grammatical tags for a word. The rules are hand-crafted based on observations on large samples of authentic texts.

The parsing process involves a lookup stage where each word is assigned all possible grammatical tags. The morphological descriptions are based on Koskeniemi's Two-level Model [Voutilainen, 1983]. There is a rule-based heuristic component that assigns grammatical tags to unknown words. Next, morphological disambiguation rules, numbering up to 1100 constraints, are used to filter out the result of the morphological lookup. The 1100 constraints use 4000 contexts of which some 400 refer to unbounded contexts. It is claimed that 93%-97% of all words are fully unambiguous with a maximum error rate of 0.3% [Voutilainen, 1994].

In the syntactic filtering stage the impossible and very unlikely (heuristically determined) grammatical tags for a word are discarded based on constraints specified in the grammar. The grammar consists of 260 syntactic disambiguation constraints and about 10 heuristic constraints. The result of this reductionist process is the grammatical analysis of the input. The syntactic disambiguation component is claimed to achieve an recall of 96-97% with about 75-85% of words being unambiguous[Voutilainen, 1994].

Unlike other parsing methods, there is no structure built during parsing. Parsing in this framework reduces to elimination of tags. The output representation is extremely shallow and quite ambiguous. Hence evaluation of such a system should involve both accuracy and precision. It is claimed that this approach outperforms other contemporary rule-based and stochastic part-of-speech taggers.

2.6 Decision Tree Parsers

Decision Tree Parsing was first introduced in [Jelinek *et al.*, 1994] and was further developed in [Magerman, 1995]. Parsing in this framework, as in other frameworks, is viewed as a sequence of decisions such as choosing the part-of-speech of a word, choosing the constituent structure and choosing the label of the constituent. However, the distinguishing factor of this framework is that the decision making process is not guided by an explicit hand-crafted knowledge base, such as a grammar or a lexicon, but instead, is guided by the knowledge

implicitly encoded in the probability distributions of a set of features. The probability distributions are estimated from a training corpus of parses using statistical decision tree models. A decision tree is a data structure wherein each internal node corresponds to a decision question and each leaf node corresponds to a choice. A statistical decision tree is a decision tree which assigns a probability to each of the possible choices conditioned on the context of the decision.

The preterminal nodes in a parse tree are labeled by part-of-speech labels that are predicted by a part-of-speech tagging model based on the two words to the left, the two words to the right and the two nodes to the left and two nodes to the right of the current word. The internal nodes in the parse tree are labeled by a constituent labeling model based on specific words being present in the constituent, two nodes to the left and two nodes to the right and two leftmost children and two rightmost children of the current constituent.

A parse tree is viewed as a pattern wherein a node in a tree represents a meeting point of a set of edges extending from the children of the node. An edge from any given node can extend either to the right, up or left. The direction of extension of a node is predicted by the extension model based on the two nodes to the left, two nodes to the right and the two leftmost and the two rightmost children of the current node. The node to be extended is predicted by a derivation model based on the current node information and the node information from a five node window.

This model of parsing has been used to parse IBM computer manuals [Jelinek *et al.*, 1994] and its performance was compared to a hand-crafted P-CFG grammar based parser whose probabilities were fine tuned to the computer manual domain. The hand-crafted grammar based parser, on the crossing bracket test over 1100 sentences, performed at 69% accuracy, while the decision tree parsing model, on the same task, performed at 78% accuracy.

In [Magerman, 1995], the same model has been trained and tested against the Penn Treebank on the Wall Street Journal corpus. The parser could retrieve 84% of the constituents that were found in the treebank and the parses for 56% of the sentences had no crossing brackets when compared against the treebank.

2.7 Bilder

Bilder [Collins, 1996] is a bigram lexical dependency based statistical parser that employs lexical information directly to model dependency relations between pairs of words. The statistical model is far simpler than the statistical decision tree parsing model but yet has been shown to outperform decision tree based parser. Bilder overcomes the shortcoming of the decision tree based parser by depending heavily on lexical items for each decision in the parse tree construction.

The parser takes as input, a part-of-speech tagged sentence and produces the most likely parse for the sentence. A part-of-speech tagger based on maximum entropy principle [Ratnaparkhi, 1996] is used as a preprocessor to tag a sentence. The input sentence is reduced by removing punctuation and replacing non-recursive noun phrases with their head-words alone. The lexical dependencies are computed using the Penn Treebank parses by mapping the constituent structure to a dependency structure. This is done by manually identifying which of the children of each constituent serves as the 'head-child' of that constituent. The head word of a constituent is the head word of its head child. A dependency relation between pairs of words is represented as a triple, the constituent label for each of the words and the label of the parent constituent. The objective of parsing is to find the most probable set of dependencies that collectively include all the words of the sentence. The problems of sparseness of data is handled by a back-off model that successively backs off from the pair of words to the part-of-speech of the pair of words.

This parsing model has been trained on Penn Treebank parses of 40,000 sentences from Wall Street Journal text. It performs at a labeled recall accuracy of 85.3% with 57.2% of the test sentences parsed with zero crossing brackets. The parser uses 180 MB of memory and discounting the time for POS tagging, the parser parses at a speed of 200 sentences/min.

2.8 Data Oriented Parser

Data Oriented Parsing (DOP) suggested by Scha [Scha, 1990] and developed in Bod ([Bod, 1995]) is a probabilistic parsing method that views parsing as a process of combining tree

fragments. The set of tree fragments is the set of all possible subtrees of parses in the training corpus. Each tree fragment is associated with a probability estimate based on its frequency in the training corpus. The tree fragments are combined using substitution operation. Substituting one tree fragment into the leftmost node of another tree fragment constitutes as one derivation step in the parsing process. The frequency estimates associated with the tree fragments are used to rank order derivation steps. Since a parse tree can be generated by many derivations involving different tree fragments, the probability of a parse tree is the sum of the probabilities of all possible derivations for that parse tree. The objective of the parser is to produce the most probable parse tree that spans the given sequence of words.

The DOP model has been used to parse the ATIS corpus, using the parses found in the Penn Treebank as the training corpus. Recently [Bod, 1995], the DOP model of parsing has been extended such that the tree fragments are associated with frequency estimates obtained from Good-Turing smoothing. Further, a dictionary is used to assign the part-of-speech categories to the words of test sentences. This extended model performs at a sentence accuracy[2] of 68% and a bracketing accuracy[3] of 93.2%.

2.9 De Marcken's Parser

De Marcken's parser [Marcken, 1990] like other robust parsers, attempts to parse simple structures rapidly while "gracefully failing" on more complicated constructions. The text chosen to parse is the LOB corpus. An N-best POS tagger is used as a front-end to the parser.

The parser uses an agenda mechanism. It maintains three tables.

1. Rule-Action Table: Specifies the action to be taken by a rule in a state that encounters a phrase with a given category and features.

2. Single-phrase-action table: Specifies if a phrase with a category and features should project or start a rule.

[2]Percentage of sentences with no crossing brackets with respect to the treebank parses
[3]Percentage of brackets of the most probable parse that do not cross the brackets in the treebank parses

3. Phrase-phrase action table: Specifies the action to be taken when two phrases abut each other.

The possible actions that the parser can take are:

1. Shift the dot: Accepting a phrase, by shifting the dot in a rule.

2. Closing: Reducing a set of phrases recognized so far to a single phrase based on a rule.

3. Lowering: Lowering is a mechanism by which, once a rule closes to a phrase, all the phrases constituting the phrase are lowered to the bottom of the agenda. Lowering helps in determinism.

An additional transducer mechanism is used to produce the tree relations for the output. Rules are given precedence over phrases in the agenda. The usual mechanism of retrieving the longest phrase starting at a place is used. This mechanism may fail to retrieve the best possible phrasal parse. The output is a list of phrasal parses, with not much attachment information. The parser commits systematic errors in distinguishing adjuncts from arguments.

2.10 Seneff's Extended Chart Parser

Seneff [1992] extends a standard chart parser [Seneff, 1992] to permit the recovery of parsable phrases and clauses within sentences. This parser was used in conjunction with a spoken language system for ATIS domain and was shown to provide a considerable improvement in the performance of the spoken language system.

The parser is designed to operate in two modes – full-sentence mode and relaxed mode. In the full-sentence mode, the parser attempts to find an analysis for the complete sentence. The parser succeeds only if it finds a sentence category that spans the entire input. In the relaxed mode, the parser proceeds left-to-right producing all parses for the string starting at the first word. The parse that spans the most number of words is selected. The parser then attempts to parse the input starting at the first subsequent word. The parser skips a word if it does not find a parse starting at that word. The selected parse after each

iteration is converted into a semantic frame based on domain knowledge. These semantic frames are glued together using discourse information. This approach to robust parsing is effective in limited domains. However, it will not be practical with a domain-independent wide-coverage grammar.

2.11 The PLNLP approach

Programming Language for Natural Language Processing (PLNLP) system [Jensen *et al.*, 1993] is an integrated, incremental system for broad-coverage syntactic and semantic analysis and synthesis of natural language. This system has been used to develop a robust computational broad-coverage grammar, the PLNLP English Grammar (PEG).

The PEG syntactic grammar is an augmented phrase structure grammar that has a feature-structure like constraint mechanism. It has 200 rules specified in the PLNLP specification language. The grammar is supported by a 100,000 entry lexicon that includes part-of-speech and subcategorization information. The output of the PEG parser is a phrase structure with each level annotated with the heads and modifiers. The grammar and the parser have been used in the context of a grammar checker and hence by definition have to handle ill-formed text.

The parser is a bottom-up chart parser that attempts to cover the input with the category for a sentence. The parser uses a preference metric so as to produce a ranked set of parses. On failing, a *parse fitting* process is initiated in which the constituents from the chart are retrieved and fitted together to form the *fitted parse* for the sentence. The parse fitting proceeds by selecting a head constituent according to a preference list and then fitting the remaining constituents into it. The fitting process is complete if the head covers the entire input. If the head constituent does not cover the entire string, then the remaining constituents are added on either side. The fitting process selects the largest constituent at each iteration. The net result of the parse fitting is a fitted parse that consists of the largest chunk of the sentence as a constituent and the remaining chunks attached to it in some reasonable manner.

2.12 TACITUS

The design philosphy of TACITUS [Hobbs *et al.*, 1991] has been to cater to detailed and thorough analysis of natural language text. However, in order to maintain adequate level of performance in terms of efficiency and robustness, application-specific components (eg. a Hispanic Name detector) that do not violate the overall design philosphy have been added to TACITUS. There are modules for handling unknown words, and a relevance filter that decides the relevance of a sentence for the task at hand even before parsing it, using n-gram information about words that appear in the domain.

The syntactic analysis component of TACITUS uses a broad coverage grammar derived from the String and the DIAGRAM grammars.[4] The output analyses are ordered using heuristics encoded in the grammar. The output of this component is translated into a predicate-argument relation and subordination relation.

The syntactic analysis component consists of an agenda-based scheduling parser, a recovery heuristic for unparsable sentences and a terminal string parser for very long sentences. The parser works bottom-up using an agenda to order the nodes and edges by their likelihood of participating in a correct parse. This allows the parser to pursue the most likely paths in the search space without trying out all possibilities. The nodes and edges are ranked based on parameters such as the number of words spanned, whether the constituent is complete or not, and certain preference heuristics. The authors mention that the preference heuristics proved to be the most effective of all. These preference heuristics are the ones discussed in [Hobbs and Bear, 1994]. A majority of the errors made by the parser were due to a few nagging problems, one such being topicalization analysis for sentences that have two sentence initial nouns.

On a parse failure, the parser retrieves the best sequence of fragments that spans the longest part of the sentence. Clauses, verb phrases, adverbial phrases and noun phrases are the fragments retrieved in that order. It was observed that in many cases the lost words, i.e. the words that did not feature in any of the fragment sequences did not add much to the propositional content of the sentence. There were on the average two fragments for each of the sentences that failed to parse.

[4]The String Grammar is a descendant of the grammar used in the UNIVAC-1 parser.

The terminal substring parser component works on sentences longer than 60 words. The sentence is segmented into smaller substrings by breaking at commas, conjunctions, relative pronouns and certain instances of the word "that". The substrings are then parsed, starting from the right and working back as one of the categories – main, subordinate and relative clauses, infinitives, verb phrases, prepositional phrases and noun phrases. At each substring, the parser also attempts to combine the set of substrings collected so far into one of the above categories. The best structure is then selected for subsequent processing.

(2.9) George Bush, the President, held a press conference yesterday.

This sentence is broken into three fragments, *George Bush*, *the president*, *held a press conference yesterday*. The last fragment parses as a VP and then an attempt is made to parse the string "the president, VP" as a single category which it is not. So "the president" is parsed as an NP. Now the string "George Bush,NP,VP" is parsed as an appositive on the subject.

The algorithm is superior to the more obvious one – to parse each fragment individually from left to right and then to attempt to piece the fragments together. The latter algorithm would need to look inside all but the last of the fragments for possible attachment positions. This idea of terminal substring parser is appealing, although it did not prove very satisfactory in terms of performance for the task at hand.

2.13 Sparser

Sparser [McDonald, 1993] was developed in the context of an information extraction system to serve as a unrestricted wide-coverage robust parser that would parse text as it comes, without any preprocessing of the text. The design of Sparser was based on the fact that most wide coverage grammars lack grammatical analyses of the breadth of linguistic phenomena appearing in real text. Hence, Sparser was designed as a partial parser that would extract a particular class of information that is specific to the task, instead of attempting to extract everything in the text. Sparser analyzes the part of the text that it is designed to analyze while skipping the 'extra-grammatical' portions of the text.

The following are the components of Sparser. These components can be configured in

any control structure as needed. Components can feed each other, on subsequent passes, information which might not have been available in the first pass.

1. A tokenizer: This is a very conservative tokenization algorithm that groups words of the same class together. So *$4.5 million* is seen as 6 tokens and left for subsequent components to combine them based on the context. This delay could be advantageous. F-14 could be a key on a keyboard or may be a airplane depending on the context. The tokenizer also deals with suffixes of unknown words to determine their POS.

2. A set of transition networks: These networks serve to group adjacent elements such as proper names, numbers and some word compounds into phrases with a flat structure. Though these phrases can be handled by grammatical rules (Phrase Structure rules) it is more natural to state them in terms of transition nets, especially since they have a flat structure.

3. A CFG parser: This is a chart-based parser that uses semantic constraints that are highly specific for the domain. For example, the word *dollar* is represented by a triplet, a semantic label *currency*, a syntactic label *head noun* and a structured ambiguity between *physical object – quantity of money worth 100 cents* and *the value of U.S currency on the international money market*. In other words, the parser integrates the process of word-sense disambiguation directly with the process of grammatical analysis. This approach obviously reduces the structural ambiguity of a purely syntactic analyzer. However, the conflation of syntactic and domain-specific semantic information limits the modularity and portability of the system to new domains.

4. A Context-Sensitive parser: This component consists of highly domain-specific rules that apply only when the context associated with the rule matches the context around the rule application in the text. An example rule is shown here.

   name → person / __ <was named the CEO>

5. A Conceptual Analyzer: This component is responsible for skipping over extragrammatical regions of the text and composing semantically related constituents that may

not be adjacent to one another. For example, a subject and predicate separated by a long appositive can be brought together using semantic and conceptual information.[5]

6. Forming Phrases Heuristically: This is a heuristic facility to identify phrasal boundaries using grammatical properties of function words and partially parsed phrases. This is particularly useful when the system encounters unknown words. It uses function words and suffixes of unknown words to determine the phrase boundaries even though it may not recognize most of the words in the phrase.

In summary, Sparser is a partial parser that is intended to serve as a front-end to an information extraction system. It relies heavily on the domain knowledge for syntactic analyses, and for combining partial parses. Hence, issues of modularity and portability are of major concern in this system.

2.14 IBM's P-CFG and HBG Model

This work [Black *et al.*, 1993b] is distinctive from the other systems discussed in this section in that robustness in this system was achieved using a probabilistic model of parsing. The goal of this work was to integrate rich contextual information into a probabilistic extension of a hand-crafted context-free grammar. Two models of including probabilities in the hand-crafted context-free grammar were explored in this work – the probabilistic context-free grammar (P-CFG) model and the history-based grammar (HBG) model. Both models used a treebank of computer manual sentences that were annotated with the correct parse as training material.

In the P-CFG model, the rules in the hand-crafted context-free grammar were annotated with probabilities which were estimated using an adaptation of the inside-outside algorithm. The adaptation was that only parses that are consistent with the treebank would contribute to the reestimation of the parameters of the P-CFG. After training, the P-CFG was tested on 760 computer manual sentences with lengths ranging from 7 to 17 words. The performance of the parsing model was measured using two measures – the *any-consistent rate* and the *Viterbi rate*. The *any-consistent rate* measured the percentage

[5]The use of a grammar formalism that provides a greater domain of locality than afforded by a CFG will eliminate this problem.

of sentences for which the parse consistent with the treebank is proposed among the many parses produced by the parser. The *Viterbi rate* measured the percentage of sentences for which the most likely parse produced by the parser was consistent with the treebank parse. A parse is regarded as consistent with the treebank parse if it matches the treebank parse structurally including the labels for non-terminals. The P-CFG model achieved a performance of 90% *any-consistent rate* and 60% *Viterbi rate*.

Unlike the P-CFG model where the grammar rule expansion was independent of the context, in the HBG model the rule expansion was conditioned on rich contextual features. The contextual features included syntactic, semantic and two lexical features. The model predicts the syntactic and semantic labels of a constituent along with its rewrite rule and its lexical features using the labels of the parent constituent, the parent's lexical heads, the parent's rewrite rule that lead to the constituent and the constituent's index in the parent's rule. These parameters are estimated using a statistical decision tree technique. The HBG model performed at a *Viterbi rate* of 75% as opposed to the 60% performance of the P-CFG model, an error reduction of 37%. Thus the HBG model showed that incorporating rich contextual features into a probabilistic model and conditioning rule expansions on the derivation structure can improve the performance of a probabilistic grammar. This provided a new approach to grammar development where the problem of parsing is divided into two parts – extending the grammar coverage and picking the correct parse for a sentence.

2.15 Summary

As can be seen from the recent literature in robust parsing, there are two significant approaches emerging. The grammar based approach that incorporates some heuristic and ranking information for selecting competing parses and resorting to partial parses to achieve robustness and the statistical approach that employ an annotated treebank to induce probability distributions pertaining to parsing decisions. Although there are evaluation metrics that have been used to rank statistical parsers, there are no metrics as yet to rank partial parsers among themselves or against statistical parsers. In chapter 7, we propose evaluation metrics that could be used to measure the performance of partial

parsers as well as statistical parsers.

Chapter 3

Merits of LTAG for Partial Parsing

Lexicalized Grammars are particularly well-suited for the specification of natural language grammars. The lexicon plays a central role in linguistic formalisms such as LFG [Kaplan and Bresnan, 1983], GPSG [Gazdar *et al.*, 1985], HPSG [Pollard and Sag, 1987], CCG [Steedman, 1987], Lexicon-Grammars [Gross, 1984], LTAG [Schabes and Joshi, 1991], Link Grammars [Sleator and Temperley, 1991], and some version of GB [Chomsky, 1992]. Parsing, lexical semantics and machine translation, to name a few areas, have all benefited from lexicalization. Lexicalization provides a clean interface for combining the syntactic and semantic information in the lexicon.

In this chapter, we discuss the merits of lexicalization and other related issues in the context of partial parsing. We take Lexicalized Tree-Adjoining Grammars (LTAGs) as a representative of the class of lexicalized grammars. This chapter is organized as follows. Section 3.1 provides an introduction to Feature-based Lexicalized Tree-Adjoining Grammars with an example. Section 3.2 discusses in detail the merits of LTAGs for partial parsing. In Section 3.3, we briefly discuss the salient details of a wide-coverage grammar system developed in LTAG formalism.

3.1 Feature-based Lexicalized Tree-Adjoining Grammar

Feature-based Lexicalized Tree-Adjoining Grammar (FB-LTAGs) is a tree-rewriting grammar formalism unlike Context-Free Grammars and Head-Grammars which are string-rewriting formalisms. FB-LTAGs trace their lineage to Tree Adjunct Grammars (TAGs), which were first developed in [Joshi *et al.*, 1975] and later extended to include unification-based feature structures [Vijay-Shanker, 1987; Vijay-Shanker and Joshi, 1991] and lexicalization [Schabes *et al.*, 1988]. For a more recent and comprehensive reference, see [Joshi and Schabes, 1996].

The primitive elements of FB-LTAGs are called *Elementary trees*. Each elementary tree is associated with at least one lexical item on its frontier. The lexical item associated with an elementary tree is called the anchor of that tree. An elementary tree serves as a complex description of the anchor and provides a domain of locality over which the anchor can specify syntactic and semantic (predicate-argument) constraints. Elementary trees are of two kinds: (a) *Initial Trees* and (b) *Auxiliary Trees*. In an FB-LTAG grammar for natural language, *initial trees* are phrase structure trees of simple sentences containing no recursion, while recursive structures are represented by *auxiliary trees*.

Examples of initial trees (αs) and auxiliary trees (βs) are shown in Figure 3.2. Nodes on the frontier of initial trees are marked as substitution sites by a '$\downarrow$', while exactly one node on the frontier of an auxiliary tree, whose label matches the label of the root of the tree, is marked as a foot node by a '$*$'. The other nodes on the frontier of an auxiliary tree are marked as substitution sites.

Each node of an elementary tree is associated with two feature structures (FS), the top and the bottom. The bottom FS contains information relating to the subtree rooted at the node, and the top FS contains information relating to the supertree at that node.[1] Features may get their values from three different sources:

- Morphology of anchor: from the morphological information of the lexical items that anchor the tree.
- Structural characteristics: from the structure of the tree itself (for example, the *mode = ind/imp* feature on the root node in the α_3 tree in Figure 3.2).

[1] Nodes marked for substitution are associated with only the top FS.

- The derivation process: from unification with features from trees that adjoin or substitute.

Elementary trees are combined by *Substitution* and *Adjunction* operations. Substitution inserts elementary trees at the substitution nodes of other elementary trees. Figure 3.1(a) shows two elementary trees and the tree resulting from the substitution of one tree into the other. In this operation, a node marked for substitution in an elementary tree is replaced by another elementary tree whose root label matches the label of the node. The top FS of the resulting node is the result of unification of the top features of the two original nodes, while the bottom FS of the resulting node is simply the bottom features of the root node of the substituting tree.

In an *adjunction* operation, an auxiliary tree is inserted into an elementary tree. Figure 3.1(b) shows an auxiliary tree adjoining into an elementary tree and the result of the adjunction. The root and foot nodes of the auxiliary tree must match the node label at which the auxiliary tree adjoins. The node being adjoined to splits, and its top FS unifies with the top FS of the root node of the auxiliary tree, while its bottom FS unifies with the bottom FS of the foot node of the auxiliary tree. Figure 3.1(b) shows an auxiliary tree and an elementary tree, and the tree resulting from an adjunction operation. For a parse to be well-formed, the top and bottom FS at each node should be unified at the end of a parse.

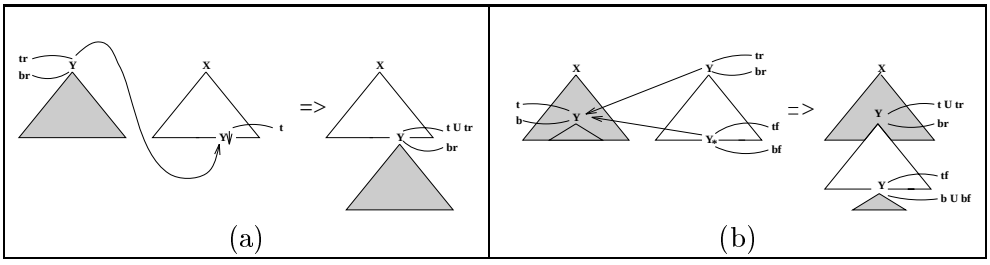


Figure 3.1: Substitution and Adjunction in LTAG

The result of combining the elementary trees shown in Figure 3.2 is the *derived tree*, shown in Figure 3.3(a). The process of combining the elementary trees to yield a parse of the sentence is represented by the *derivation tree*, shown in Figure 3.3(b). The nodes of the derivation tree are the tree names that are anchored by the appropriate lexical items. The

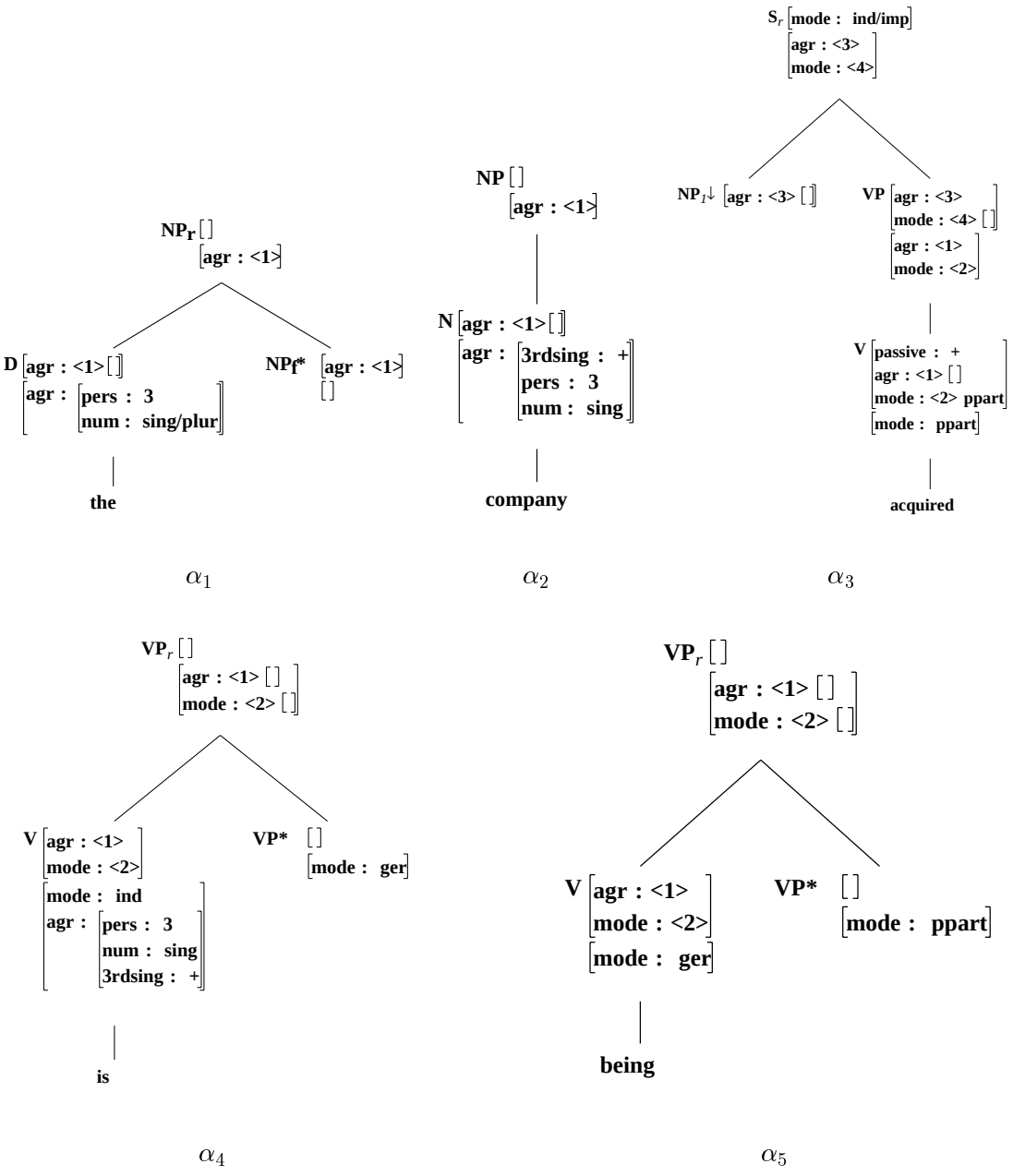
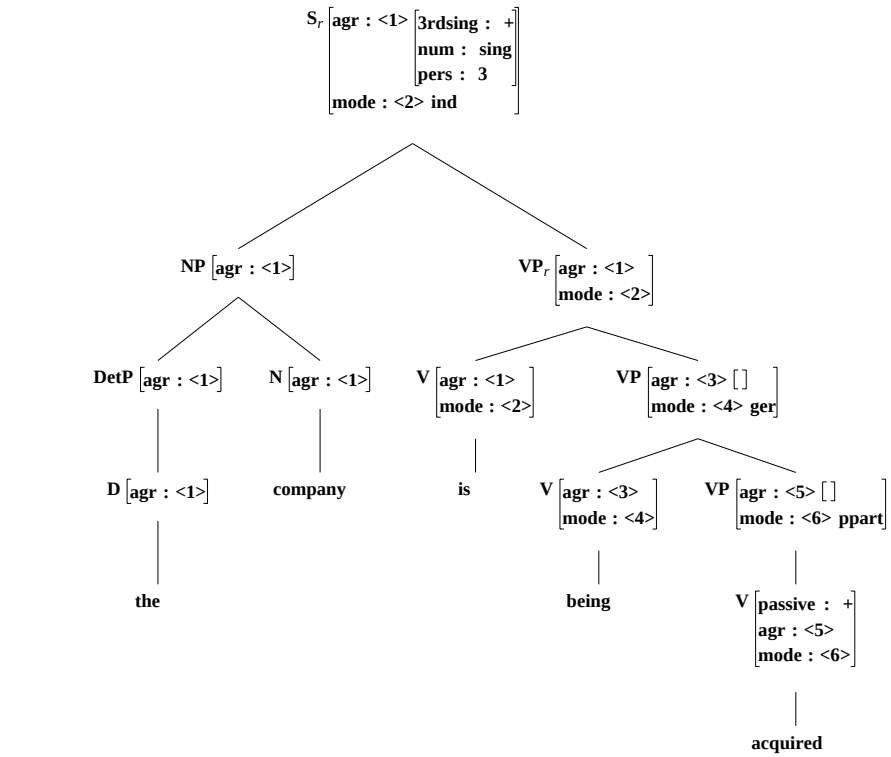
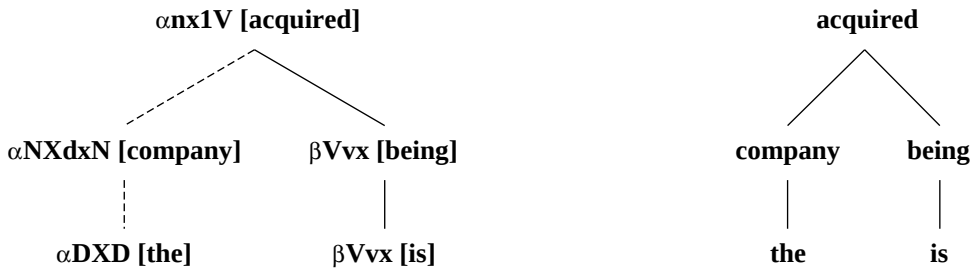


Figure 3.2: Elementary Trees for the sentence: *The company is being acquired*



(a)



(b) (c)

Figure 3.3: (a) Derived Tree, (b) Derivation Tree, and (c) Dependency tree for the sentence: *The company is being acquired*

combining operation is indicated by the type of the arcs (a broken line indicates substitution and a bold line indicates adjunction) while the address of the operation is indicated as part of the node label. The derivation tree can also be interpreted as a *dependency tree* with unlabeled arcs between words of the sentence, as shown in Figure 3.3(c).

3.2 Key properties of LTAGs

In this section, we define the key properties of LTAGs: Lexicalization, Extended Domain of Locality (EDL) and Factoring of Recursion from Domain of Dependency (FRD) and discuss how these properties are realized in natural language grammars written in LTAGs. A more detailed discussion about these properties is presented in [Joshi, 1985; Kroch and Joshi, 1985; Joshi, 1987; Schabes *et al.*, 1988; Joshi and Schabes, 1996].

Lexicalized Grammar: *A grammar is lexicalized if it consists of*

- *a finite set of elementary structures (strings, trees, directed acyclic graphs, etc.), each structure anchored on a lexical item.*
- *lexical items, each associated with at least one of the elementary structures of the grammar*
- *a finite set of operations combining these structures.*

This property proves to be linguistically crucial since it establishes a direct link between the lexicon and the syntactic structures defined in the grammar. In fact, in lexicalized grammars all we have is the lexicon, which projects the elementary structures of each lexical item; there is no independent grammar.

Extended domain of Locality (EDL) *This property has two parts.*

(a) Every elementary structure must contain all and only the arguments of the anchor in the same structure.

(b) For each lexical item, the grammar must contain an elementary structure for each syntactic environment the lexical item might appear in.

Part (a) of EDL allows the anchor to impose syntactic and semantic constraints on its arguments directly since they appear in the same elementary structure that it anchors. Hence, all elements that appear within one elementary structure are considered to be local. This property also defines how large an elementary structure in a grammar can be. Figure 3.4(a) shows the elementary tree anchored by *seem* that is used to derive a raising analysis for sentence 3.1. Notice that the elements appearing in the tree are only those that serve as arguments to the anchor and nothing else. In particular, the Subject NP (*John* in sentence 3.1) does not appear in the elementary tree for *seem* since it does not serve as an argument for *seem*. Figure 3.4(b) shows the elementary tree anchored by the transitive verb *hit* in which both the Subject NP and Object NP are realized within the same elementary tree.

(3.1) John seems to like Mary.

LTAG is distinguished from other grammar formalisms by possessing part (b) of the EDL property. In LTAGs, there is one elementary tree for every syntactic environment that the anchor may appear in. Each elementary tree encodes the linear order of the arguments of the anchor in a particular syntactic environment. For example, a transitive verb such as *hit* is associated with both the elementary tree shown in Figure 3.4(a) for a declarative transitive sentence such as sentence 3.2, and the elementary tree shown in Figure 3.4(b) for an object extracted transitive sentence such as sentence 3.3. Notice that the object noun phrase is realized to the left of the subject noun phrase in the object extraction tree.

(3.2) John hit Mary.

(3.3) Who did John hit.

As a consequence of the fact that LTAGs possess the part(b) of the EDL property, the derivation structure in LTAGs contain the information of a dependency structure. Another aspect of EDL is that the arguments of the anchor can be filled in any order. This is possible because the elementary structures allocate a slot for each argument of the anchor in each syntactic environment that the anchor appears in.

There can be many ways of constructing the elementary structures of a grammar so as to possess the EDL property. However, by requiring that the constructed elementary

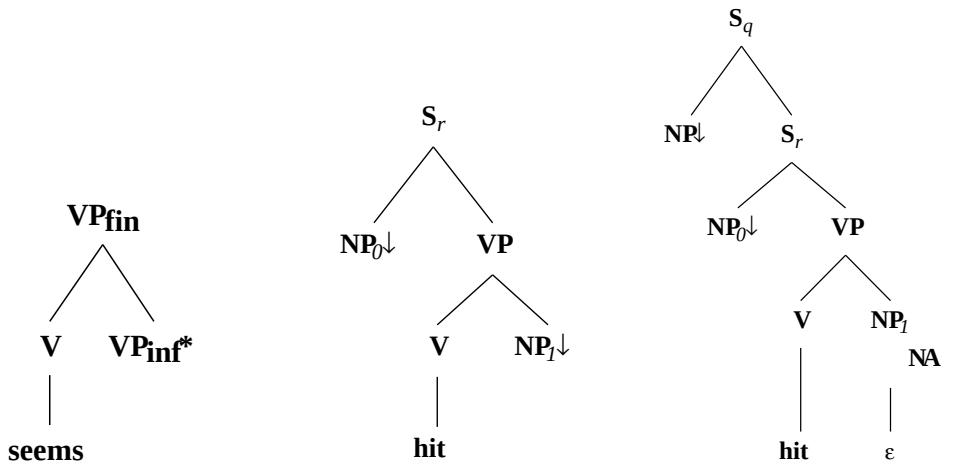


Figure 3.4: (a): Tree for Raising analysis, anchored by *seems* (b): Transitive tree (c): Object extraction tree for the verb *hit*

structures be "minimal", the third property of LTAGs namely, Factoring of Recursion from the Domain of Dependencies, follows as a corollary of EDL.

Factoring of Recursion from the Domain of Dependencies (FRD):

Recursion is factored away from the domain for the statement of dependencies.

In LTAGs, recursive constructs are represented as auxiliary trees. They combine with elementary trees by the operation of adjunction. Elementary trees define the domain for stating dependencies such as agreement, subcategorization, and filler-gap dependencies. Auxiliary trees, by adjunction to elementary trees, account for the long-distance behavior of these dependencies.

An additional advantage of a grammar possessing FRD and EDL properties is that feature structures in these grammars are extremely simple. Since the recursion has been factored out of the domain of dependency, and since the domain is large enough for agreement, subcategorization, and filler-gap dependencies, feature structures in such systems do not involve any recursion. In fact they reduce to typed terms that can be combined by simple term-like unification.

3.2.1 Derivation Structure

In this section, we briefly discuss the notion of a derivation structure and discuss its status in various formalisms. A derivation structure provides information regarding the elementary structures that participated in that derivation and how they were combined during the process of derivation. However, it does not indicate the order of the derivational steps. Thus a derivation structure is a representative of an equivalence class of derivations modulo the order of individual derivational steps.

In string rewriting systems, such as CFGs, there is no independent notion of a derivation structure. The result of a parse is a phrase structure tree that represents the rules that were combined and the result of that combination process.

In Categorial Grammars (CG), the result of a parse is a proof tree. The proof tree is like a phrase structure tree of CFG, where each node is a category label. However, unlike a CFG phrase structure tree, each step in the proof tree is annotated with the operation that was used to combine the categories that appear at that step.

In LTAG, a parse produces two structures – the derived tree and the derivation tree. The derived tree is a phrase structure tree while the derivation tree records the elementary trees that were combined using the substitution and adjunction operations in the derivation process. The nodes of the derivation tree are labels of the elementary trees and the edges are either substitution or adjunction links. The edges are labeled with tree addresses that indicate the location of operation of the tree labeling the daughter node into the tree labeling the parent node.

The fact that LTAGs produce both a phrase structure tree and a derivation tree distinguishes it from its counterparts in the group of Mildly Context-Sensitive Grammars such as Combinatory Categorial Grammars, Head Grammars and Linear Indexed Grammars [Joshi *et al.*, 1990]. This is a direct consequence of the fact that the LTAGs are tree-rewriting systems while its counterparts are string rewriting systems.

Linguistic Relevance of a derivation structure

A phrase structure represents the constituency and linear word order information for the given input. However, predicate-argument relations, dependency relations such as head-complement and head-modifier relations and some aspects of non-compositional semantics

are not encoded in a phrase structure. To compute these relations in a formalism with a CFG backbone requires additional mechanisms such as functional structures, as in LFG.

In a lexicalized formalism, the elementary structures are anchored by a lexical item that serves as the head of the structure. However, in a lexicalized string-rewriting formalism, even though the elementary structures are headed, when two headed strings are combined, the operation that combines them has to choose from the two heads to head the resulting string. Such an approach is pursued in Head Grammars and Headed CFGs [Abney, 1994a].

Derivation Structures and Dependencies in LTAG

As mentioned before, the LTAG derivation structure consists of nodes that are labeled by elementary tree names along with their anchors. The nodes are connected by substitution and adjunction links which can be seen as relating two lexical items. These lexical relations can be identified as head-complement and head-modifier relations. However, as Vijay-Shankar [1992] observes, while substitution is used to add a complement, adjunction is used for modification and clausal complementation. The reason for this is that auxiliary trees could represent head-modifier relations as well as head-complement (clausal) relations. This is because auxiliary trees represent *recursive* structures of the language and are independent of the syntactic relations that they express. Hence it has been suggested that the auxiliary trees be further partitioned into predicative auxiliary trees and modifier auxiliary trees. Predicative auxiliary trees are recursive and record head-complement relations, while modifier auxiliary trees are recursive and record head-modifier relations. As a result of this, it is not the case that complements enter the derivation through substitution and modifiers through adjunction.[2] Also, the derivation structure with predicative auxiliary trees needs to be interpreted differently to obtain the "standard" notion of dependency structure.

In Chapter 4, we discuss how lexicalization and EDL can be exploited for dependency style parsing of LTAGs. In Chapter 5, we discuss how FRD, in conjunction with lexicalization and EDL, can be exploited for parsing LTAGs in restricted domains.

[2]D-Tree Grammars [Rambow *et al.*, 1995] attempt to identify the type of syntactic dependency with the type of operation used to compose the elementary structure.

3.2.2 Categorial Grammars

In this section we briefly review how the properties of Lexicalization, EDL and FRD are realized in natural language grammars written in Categorial Grammar(CG) framework. CGs, like LTAGs, are lexicalized grammars. A CG consists a finite set of categories and every category is anchored by a lexical item. Categories are combined by application and composition operations. However, CGs differ from LTAGs on the EDL property. While CGs possess part (a) they do not have the part(b) of the EDL property. As a consequence of this, in a CG, *hit* is assigned only one category, (S\NP)/NP. This category contains both the arguments of the anchor in the same structure, and encodes the direction of the arguments in the declarative transitive sentence. However, CGs do not contain categories that encode the possible orderings of the arguments in other syntactic environments. Thus *hit* is assigned the same category in sentence 3.2 and sentence 3.3. The derivation for sentence 3.3 requires *type-raising* of the subject [Steedman, 1996]. Hence the category, (S\NP)/NP, alone does not encode the syntactic environment in which it would be used in. A part of the proof tree needs to be specified to determine the ordering of the arguments. Thus the history of the derivation alone does not encode the dependency structure of the input.

In the system described by Joshi and Kulick [1997], the primitive objects are partial proof trees that are formed from the categorial grammar categories by proof building operations. These proof trees, like LTAG trees, encode the possible orderings of the arguments in each syntactic environments they would be used in. In such a system the history of the derivation encodes the dependency structure of the input.

3.2.3 Chunking and LTAGs

As discussed in Chapter 2, a chunk-based partial parser is necessitated from an engineering requirement in wide-coverage grammar systems. The reasons include the following: grammatical coverage of these systems is seldom complete, full parsing is often extremely slow and sometimes requires enormous amount of semantic knowledge, and unrestricted texts are often ungrammatical.

Abney [1991] provides another reason for chunking. His motivation for chunking is

based on psycholinguistic studies on sentence processing by Gee and Grosjean [1983] that link pause duration in reading and naive word grouping to text clusters called ϕ-phrases. Abney [1991] defines *chunks* as the parse tree fragments with "problematic" elements such as conjuncts, modifiers and in some cases, arguments, unattached. A clause is a sequence of chunks with no nesting of chunks. An example of chunks for the sentence 3.4 taken from [Abney, 1991] is shown in Figure 3.5.

(3.4) The professor from Milwaukee was reading about a biography of Marcel Proust.

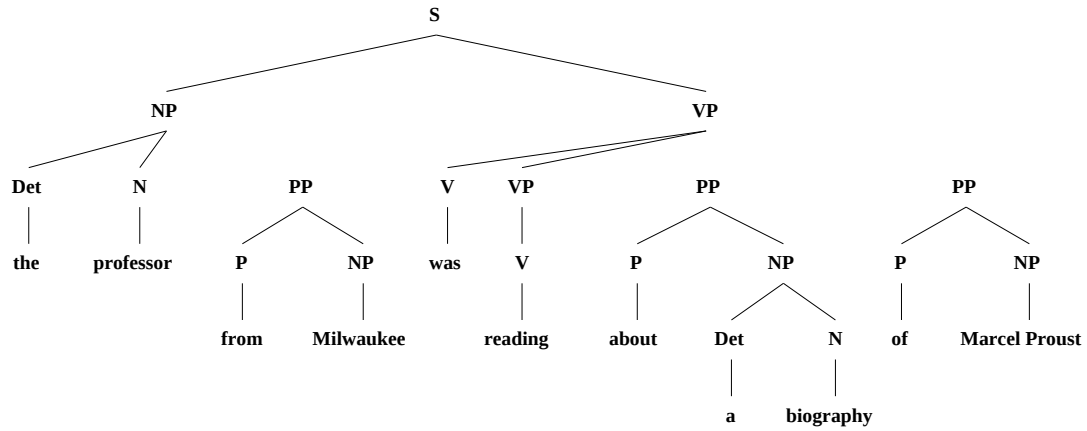


Figure 3.5: Chunks for *The professor from Milwaukee was reading about a biography of Marcel Proust.*

Besides the psycholinguistic motivation, chunking provides a computational advantage. Abney [1991] shows that chunks can be identified with very simple computationally inexpensive finite-state devices. If the decisions of attachments of chunks can be factored out from the task of resolving ambiguities in placing chunk boundaries, then the two tasks can be handled separately and the parsing process could be simplified considerably. Once chunks are identified, they are pieced together using dependency information between chunks.

LTAGs possess properties that are conducive for a chunk-based parsing approach. Elementary trees compile out the various syntactic environments the anchor of the tree can appear in. Also, elementary trees factor out recursion. Hence elementary trees may be regarded as linguistically motivated chunk templates with "slots" for the arguments

of the anchor. Frozen and nearly frozen groups of words or chunks can be represented as multi-component anchors of chunk templates. Identifying chunk boundaries in the input reduces to assignment of the correct elementary trees to the words of the input. Figure 3.6 illustrates the elementary assigned to sentence 3.4 in LTAG. When an anchor of an elementary tree along with its arguments that appear within that elementary tree is viewed as a chunk, then it is hard to miss the striking similarity between LTAG chunks and the chunks proposed by Abney [1991].

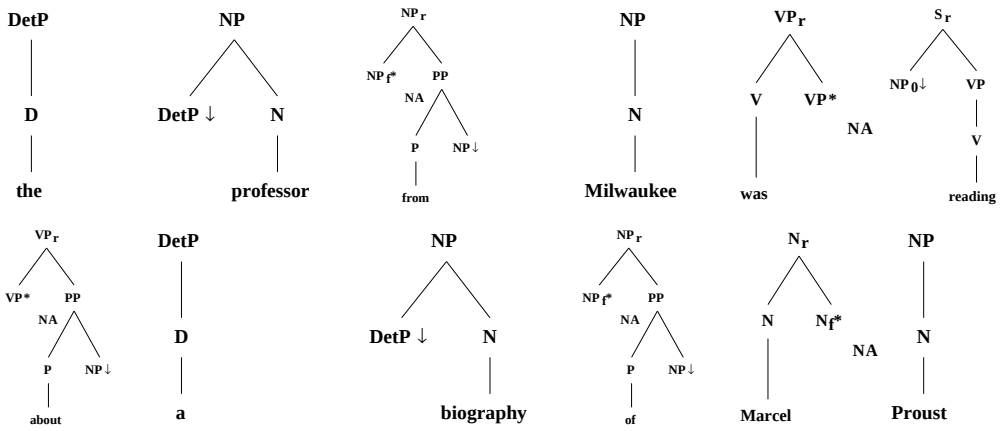


Figure 3.6: Chunks in LTAG for *The professor from Milwaukee was reading about a biography of Marcel Proust.*

Since LTAG represents modifiers, conjuncts and other recursive structures as auxiliary trees, the process of identifying the sites of attachments of these structures can be separated from the process of identifying the chunk boundaries. Also, since the "slots" in a chunk are constrained by the anchor, constraints between chunks can be imposed naturally. Thus, dependencies across chunks can be captured in LTAGs. Stochastic models and simple finite-state devices can be used for assigning elementary trees. Some of these models are discussed in Chapter 4.

3.2.4 Language Modeling and LTAG

There has been an increasing interest in predictive language modeling for a variety of applications including speech recognition, machine translation, and statistical parsing. The most popular language models applied to this task are N-gram based models, largely due

to their success in speech recognition. There have also been many attempts at applying probabilistic context-free grammars as well [Fujisaki *et al.*, 1989; Lafferty *et al.*, 1992; Jurafsky *et al.*, 1995]. However, both these models show a mismatch between the prediction of the model and the distribution of the information. We summarize the inadequacies of these models for the purpose of language modeling. A more detailed discussion is presented in [Hindle, 1993].

- N-gram models: N-gram models predict the next word based on the preceding N-1 words. This model is highly sensitive to lexical associations which are very useful in identifying fixed expressions and idioms. However, this model fails to capture any dependency relation that extends beyond the fixed N-word window. In particular, since it does not include any structural information, it will not be able to predict associations between words that are separated in terms of string positions but are structurally local. For example, using a trigram model, *to* will be predicted in 3.5 but not in 3.6, just because the relative clause separates *to* from its predictor beyond the N-word window.

  (3.5) ... give kittens to ...

  (3.6) ... give kittens that were born yesterday to ...

  Another inadequacy of N-gram models is that different word sequences require different values of N. Hindle [1993] shows that while the phrase *New York Stock Exchange* is followed by the phrase *composite* more than half the time in a 60 million word Wall Street Journal corpus, *composite New York Stock Exchange* is never followed by *composite*. To accommodate this, 5-grams are inadequate. A 6-gram model is required and with it the issues of sparseness of data needs to be addressed.

- Probabilistic Context-Free Grammars (PCFGs): PCFGs are probabilistic versions of CFGs where each production of a CFG is associated with a probability and the probability of a derivation is the product of the probabilities of the rules participating in the derivation. The major problem with PCFGs is that they are not inherently sensitive to lexical associations. So the fact that *powerful tea* is less probable than *strong tea* is not expressed naturally in a PCFG.

Another inadequacy of PCFG is that rule expansions are independent of the derivation context (context free). Hence, the distributional evidence that pronouns are more likely to be subject noun phrases than non-subject noun-phrases is not modeled, since a noun phrase is derived by the same set of rules whether it occurs in the subject position or not.

We now discuss the four features of LTAG that make it a more appropriate formalism for language modeling. First, in LTAGs, since every tree is lexicalized, words are associated more tightly with the structure they anchor. This allows the possibility of making the operations of combining elementary trees to be sensitive to lexical context. Second, since the anchors of trees can potentially be multi-word lexical items, idioms and fixed collocations can be modeled naturally, without any additional mechanism. Third, the EDL property of LTAGs allows the anchor to specify different constraints on the different argument positions in the tree. Thus, the distributional evidence of pronouns being more frequent in a subject noun-phrase position than in a non-subject noun-phrase position can be naturally captured since each noun phrase appearing in an elementary tree can be identified with its grammatical function. Finally, since LTAGs factor out recursion from the domain of dependency, the problem of intervening material disrupting locality as seen in an N-gram approach does not arise. Referring to the previous example, the co-occurrence of *give* and *to* is structurally local and is not affected by the relative clause on *kittens*.

Probabilistic versions of LTAGs have been proposed by Schabes [1992] and Resnik [1992]. Experiments on the efficacy of Probabilistic LTAGs as language models have been stalled due to unavailability of data needed to estimate the statistical parameters of the model. Also, the estimation of the parameters requires a starting grammar. One possibility is to include all possible elementary structures in the starting grammar and let the re-estimation weed out useless structures. Alternatively, a hand-crafted grammar can serve as a very good starting grammar. In the next section, we discuss a wide-coverage grammar system developed in the LTAG formalism, and in subsequent chapters present some novel uses of probabilities in LTAGs.

3.3 XTAG

A broad-coverage grammar system, XTAG, has been implemented in the LTAG formalism. In this section, we briefly discuss some aspects related to XTAG for the sake of completeness. A more detailed report on XTAG can be found in [XTAG-Group, 1995]. The XTAG system consists of a Morphological Analyzer, a Part-of-Speech Tagger, a wide-coverage LTAG English Grammar, a predictive left-to-right Earley-style Parser for LTAG [Schabes, 1990] and an X-windows Interface for Grammar development [Doran *et al.*, 1994]. Figure 3.7 shows a flowchart of the XTAG system. The input sentence is subjected to morphological analysis and is tagged with parts-of-speech before being sent to the parser. The parser retrieves the elementary trees that the words of the sentence anchor and combines them by adjunction and substitution operations to derive a parse of the sentence.

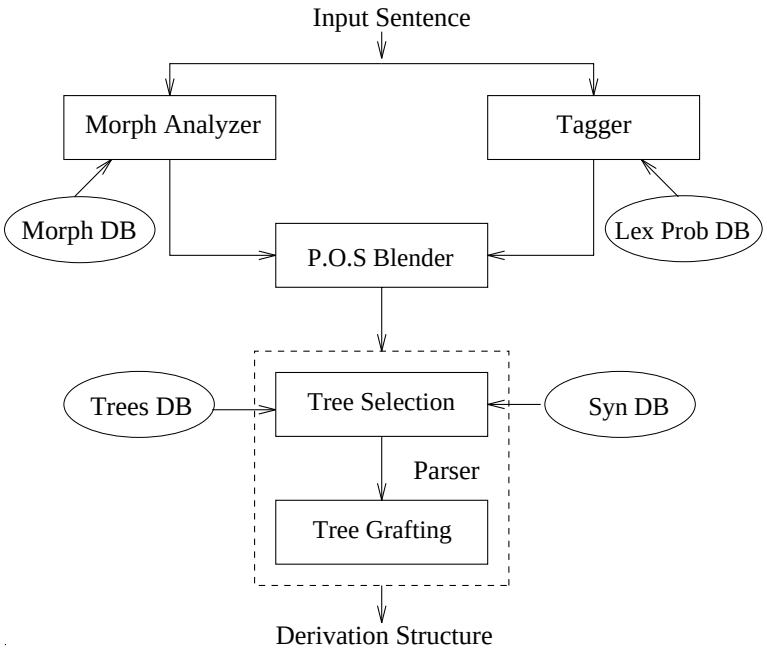


Figure 3.7: Flowchart of the XTAG system

A summary of each of the system component is presented in Table 3.1. A more detailed description of each of these components is presented in [Doran *et al.*, 1994; XTAG-Group, 1995].

Component	Details
Morphological Analyzer and Morph Database	Consists of approximately 317,000 inflected items. Thirteen parts of speech are differentiated. Entries are indexed on the inflected form and return the root form, POS, and inflectional information. Database does not address derivational morphology.
POS Tagger and Lexical Probabilities Database	Wall Street Journal-trained trigram tagger [Church, 1988] extended to output N-best POS sequences [Soong and Huang, 1990]. Decreases the time to parse a sentence by an average of 93%.
POS Blender	Combines information from the Morphology and the POS tagger. Outputs N-best POS sequences with morphological information for each word of the input sentence.
Tree Database	566 trees, divided into 40 tree families and 62 individual trees. Tree families represent subcategorization frames. E.g., the intransitive tree family contains the following trees: indicative, wh-question, relative clause, imperative and gerund. Individual trees are generally anchored ($\diamond$) by non-verbal lexical items that substitute or adjoin into the clausal trees. Feature values may be specified within a tree or may be derived from the syntactic database.
Syntactic Database and Statistical Database	Associates lexical items with the appropriate trees and tree families based on subcategorization information. Extracted from OALD and ODCIE and contains more than 105,000 entries. Each entry consists of: the uninflected form of the word, its POS, the list of trees or tree-families associated with the word, and a list of feature equations that capture lexical idiosyncrasies.
X-Interface	Provides the following: Menu-based facility for creating and modifying tree files; User controlled parser parameters: parser's start category, enable/disable/retry on failure for POS tagger; Storage/retrieval facilities for elementary and parsed trees as text and postscript files; Graphical displays of tree and feature data structures; Hand combination of trees by adjunction or substitution for diagnosing grammar problems.

Table 3.1: XTAG System Summary

The grammar of XTAG has been used to parse sentences from ATIS, IBM manual and WSJ corpora [XTAG-Group, 1995]. The resulting XTAG corpus contains sentences from these domains along with all the derivations for each sentence. The derivations provide predicate argument relationships for the parsed sentences. This information is used in the experiments discussed in the subsequent chapters.

3.4 Summary

In this chapter, we introduced the LTAG formalism and reviewed the properties of lexicalization, extended domain of locality and factoring of recursion from the domain of dependencies in LTAGs. We also discussed the relevance of these properties for partial parsing. Finally, we presented a brief overview of XTAG, a large wide-coverage grammar for English that is being developed in the LTAG formalism. In the following chapters we discuss two novel methods of exploiting the properties of LTAGs for partial parsing.

Chapter 4

Supertags

Part-of-speech disambiguation techniques (POS taggers) [Church, 1988; Weischedel *et al.*, 1993; Brill, 1993] are often used prior to parsing to eliminate (or substantially reduce) the part-of-speech ambiguity. The POS taggers are all local in the sense that they use information from a limited context in deciding which tag(s) to choose for each word. As is well known, these taggers are quite successful.

In a lexicalized grammar such as the Lexicalized Tree-Adjoining Grammar (LTAG), each lexical item is associated with at least one elementary structure (tree). The elementary structures of LTAG localize dependencies, including long distance dependencies, by requiring that all and only the dependent elements be present within the same structure. As a result of this localization, a lexical item may be (and, in general, almost always is) associated with more than one elementary structure. We will call these elementary structures *supertags*, in order to distinguish them from the standard part-of-speech tags. Note that even when a word has a unique standard part-of-speech, say a verb (V), there will usually be more than one supertag associated with this word. Since there is only one supertag for each word (assuming there is no global ambiguity) when the parse is complete, an LTAG parser [Schabes *et al.*, 1988] needs to search a large space of supertags to select the right one for each word before combining them for the parse of a sentence. It is this problem of supertag disambiguation that we address in this chapter.

Since LTAGs are lexicalized, we are presented with a novel opportunity to eliminate or substantially reduce the supertag assignment ambiguity by using local information such as

local lexical dependencies, prior to parsing. As in standard part-of-speech disambiguation, we can use local statistical information in the form of N-gram models based on the distribution of supertags in a LTAG parsed corpus. Moreover, since the supertags encode dependency information, we can also use information about the distribution of distances between a given supertag and its dependent supertags.

Note that as in standard part-of-speech disambiguation, supertag disambiguation could have been done by a parser. However, carrying out part-of-speech disambiguation prior to parsing makes the job of the parser much easier and therefore speeds it up. Supertag disambiguation reduces the work of the parser even further. After supertag disambiguation, we would have effectively completed the parse and the parser need 'only' combine the individual structures; hence the term–*almost parsing*. This method can also be used to parse sentence fragments and in cases where the supertag sequence after disambiguation may not combine into a single structure.

In this chapter, we present techniques for disambiguating supertags, and evaluate their performance and their impact on LTAG parsing. Although presented with respect to LTAG, these techniques are applicable to other lexicalized grammars as well. Section 4.1 illustrates the objective of supertag disambiguation through an example. Various methods and their performance results for supertag disambiguation are discussed in detail in Section 4.2 and Section 4.3. In Section 4.4, we discuss the efficiency gained in performing supertag disambiguation before parsing. A robust and lightweight dependency analyzer that uses the supertag output is presented in Section 4.5. In Section 4.6, we will discuss the applicability of Supertag disambiguation to other lexicalized grammars.

4.1 Example of Supertagging

LTAGs, by the virtue of possessing the Extended Domain of Locality (EDL) property, associate with each lexical item, one elementary tree for each syntactic environment that the lexical item may appear in. As a result, each lexical item is invariably associated with more than one elementary tree. We call the elementary structures associated with each lexical item as super parts-of-speech (super POS) or **supertags**. Figure 4.1 illustrates a few elementary trees associated with each word of the sentence, *the purchase price includes*

two ancillary companies. Table 4.1 provides an example contexts in which each supertag shown in Figure 4.1 would be used.

Supertag	Construction	Example
α_1	Nominal Predicative	this is the *purchase*
α_2	Noun Phrase	the *price*
α_3	Topicalization	Almost everything, the price *includes*
α_4	Adjectival Predicative	this is *ancillary*
α_5	Noun Phrase	the *company*
β_1	Determiner	*the* company
β_2	Nominal Modifier	*purchase* order
α_6	Nominal Predicative Subject Extraction	what is the *price*
α_7	Imperative	*include* the share price
β_3	Determiner	*two* hundred men
β_4	Adjectival Modifier	*ancillary* unit
α_8	Nominal Predicative Subject Extraction	which are the *companies*
α_9	Noun Phrase	*purchases* have not increased.
α_{10}	Nominal Predicative	this is the *price*
α_{11}	Transitive Verb	the price *includes* everything
α_{12}	Adjectival Predicative Subject Extraction	what is *ancillary*
α_{13}	Noun Phrase	*companies* have not been profitable

Table 4.1: Examples of syntactic environments
where the supertags shown in Figure 4.1 would be used.

The example in Figure 4.2 illustrates the initial set of supertags assigned to each word of the sentence *the purchase price includes two ancillary companies.* The order of the supertags for each lexical item in the example is not relevant. Figure 4.2 also shows the final supertag sequence assigned by the supertagger which picks the best supertag sequence using statistical information (described in Section 4.3) about individual supertags and their dependencies on other supertags. The chosen supertags are combined to derive a parse. Without the supertagger, the parser would have to process combinations of the entire set of trees (at least the 17 trees shown); with it the parser need only processes combinations of 7 trees.

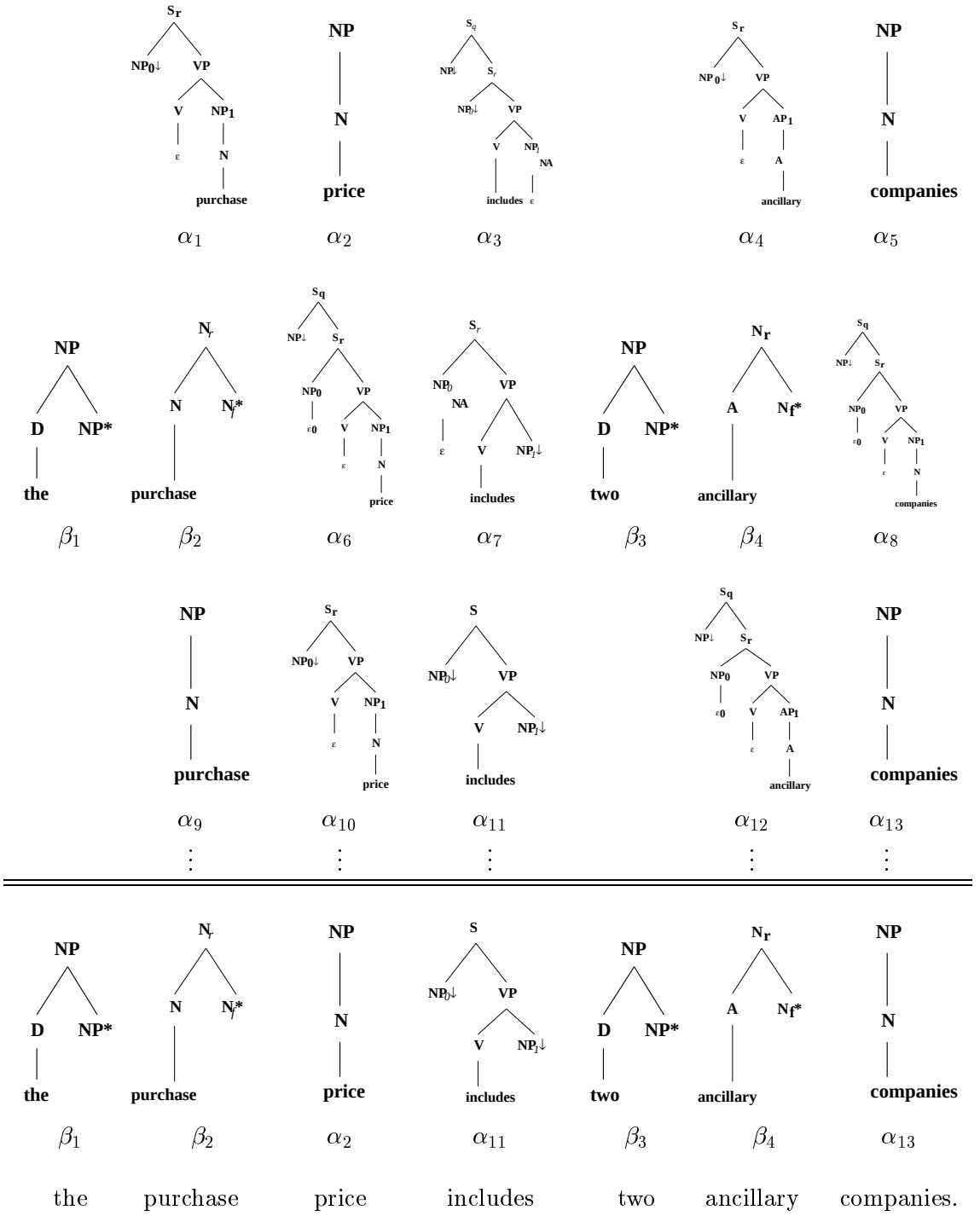


Figure 4.1: A selection of the supertags associated with each word of the sentence *the purchase price includes two ancillary companies*

Sent:	the	purchase	price	includes	two	ancillary	companies.
Initial		α_1	α_2	α_3		α_4	α_5
Assig.	β_1	β_2	α_6	α_7	β_3	β_4	α_8
		α_9	α_{10}	α_{11}		α_{12}	α_{13}
		$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
Final Assig.	β_1	β_2	α_2	α_{11}	β_3	β_4	α_{13}

Figure 4.2: Supertag disambiguation for the sentence *the purchase price includes two ancillary companies*

4.2 Reducing supertag ambiguity using structural information

The structure of the supertag can be best seen as providing admissibility constraints on syntactic environments in which it may be used. Some of these constraints can be checked locally. The following are a few constraints that can be used to determine the admissibility of a syntactic environment for a supertag.[1]

- Span of the supertag : Span of a supertag is the minimum number of lexical items that the supertag can cover. Each substitution site of a supertag will cover at least one lexical item in the input. A simple rule can be used to eliminate supertags based on the *span constraint*: if the span of a supertag is larger than the input string, then the supertag cannot be used in any parse of the input string.

- Left (Right) span constraint: If the span of the supertag to the left (right) of the anchor is larger than the length of the string to the left (right) of the word that anchors the supertag, then the supertag cannot be used in any parse of the input string.

[1]Prof. Mitch Marcus pointed out that these tests are similar to the *generalized shaper tests* used in the Harvard Predictive Analyzer[Kuno, 1966].

- Lexical items in the supertag: A supertag can be eliminated if the terminals appearing on the frontier of the supertag do not appear in the input string. Supertags with the built-in lexical item *by*, that represent passive constructions are typically eliminated from being considered during the parse of an active sentence.

  More generally, these constraints can be used to eliminate supertags that cannot have their features satisfied in the context of the input string. An example of this is the elimination of supertag that requires a *wh*+ NP when the input string does not contain wh-words.

Table 4.2 indicates the decrease in supertag ambiguity for 2012 WSJ sentences (48,763 words)[2] by using the structural constraints relative to the supertag ambiguity without the structural constraints.

System	Total # of words	Av. # of Supertags/word
Without structural constraints	48,783	47.0
With structural constraints	48,783	25.0

Table 4.2: Supertag ambiguity with and without the use of structural constraints

These filters prove to be very effective in reducing supertag ambiguity. The graph in Figure 4.3 plots the number of supertags at the sentence level for sentences of length 2 to 50 words with and without the filters. As can be seen from the graph, the supertag ambiguity is significantly lower when the filters are used. The graph in Figure 4.4 shows the percentage drop in supertag ambiguity due to filtering for sentences of length 2 to 50 words. As can be seen the average reduction in supertag ambiguity is about 50%. This means that given a sentence, close to 50% of the supertags can be eliminated even before parsing begins by just using structural constraints of the supertags. This reduction in supertag ambiguity speeds up the parser significantly. In fact, the supertag ambiguity in XTAG system is so large that the parser is prohibitively slow without the use of these filters.

Table 4.3 tabulates the reduction of supertag ambiguity due to the filters against various parts-of-speech. Verbs in all their forms contribute most to the problem of supertag ambiguity and most of the supertag ambiguity for verbs is due to light verbs and verb

[2] wsj_20 of the Penn Treebank

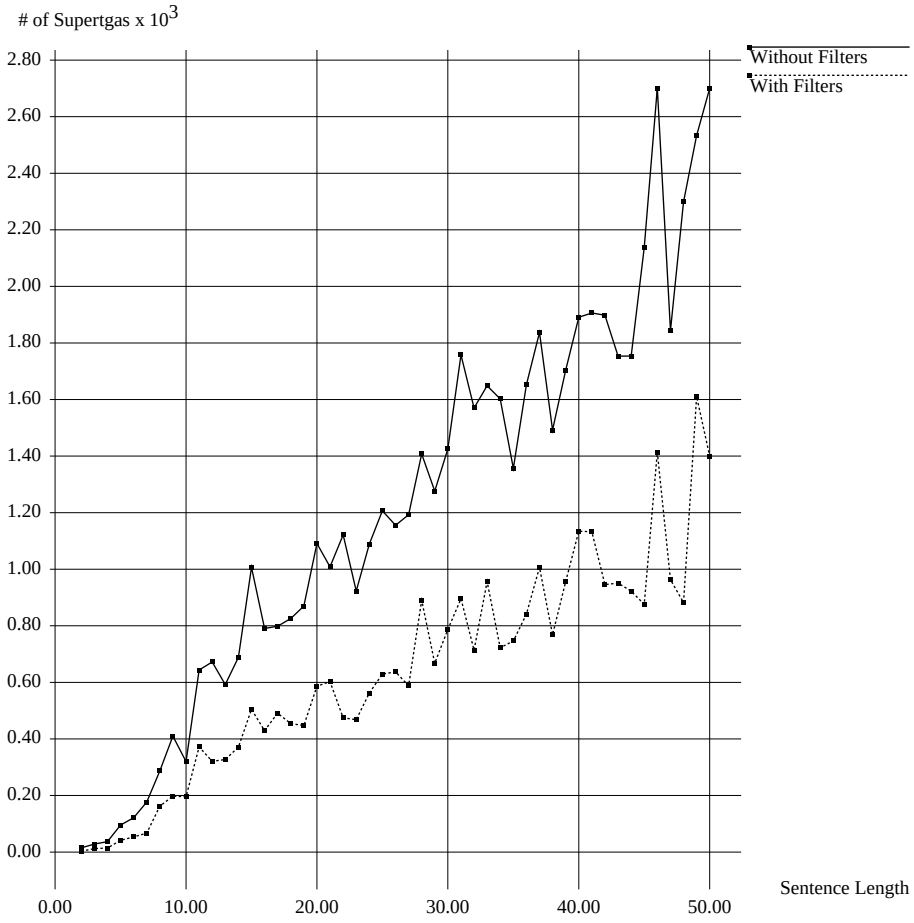


Figure 4.3: Comparison of number of supertags with and without filtering for sentences of length 2 to 50 words.

particles. The filters are very effective in eliminating over 50% of the verb anchored supertags.

Even though structural constraints are effective in reducing supertag ambiguity, the search space for the parser is still sufficiently large. In the next few sections, we present stochastic and rule-based approaches to supertag disambiguation.

POS	Average # of supertags without filters	Average # of supertags with filters	Percentage drop in supertag ambiguity
VBP	516.5	250.0	51.6
VB	435.8	224.9	48.4
VBD	209.0	100.7	51.8
VBN	188.2	74.7	60.3
MD	167.2	121.0	27.6
VBZ	165.1	71.6	56.6
VBG	100.7	49.8	50.5
RP	34.5	30.9	10.5
IN	24.3	20.9	14.0
JJS	23.8	12.7	46.9
WRB	23.1	14.3	38.2
JJR	22.7	14.2	37.7
JJ	21.7	13.5	37.9
,	20.0	10.7	46.6
NN	19.8	10.7	46.0
NNS	17.0	10.5	38.6
NNP	15.0	10.2	31.9
NNPS	15.0	10.2	32.1
LS	15.0	15.0	0.0
FW	15.0	15.0	0.0
-RRB-	15.0	10.7	28.4
-LRB-	15.0	12.3	18.0
RBR	14.9	9.5	36.3
RBS	14.9	6.1	59.2
CC	14.8	3.4	76.9
EX	14.0	5.8	58.7
CD	13.3	9.9	25.8
TO	11.3	10.8	4.5
PRP	10.7	5.3	50.2
UH	10.0	3.0	70.0
RB	10.0	5.3	46.4
“	6.0	3.2	46.7
:	5.5	3.2	42.1
PDT	5.4	4.9	9.0
WP	4.6	2.9	35.8
WP$	4.0	1.8	56.2
DT	3.9	3.1	21.8
PRP$	3.8	2.9	22.2
.	3.0	1.0	65.4
POS	2.5	2.1	13.9
WDT	1.2	1.1	5.5

Table 4.3: The effect of filters on supertag ambiguity tabulated against part-of-speech.

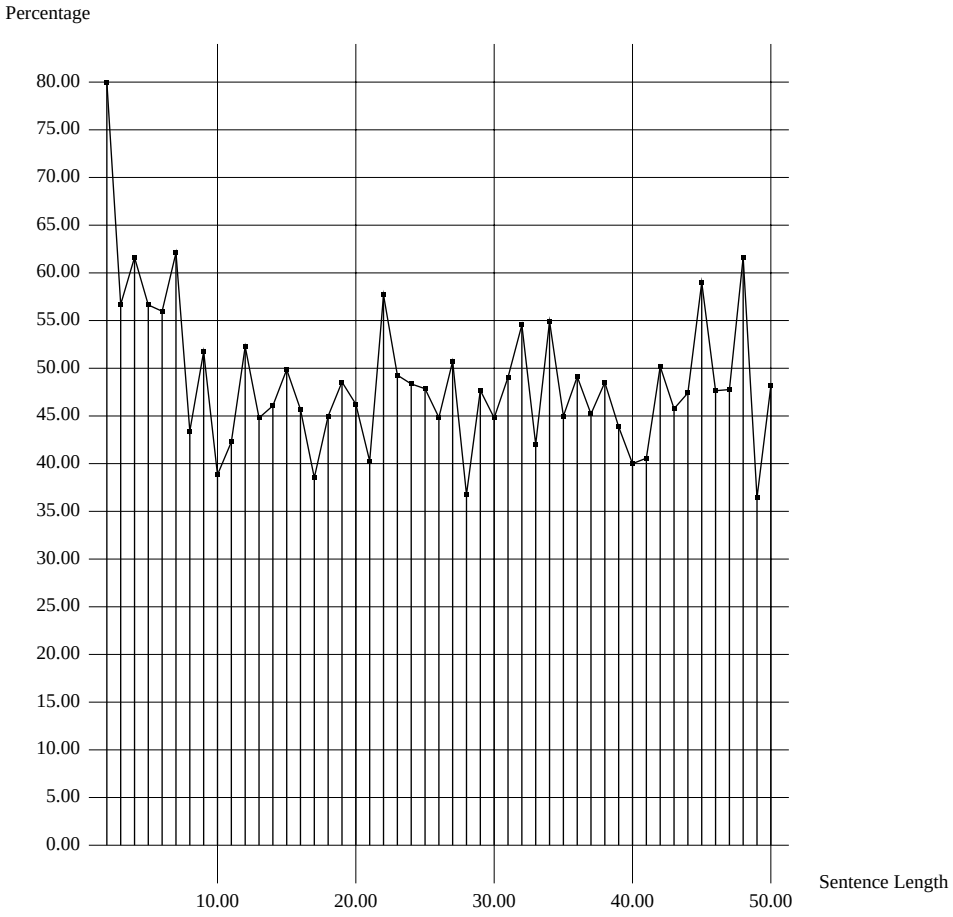


Figure 4.4: Percentage drop in the number of supertags with and without filtering for sentences of length 2 to 50 words.

4.3 Models, Data, Experiments and Results

Before proceeding to discuss the various models for supertag disambiguation we would like to trace the time course of development of this work. We do this not only to show the improvements made to the early work reported in COLING'94 [Joshi and Srinivas, 1994] but also to explain the rationale for choosing certain models of supertag disambiguation over others. We summarize the early work in the following subsection.

4.3.1 Early Work

As reported in the COLING'94 paper, we experimented with a trigram model as well as the dependency model for supertag disambiguation. The trigram model that was trained on (part-of-speech, supertag) pairs collected from the LTAG derivations of 5000 WSJ sentences and tested on 100 WSJ sentences produced a correct supertag for 68% of the words in the test set. We have since significantly improved the performance of the trigram model by using a larger training set and incorporating smoothing techniques. We present a detailed discussion of the model and its performance on a range of corpora in Section 4.3.5. In this section, we describe the dependency model of supertagging that was reported in the early work.

4.3.2 Dependency model

In an n-gram model for disambiguating supertags, dependencies between supertags that appear beyond the n word window cannot be incorporated. This limitation can be overcome if no a priori bound is set on the size of the window but instead a probability distribution of the distances of the dependent supertags for each supertag is maintained. We define dependency between supertags in the obvious way: A supertag is *dependent* on another supertag if the former substitutes or adjoins into the latter. Thus, the substitution and the foot nodes of a supertag can be seen as specifying dependency requirements of the supertag. The probability with which a supertag depends on another supertag is collected from a corpus of sentences annotated with derivation structures. Given a set of supertags for each word and the dependency information between pairs of supertags, the objective of the dependency model is to compute the most likely dependency linkage that spans the entire string. The result of producing the dependency linkage is a sequence of supertags, one for each word of the sentence along with the dependency information.

Experiments and Results

The data required for the dependency model was collected from all the derivation structures of 5000 WSJ sentences. A derivation structure associates each word with a supertag and links two supertags with substitution and adjunction operations. Using the derivation

structure, the *word emit probability* and the *dependency link probability* are computed. The word emit probability, as defined in (4.1), is the probability of emitting a word given a supertag. The dependency link probability, defined in (4.2), is the probability that supertag t_j at ordinal distance d_k depends on another supertag t_i.

(4.1) $\Pr(w|t) = \frac{frequency(w,t)}{frequency(t)}$

(4.2) $\Pr(t_j,d_k|t_i) = \frac{frequency(t_i,t_j,d_k)}{\sum_{t_l}\sum_{d_m \ni sign(d_m)=sign(d_k)} frequency(t_i,t_l,d_m)}$

Due to sparseness of data, we used POS tags instead of words. Since the POS is completely determined given a supertag, the word emit probability did not play a role in this experiment. Table 4.4 shows the data needed for the dependency model of supertag disambiguation. Each entry contains the following information.

(P.O.S,Supertag)	Direction of Dependent Supertag	Dependent Supertag	Ordinal Position	Probability
(D,β_1)	(+)	α_{13}	+1	0.97
(N,β_5)	(+)	α_{13}	+1	0.58
(N,α_{13})	()	–	–	–
(V,α_{15})	(–, +)	α_{13}	−1	0.70
(V,α_{15})	(–, +)	α_{13}	1	0.42

Table 4.4: Dependency Data

- POS and Supertag pair.
- List of + and −, representing the direction of the dependent supertags with respect to the indexed supertag. (Size of this list indicates the total number of dependent supertags required.)
- Dependent supertag.
- Signed number representing the direction and the ordinal position of the particular dependent supertag mentioned in the entry from the position of the indexed supertag.
- The probability of occurrence of such a dependency. The sum probability over all the dependent supertags at all ordinal positions in the same direction is one.

For example, the fourth entry in the Table 4.4 reads that the tree α_{15}, anchored by a verb (V), has a left and a right dependent ($-$, $+$) and the first word to the left (-1), with the tree α_{13}, is dependent on the current word. The strength of this association is represented by the probability 0.70.

The algorithm for computing a dependency linkage is similar to the CKY algorithm for categorial grammars [Steedman, 1997] and for link grammars [Sleator and Temperley, 1991]. The dependency model of disambiguation works as follows. Suppose α_{15} is a member of the set of supertags associated with a word at position n in the sentence. The algorithm proceeds to satisfy the dependency requirement of α_{15} by picking up the dependency entries for each of the directions. It picks a dependency data entry (the fourth entry, say) from the database that is indexed by α_{15} and proceeds to set up a path with the first word to the left that has the dependent supertag (α_{13}) as a member of its set of supertags. If the first word to the left that has α_{13} as a member of its set of supertags is at position m, then an arc is set up between α_{15} and α_{13}. Also, the arc is verified not to kite-string-tangle[3] with any other arcs in the path up to α_{15}. The path probability up to α_{15} is incremented by $\log 0.70$ to reflect the success of the match. The path probability up to α_{13} incorporates the unigram probability of α_{13}. On the other hand, if no word is found that has α_{13} as a member of its set of supertags then the entry is ignored. The algorithm makes a greedy choice by selecting the path with the maximum path probability to extend to the remaining directions in the dependency list. A successful supertag sequence is one which assigns a supertag to each position such that each supertag has all of its dependents and maximizes the accumulated path probability.

Table 4.5 shows results on the same held out test set of 100 Wall Street Journal sentences. The table shows two measures of evaluation. In the first, the dependency link measure, the test sentences were independently hand tagged with dependency links and then were used to match the links output by the dependency model. The columns show the total number of dependency links in the hand tagged set, the number of matched links output by this model and the percentage correctness. The second measure, supertags, shows the total number of correct supertags assigned to the words in the corpus by this model.

[3]Two arcs (a,c) and (b,d) kite-string-tangle if $a < b < c < d$ or $b < a < d < c$.

Criterion	Total number	Number correct	% correct
Dependency links	815	620	76.07%
Supertags	915	707	77.26%

Table 4.5: Results of Dependency model

Discussion

Since first reported in [Joshi and Srinivas, 1994], we have not continued experiments using this model of supertagging, primarily for two reasons. Although we are certain that the performance of this model can be greatly improved if lexical probabilities are taken into account, we are restrained by the lack of a large corpus of LTAG parsed derivation structures that is needed to reliably estimate the various parameters of this model. We are currently in the process of collecting a large LTAG parsed WSJ corpus, with each sentence annotated with the correct derivation. A second reason for the disuse of the dependency model for supertagging is that the objective of supertagging is to see how far local techniques can be used to disambiguate supertags even before parsing begins. The dependency model, in contrast, is too much like full parsing and is contrary to the spirit of supertagging.

4.3.3 Recent Work

In recent work, we have improved the performance of trigram model by incorporating smoothing techniques into the model and training the model on a larger training corpus. We have also proposed some new models for supertag disambiguation. In this section, we discuss these developments in detail.

Two sets of data are used for training and testing the models for supertag disambiguation. The first set has been collected by parsing the Wall Street Journal[4], IBM-manual and ATIS corpora using the wide-coverage English grammar being developed as part of the XTAG system [Doran *et al.*, 1994]. The correct derivation from all the derivations

[4]Sentences of length $\leq$ 15 words

produced by the XTAG system was picked for each sentence from these corpora.

The second and larger data set was collected by converting the Penn Treebank parses of the Wall Street Journal sentences. The objective was to associate each lexical item of a sentence with a supertag, given the phrase structure parse of the sentence. This process involved a number of heuristics based on local tree-contexts. The heuristics availed to information about the labels of its dominating nodes (parent, grandparent and great grandparent), labels of its siblings (left and right) and siblings of its parent. It must be noted that this conversion is not perfect and is correct only to a first order of approximation owing to mostly to errors in conversion and lack of certain kinds of information such as distinction among adjunct and argument preposition phrases, in the Penn Treebank parses. Even though the converted supertag corpus can be refined further, the corpus in its present form has proved to be an invaluable resource in improving the performance of the supertag models as is discussed in the following sections.

4.3.4 Unigram model

Using structural information to filter out supertags that cannot be used in any parse of the input string reduces the supertag ambiguity but obviously does not eliminate it completely. One method of disambiguating the supertags assigned to each word is to order the supertags by the lexical preference that the word has for them. The frequency with which a certain supertag is associated with a word is a direct measure of its lexical preference for that supertag. Associating frequencies with the supertags and using them to associate a particular supertag with a word is clearly the simplest means of disambiguating supertags. Therefore a unigram model is given by,

(4.3) $\text{Supertag}(w_i) = t_k \ni \text{argmax}_{t_k} \Pr(t_k \mid w_i)$.

where

(4.4) $\Pr(t_k \mid w_i) = \frac{frequency(t_k, w_i)}{frequency(w_i)}$

Thus, the most frequent supertag that a word is associated with in a training corpus is selected as the supertag for the word according to the unigram model. For the words that do not appear in the training corpus we back-off to the part-of-speech of the word

and use the most frequent supertag associated with that part-of-speech as the supertag for the word.

Experiments and Results

We tested the performance of the unigram model on the previously discussed two sets of data. The words are first assigned standard parts-of-speech using a conventional tagger [Church, 1988] and then are assigned supertags according to the unigram model. A word in a sentence is considered correctly supertagged if it is assigned the same supertag as it is associated with in the correct parse of the sentence. The results of these experiments are tabulated in Table 4.6.

Data Set	Training Set	Test Set	Top n Supertags	% Success
XTAG Parses	8,000	3,000	$n = 1$	73.4%
			$n = 2$	80.2%
			$n = 3$	80.8%
Converted Penn Treebank Parses	1,000,000	47,000	$n = 1$	77.2%
			$n = 2$	87.0%
			$n = 3$	91.5%

Table 4.6: Results from the Unigram Supertag Model

Although the performance of the unigram model for supertagging is significantly lower than the performance of the unigram model for part-of-speech tagging (91% accuracy), it performed much better than expected considering the size of the supertag set is much larger than the size of part-of-speech tagset. One of the reasons for this high performance is that the most frequent supertag for the most frequent words – determiners, nouns and auxiliary verbs is the correct supertag most of the time. Also, backing-off to the part-of-speech helps in supertagging unknown words, which most often are nouns. Bulk of the errors committed by the unigram model is incorrectly tagged verbs (subcateogrization and transformation), prepositions (noun attached vs verb attached) and nouns (head vs modifier noun).

4.3.5 n-gram model

We had first explored the use of trigram model of supertag disambiguation in [Joshi and Srinivas, 1994]. The trigram model was trained on (part-of-speech, supertag) pairs collected from the LTAG derivations of 5000 WSJ sentences and tested on 100 WSJ

sentences. It produced a correct supertag for 68% of the words in the test set. A major drawback of this early work was that it used no lexical information in the supertagging process as the training material consisted of (part-of-speech, supertag) pairs. Since that early work we have improved the performance of model by incorporating lexical information and sophisticated smoothing techniques besides training on a larger training sets. In this section, we present the details and the performance evaluation of this model.

In a unigram model, a word is always associated with the supertag that is most preferred by the word, irrespective of the context in which the word appears. An alternate method that is sensitive to context is the n-gram model. The n-gram model takes into account the contextual dependency probabilities between supertags within a window of n words in associating supertags to words. Thus, the most probable supertag sequence for a N word sentence is given by:

(4.5) $\hat{T} = \text{argmax}_T \Pr(T_1,T_2,\ldots,T_N) * \Pr(W_1,W_2,\ldots,W_N \mid T_1,T_2,\ldots,T_N)$

where T_i is the supertag for word W_i.

To compute this using only local information, we approximate, assuming that the probability of a word depends only on its supertag

(4.6) $\Pr(W_1,W_2,\ldots,W_N|T_1,T_2,\ldots,T_N) \approx \prod_{i=1}^{N} \Pr(W_i \mid T_i)$

and also use an n-gram (trigram, in this case) approximation

(4.7) $\Pr(T_1,T_2,\ldots,T_N) \approx \prod_{i=1}^{N} \Pr(T_i \mid T_{i-2}, T_{i-1})$

The term $\Pr(T_i \mid T_{i-2}, T_{i-1})$ is known as the *contextual probability* since it indicates the size of the context used in the model and the term $\Pr(W_i \mid T_i)$ is called as the *word emit probability* since it is the probability of emitting the word W_i given the tag T_i. These probabilities are estimated using a corpus where each word is tagged with its correct supertag.

The contextual probabilities were estimated using the relative frequency estimates of the contexts in the training corpus. To estimate the probabilities for contexts that do

not appear in the training corpus, we used the Good-Turing discounting technique [Good, 1953] combined with Katz's back-off model [Katz, 1987]. The idea here is to discount the frequencies of events that occur in the corpus by an amount proportional to their frequencies and utilize this discounted probability mass in the back-off model to distribute to unseen events. Thus, the Good-Turing discounting technique estimates the frequency of unseen events based on the distribution of the frequency of frequency of observed events in the corpus. If r is the observed frequency of an event, and N_r is the number of events with the observed frequency r and N is the total number of events, then the probability of an unseen event is given by N_1/N. Furthermore, the frequencies of the observed events are adjusted so that the total probability of all events sums to 1. The adjusted frequency for observed events, r^*, is computed as

(4.8) $r^* = (\mathrm{r}+1)*\frac{N_{r+1}}{N_r}$

Once the frequencies of the observed events is discounted and the frequencies for unseen events is estimated, Katz's back-off model is used. In this technique, if the observed frequency of an <n-gram, supertag> sequence is zero then its probability is computed based on the observed frequency of a (n-1)-gram sequence. Thus,

$$\begin{aligned}
\tilde{Pr}(T_3|T_1,T_2) &= Pr(T_3|T_1,T_2)\ if\ Pr(T_3|T_1,T_2) > 0 \\
&= \alpha(T_1,T_2) * \tilde{Pr}(T_3|T_2)\ \ if\ Pr(T_2|T_1) > 0 \\
&= Pr(T_3|T_2)\ otherwise
\end{aligned}$$

$$\begin{aligned}
\tilde{Pr}(T_2|T_1) &= Pr(T_2|T_1)\ \ if\ Pr(T_2|T_1) > 0 \\
&= \beta(T_1) * Pr_1(T_2)\ otherwise
\end{aligned}$$

where $\alpha(T_i,T_j)$ and $\beta(T_k)$ are constants to ensure that the probabilities sum to one.

The word emit probability for the {word,supertag} pairs that appear in the training corpus is computed using the relative frequency estimates:

$$\begin{aligned}
Pr(w_i|T_i) &= \frac{N(w_i,T_i)}{N(T_i)}\ if\ N(w_i,T_i) > 0 \\
&= Pr(UNK|T_i) * Pr(word-features|T_i) otherwise
\end{aligned}$$

The counts for the {word,supertag} pairs for the words that do not appear in the corpus is estimated using the leaving-one-out technique [T.R. Niesler and P.C. Woodland, 1996; Ney *et al.*, 1995]. A token "UNK" is associated with each supertag and its count N_{UNK} is estimated by:

$$Pr(UNK|T_j) = \frac{N_1(T_j)}{N(T_j) + \eta}$$

$$N_{UNK}(T_j) = \frac{Pr(UNK|T_j) * N(T_j)}{1 - Pr(UNK|T_j)}$$

where $N_1(T_j)$ is the number of words that are associated with the supertag T_j that appear in the corpus exactly once. $N(T_j)$ is the frequency of the supertag T_j and $N_{UNK}(T_j)$ is the estimated count of UNK in T_j. The constant η is introduced so as to ensure that the probability is not greater than one, especially for supertags that are sparsely represented in the corpus.

We use word features similar to the ones used in [Weischedel *et al.*, 1993], such as capitalization, hyphenation and endings of words, for estimating the word emit probability of unknown words.

Experiments and Results

We tested the performance of the trigram model on various domains such as Wall Street Journal (WSJ), the IBM Manual Corpus and the ATIS. For the IBM Manual Corpus and the ATIS domains, a supertag annotated corpus was collected using the parses of the XTAG system [Doran *et al.*, 1994] and selecting the correct analysis for each sentence. The corpus was then randomly split into training and test material. Supertag performance is measured as the percentage of words that are correctly supertagged by a model when compared with the key for the words in the test corpus.

Experiment 1: (Performance on the Wall Street Journal corpus) We used the two sets of data, from the XTAG parses and from the conversion of the Penn Treebank parses to evaluate the performance of the trigram model. Table 4.7 shows the performance

on the two sets of data. The first data set, data collected from the XTAG parses, was split into 8,000 words of training and 3,000 words of test material. The data collected from converting the Penn Treebank was used in two experiments differing in the size of the training corpus; 200,000 words[5] and 1,000,000 words[6] and tested on 47,000 words[7]. A total of 300 different supertags were used in these experiments.

Data Set	Size of training set (words)	Training	Size of test set (words)	% Correct
XTAG Parses	8,000	Unigram (Baseline)	3,000	73.4%
		Trigram	3,000	86%
Converted Penn Treebank Parses	200,000	Unigram (Baseline)	47,000	75.3%
		Trigram	47,000	90.9%
	1,000,000	Unigram (Baseline)	47,000	77.2%
		Trigram	47,000	92.2%

Table 4.7: Performance of the supertagger on the WSJ corpus

Experiment 2: (Performance on the IBM Manual corpus and ATIS) For testing the performance of the trigram supertagger on the IBM manual corpus, a set of 14000 words correctly supertagged was used as the training corpus and a set of 1000 words was used as a test corpus. The performance of the supertagger on this corpus is shown in Table 4.8. Performance on the ATIS corpus was evaluated using a set of 1500 words correctly supertagged as the training corpus and a set of 400 words as a test corpus. The performance of the supertagger on the ATIS corpus is also shown in Table 4.8.

As expected, the performance on the ATIS corpus is higher than that of the WSJ and the IBM Manual corpus despite the extremely small training corpus. Also, the performance of the IBM Manual corpus is better than the WSJ corpus when the size of the training corpus is taken into account. The baseline for the ATIS domain is remarkably high due to the repetitive constructions and limited vocabulary in that domain. This is also true for the IBM Manual corpus, although to a lesser extent. The trigram model of supertagging

[5]Sentences in wsj_15 through wsj_18 of Penn Treebank.
[6]Sentences in wsj_00 through wsj_24, except wsj_20 of Penn Treebank.
[7]Sentences in wsj_20 of Penn Treebank.

Corpus	Size of training set (words)	Training	Size of test set (words)	% Correct
IBM Manual	14,000	Unigram (Baseline)	1,000	77.8%
		Trigram	1,000	90.3%
ATIS	1,500	Unigram (Baseline)	400	85.7%
		Trigram	400	93.8%

Table 4.8: Performance of the supertagger on the IBM Manual corpus and ATIS corpus

is attractive for limited domains since it performs quite well with relatively insignificant amounts of training material. The performance of the supertagger can be improved in an iterative fashion by using the supertagger to supertag larger amounts of training material which can be quickly hand corrected and used to train a better performing supertagger.

Effect of Lexical versus Contextual Information

Lexical information contributes most to the performance of a POS tagger, since the baseline performance of assigning the most likely POS for each word produces 91% accuaracy [Brill, 1993]. Contextual information contributes relatively small amount towards the performance, improving it from 91% to 96-97%, a 5.5% improvement. In contrast, contextual information has greater effect on the performance of the supertagger. As can be seen, from the above experiments, the baseline performance of the supertagger is about 77% and the performance improves to about 92% with the inclusion of contextual information, an improvement of 19.5%. The relatively low baseline performance for the supertagger is a direct consequence of the fact that there are many more supertags per word than there are POS tags. Further, since many combinations of supertags are not possible, contextual information has a larger effect on the performance of the supertagger.

4.3.6 Error-driven Transformation-based Tagger

In an Error-driven Transformation-based (EdTb) tagger [Brill, 1993], a set of pattern-action templates that include predicates which test for features of words appearing in the

context of interest are defined. These templates are then instantiated with the appropriate features to obtain transformation rules. The effectiveness of a transformation rule to correct an error and the relative order of application of the rules are learnt using a corpus. The learning procedure takes a gold corpus in which the words have been correctly annotated and a training corpus that is derived from the gold corpus by removing the annotations. The objective in the learning phase is to learn the optimum ordering of rule applications so as to minimize the number of tag mismatches between the training and the gold corpus.

Experiments and Results

A EdTb model has been trained using templates defined on a three word window. We trained the templates on 200,000 words[8] and tested on 47,000 words[9] of the WSJ corpus. The model performed at an accuracy of 90%. The EdTb model provides a great deal of flexibility to integrate domain specific and linguistic information into the model. However, a major drawback is that the training procedure is extremely slow; it took 18 days to train on the 200,000 words but could not complete training on the 1,000,000 data set due to down time of the computer during the training process.

4.3.7 Head Trigram Model

As is well known, a trigram model is inadequate at modeling dependencies that appear beyond a three word window. This may not be a severe limitation in the case of predicting POS tags since long range dependencies between POS labels are not very typical. However, since supertags encode rich syntactic information such as the subcategorization of a verb, syntactic transformations and attachment preferences, selection of some supertags is dictated by dependencies that usually span beyond a three word window. The performance of the trigram supertagger can be improved further if these dependencies are modeled. Consider, for example, sentences in (4.9), (4.10) and (4.11) as examples illustrating the inadequacies of a trigram supertagger.

(4.9) An assassin in Colombia killed a federal judge *on* a Medellin street.

[8] wsj_15 to wsj_18 of the Penn Treebank
[9] wsj_20 of the Penn Treebank

(4.10) First Options didn't have to *put* any money *into* the bailout.

(4.11) the man Mr. Krenz *replaces* has left an indelible mark on East German society.

In sentence (4.9), the decision that the preposition *on* is assigned an argument supertag or NP adjunct supertag or a VP adjunct supertag is made based on the supertags for *federal* and *judge*. However, ideally, the decision should be based on the supertags of the verb *killed* and the noun *judge*. Similarly, in sentence (4.10), the decision if the preposition *into*, serves as an adjunct or an argument is made based on supertags for *any* and *money* since the verb is out of the trigram context. In sentence (4.11), the relative clause supertag for the verb *replaces* would be hard to get since the word *man* is beyond the trigram window. This makes the trigram context appear no different from the context in which an indicative transitive supertag would appear in. Thus, the limitation of a trigram model is that supertags from the preceding context that influence the selection of a supertag for the current word are not "visible" in the current context. We term the supertags in the prior context that influence the selection of the current supertag as *head supertags* and the words they are associated with as *head words*.

One method of allowing supertags beyond the trigram window to influence the selection of a supertag is to allow it to be "visible" in the current context. This can be achieved in one of two ways. First, the size of the window could be increased to be more than three words. One major drawback of this method is that since an a priori bound is set on the size of the window, no matter how large it is, dependencies that appear beyond that window size cannot be adequately modeled.[10] Also, by increasing the size of the window, the number of parameters to be estimated increases exponentially and sparseness of data becomes a significant problem.

An alternate approach to making head supertags "visible" in the current context is to first identify the head positions and then percolate the supertags at the head positions through the preceding context. Once the positions of the head supertags are identified, the contextual probability in a trigram model is changed so as to be conditioned on the supertags of the two previous head words instead of the two immediately preceding previous supertags. Thus, the contextual probability for the head trigram model is given as:

[10] An interesting approach of incorporating variable length n-grams in a part-of-speech tagging model is presented in [T.R. Niesler and P.C. Woodland, 1996].

(4.12) $\Pr(T_1,T_2,\ldots,T_N) \approx \prod_{i=1}^{N} \Pr(T_i|\ T_{H_{i-2}},\ T_{H_{i-1}})$

where

$T_{H_{i-1}}$ and $T_{H_{i-2}}$ are the supertags associated with the two previous head words.

The head positions are percolated according to the following equations.

(4.13)

Initialize:	(H_{-2},H_{-1})	$= (-2,-1)$
Update :	(H_{i-1},H_i)	$= (H_{i-2},H_{i-1})$ if W_i is not a head word
(head positions at i+1)		$= (H_{i-1},i)$ if W_i is a head word

The head trigram supertagger has two passes. In the first pass, the words are annotated for heads and in the second pass the head information is used to compute the appropriate contextual probability. For the first pass, the head words can be identified using a stochastic trigram tagger that uses two labels and tags words either as head words or as non-head words. The training and test material for the head-word tagger was created from the Penn Treebank parses of WSJ sentences using head percolation rules presented in [Jelinek *et al.*, 1994; Magerman, 1995]. The tagger was trained on 1,000,000 words of WSJ corpus and tested on 47,000 words from Section 20 of WSJ. The tagger assigned the correct head-tag for 91.2% of the words, improving on the baseline of 81.4%. Most of head-word tagging errors were mistakes of tagging modifier nouns as head words in nouns group sequences. Using the head information provided by head-word tagger, the head trigram supertagger, trained on the 1,000,000 words of WSJ corpus and tested on 47,000 words, assigned the correct supertag for 87% of words. We suspect that since local constraints are not modeled by the head trigram model, the performance of the trigram model is better than this model. We are working towards integrating the two models to produce a mixed model that takes into account both local and non-local constraints.

4.3.8 Head Trigram Model with Supertags as Feature Vectors

As previously mentioned, supertags contain subcategorization information and localize filler-gap dependency information within the same structure. As a result, for example, there are not only supertags that realize the complements of a transitive verb in their canonical position, but there are supertags that realize complements in passives, supertags for wh-extraction and relative clause for each of the argument positions. These various

supertags are present for each subcategorization frame and are transformationally related to one another in a movement-based approach. The supertags related to one another are said to belong to a "family". The supertags for verbs can be organized in a hierarchy as shown in Figure 4.5. It must be noted that just as the hierarchy in Figure 4.5 is organized in terms of the subcategorization information, similar hierarchies can be organized based on transformations such as Relative Clause or Wh-extraction.

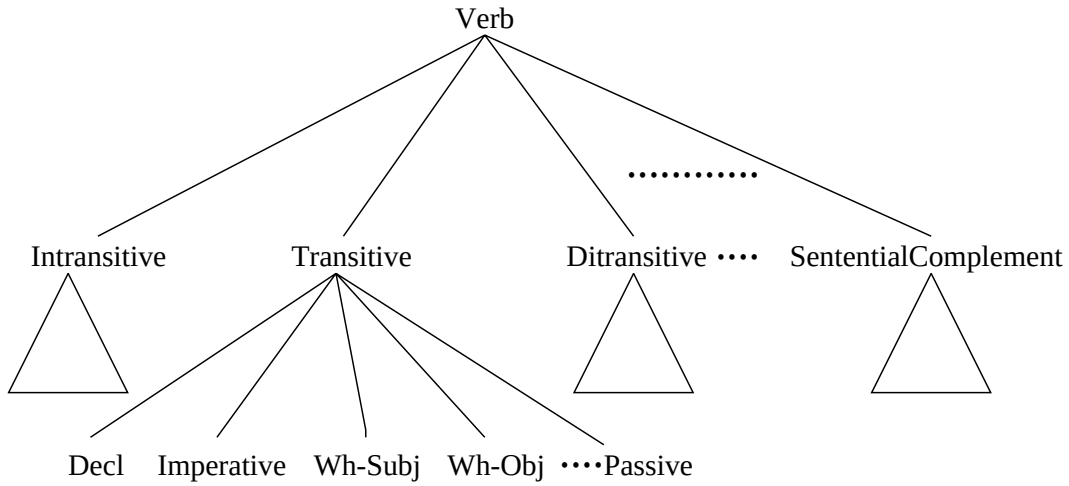


Figure 4.5: Supertag hierarchy

The supertags for other categories such as nouns, adjectives and prepositions can also be organized in similar hierarchies based on features such as modifiers, complements and predicates.

The representation of supertags as labels, as was done in the previous models, does not exploit the relationship among supertags. Instead, supertags can be viewed as feature vectors where each feature represents a dimension along which the supertags can be clustered. Table 4.9 shows the list of features used in representing supertags as feature vectors.

We are currently working on implementing a trigram model with appropriate back-off strategies. The trigram model, treating the supertags as feature vectors, is given in 4.14. A more appropriate model, we believe, for such a distributed encoding of supertags is a model that has the flexibility to select the best back-off strategy as in maximum entropy

Feature	Description
Part-of-speech	Part of speech of anchor, 13 different parts of speech.
Type of supertag	modifier or complement supertag
Transformation	Gerundive, Imperative, Relative Clause, Wh-extraction, Indicative
Argument position	for the transformation: Subject/Object/Indirect Object
Passive	flag to indicate passive supertag
Size of argument frame	number of arguments required by a supertag
Type of argument	constituent type of the argument: NP, S, N, A, Adv, VP, PP
Direction of argument	relative to the anchor of a supertag: left or right
Nature of argument	complement or modifiee

Table 4.9: The list of features used to encode supertags

model. We plan to implement a maximum entropy model for supertagging in the near future.

(4.14) $\Pr(\vec{T}_1,\vec{T}_2,\ldots,\vec{T}_N) = \text{argmax}_{\vec{T}} \prod_{i=1}^{N} \Pr(W_i \mid \vec{T}_i) * \Pr(\vec{T}_i \mid \vec{T}_{i-2}, \vec{T}_{i-1})$

The output of this model is a feature vector for each word of input sentence. The performance of the model can be evaluated on a number of criteria, the most straightforward being an exact match of the feature vector to the gold standard. However, other metrics of performance could focus on the accuracy of predicting any one or combination of the features used to represent a supertag. For example, information pertaining to subcategorization (number, direction and type of arguments) and nature of transformation are the more important features useful for further processing. These features could be used for evaluating the supertagger.

4.4 Supertagging before Parsing

As mentioned earlier, a lexicalized grammar parser can be conceptualized to consist of two stages [Schabes *et al.*, 1988]. In the first stage, the parser looks up the lexicon and selects all the supertags associated with each word of the sentence to be parsed. In the second stage, the parser searches the lattice of selected supertags in an attempt to combine them using substitution and adjunction operations so as to yield a derivation that spans the

input string. At the end of second stage, the parser would not only have parsed the input, but would have associated a small set of (usually one) supertag with each word.

The supertagger can be used as a front-end to a lexicalized grammar parser so as to prune the search space of the parser even before parsing begins. It should be clear that by reducing the number of supertags that are selected in the first stage, the search space for the second stage can be reduced significantly and hence the parser can be made more efficient. Supertag disambiguation techniques, as discussed in the previous sections attempt to disambiguate the supertags selected in the first pass, based on lexical preferences and local lexical dependencies so as to ideally select one supertag for each word. Once the supertagger selects the appropriate supertag for each word, the second stage of the parser is needed 'only' to combine the individual supertags to arrive at the parse of the input. Tested on about 1300 WSJ sentences with each word in the sentence correctly supertagged, the LTAG parser took approximately 4 seconds per sentence to yield a parse (combine the supertags and perform feature unification). In contrast, the same 1300 WSJ sentences without the supertag annotation took nearly 120 seconds per sentence to yield a parse. Thus the parsing speed-up gained by this integration is a factor of about 30.

In the XTAG system, we have integrated the trigram supertagger as a front-end to an LTAG parser to pick the appropriate supertag for each word even before parsing begins. However, a drawback of this approach is that the parser would fail completely if any word of the input is incorrectly tagged by the supertagger. This problem could be circumvented to an extent by extending the supertagger to produce N-best supertags for each word. Although this extension would increase the load on the parser, it would certainly improve the chances of arriving at a parse for a sentence. In fact, Table 4.10 presents the performance of the supertagger that selects at most top three supertags for each word. The optimum value of the number of supertags that need to be output so as to balance the success rate of the parser against the efficiency of the parser must be determined empirically.

A more serious limitation of this approach is that it fails to parse ill-formed and extragrammatical strings such as those encountered in spoken utterances and unrestricted texts. This is due to the fact the Earley style LTAG parser attempts to combine the supertags so as to construct a parse that spans the entire string. In cases where the

Data Set	Size of test set (words)	Size of training set (words)	Training	% Correct
Converted Penn Treebank Parses	47,000	200,000	Trigram (Best Supertag)	90.9%
			Trigram (3-Best Supertags)	95.8%
		1,000,000	Trigram (Best Supertag)	92.2%
			Trigram (3-Best Supertags)	97.1%

Table 4.10: Performance improvement of 3-best supertagger over the 1-best supertagger on the WSJ corpus

supertag sequence for a string cannot be combined into a unified structure, the parser fails completely. One possible extension to account for ill-formed and extragrammatical strings is to extend the Earley parser to produce partial parses for the fragments whose supertags can be combined. An alternate method of computing dependency linkages robustly is presented in the next section.

4.5 Lightweight Dependency Analyzer

Supertagging associates each word with a unique supertag. To establish the dependency links among the words of the sentence, we exploit the dependency requirements encoded in the supertags. Substitution nodes and foot nodes in supertags serve as slots that must be filled by the arguments of the anchor of the supertag. A substitution slot of a supertag is filled by the complements of the anchor while the foot node of a supertag is filled by a word that is being modified by the supertag. These argument slots have a polarity value reflecting their orientation with respect to the anchor of the supertag. Also associated with a supertag is a list of internal nodes (including the root node) that appear in the supertag. Using the structural information coupled with the argument requirements of a supertag, a simple algorithm such as the one below provides a method for annotating the sentence with dependency links.

Step 1: For each modifier supertag s in the sentence
Compute the dependencies for s
Mark the words serving as complements as unavailable for step 2.

Step 2: For the initial supertags s in the sentence
Compute the dependencies for s

Compute Dependencies for s_i of w_i:

For each slot d_{ij} in s_i do
Connect word w_i to the nearest word w_k to the left or right of w_i depending on the direction of d_{ij}, skipping over marked supertags, if any, such that $d_{ij} \in$ internal_nodes(s_k)

An example illustrating the output from this algorithm is shown in Table 4.11. The first column lists the word positions in the input, the second column lists the words, the third lists the names of the supertags assigned to each word by a supertagger. The slot requirement of each supertag is shown in column four and the dependency links among the words, computed by the above algorithm, is shown in the seventh column. The two pass process for dependency linking is shown in the fifth and the sixth columns. The $*$ and the . beside a number indicate the type of the dependency relation, $*$ for modifier relation and . for complement relation.

LDA is a heuristic-based, linear time, deterministic algorithm which produces dependency linkages not necessarily spanning the entire sentence. LDA can produce a number of partial linkages since it is a driven primarily by the need to satisfy local constraints without being driven to construct a single dependency linkage that spans the entire input. This, in fact, contributes to the robustness of LDA and promises to be a useful tool for parsing sentence fragments that are rampant in speech utterances exemplified by the Switchboard corpus.

Due to the fact that LDA produces a dependency annotated sentence as the output of the parsing process, parsing evaluation metrics that measure the performance of constituent bracketing such as Parseval [Harrison *et al.*, 1991] are unsuitable for evaluating

Pos	Word	Supertag	Slot req.	Pass 1	Pass 2	Dep Links
0	The	β_1	+NP*	3*	–	3*
1	implicit	β_2	+N*	2*		2*
2	interior	β_2	+N*	3*		3*
3	state	α_2	–	–	–	–
4	of	β_1	-NP* +NP.	3* 6.		3* 6.
5	the	β_1	+NP*	6*	–	6*
6	iteration	α_2	–	–	–	–
7	over	β_1	-NP* +NP.	6* 11.		6* 11.
8	the	α_1	+NP*	11*		11*
9	hash	β_3	+N*	10*		10*
10	table	β_3	+N*	11*		11*
11	entries	α_2	–	–	–	–
12	has	α_3	+NP. -NP.		3. 14.	3. 14.
13	dynamic	β_2	+N*	14*		14*
14	extent	α_4	–	–	–	–

Table 4.11: An example sentence with the supertags assigned to each word and dependency links among words

the performance of LDA. Although the supertags contain constituent information, and it is possible to convert a dependency linkage into a constituency based parse, the Parseval metric does not have provision for evaluating unrooted parse trees since the output of LDA can result in disconnected dependency linkages. In Chapter 7, we suggest metrics that can be used to evaluate dependency based parsers and in particular, LDA. We also present performance evaluation results for LDA in that chapter.

4.5.1 Discussion

In this section, we discuss a few points about the supertag and LDA system. It is often a topic of tension to balance the domain specificity versus the portability of a natural language grammar. The more domain specific a grammar is, the better is its performance on that domain but the less portable it is. On the other hand a wide coverage grammar is portable, but needs to be specialized to exploit the idiosyncrasies of the domain in order to be efficient. Given a new domain either the parsing system or the heuristics employed by the grammar have to be retrained. In contrast, the modularising of the supertag and LDA system achieves the balance of domain specificity versus portability quite elegantly.

The domain dependent aspects are factored out and associated with the supertagger while the LDA continues to be the domain independent unit of the system. The probability model of the supertagger models the domain specific idiosyncrasies while the LDA uses a wide-coverage grammar and linguistically motivated domain independent structures to compute the dependency linkage. Each time this system is ported to a new domain, the only module that needs retraining is the supertagger. The data used by the LDA remains unchanged.

It is interesting to observe that the LDA works by first identifying the adjunct structures using the distinction between recursive and non-recursive supertags. Once the arguments of recursive structures are identified, they are considered to be skippable for establishing the arguments of non-recursive structures. The two-pass strategy of the LDA simulates a stack. Further, the LDA relies on the locality of the arguments modulo recursive structures to establish the dependency links. This is made possible due to the two properties of LTAGs, extended domain of locality and factoring of recursion. The effectiveness of the LDA on some examples involving relative clauses, sentence complement and PP attachment are shown below.

Pos	Word	Supertag	Slot req.	Pass 1	Pass 2	Dep Links
0	The	β_1	+NP*	1*		1*
1	rat	α_2	–	–	–	–
2	the	β_2	+NP*	3*		3*
3	cat	α_3	–	–	–	–
4	the	β_3	+NP*	5*		5*
5	dog	α_4	–	–	–	–
6	chased	β_4	-NP. -NP*	5. 3*		5. 3*
7	bit	β_5	-NP. +NP*	3. 1*		3. 1*
8	ate	α_5	-NP. +NP.		1. 10.	1. 10.
9	the	β_6	+NP*	10*		10*
10	cheese	α_6	–	–		–

Table 4.12: An example illustrating the working of LDA on a center-embedded construction.

Pos	Word	Supertag	Slot req.	Pass 1	Pass 2	Dep Links
0	who	α_1	–	–	–	–
1	do	β_1	+VP*	3*		3*
2	you	α_2	–	–	–	–
3	think	β_2	-NP. +S*	2. 5*		2. 5*
4	Bill	α_3	–	–	–	–
5	believes	β_3	-NP. +S*	4. 7*		4. 7*
6	John	α_4	–	–	–	–
7	loves	α_5	-NP. -NP.		6. 0.	6. 0.

Table 4.13: An example illustrating the working of LDA on a sentence with long distance extraction.

4.6 Applicability to other Lexicalized Grammars

Lexicalized grammars associate with each word richer structures (trees in case of LTAGs and categories in case of Combinatory Categorial Grammars (CCGs)) over which the word specifies syntactic and semantic constraints. Hence, every word is associated with a much larger set of more complex structures than in the case where the words are associated with standard parts-of-speech. However, these more complex descriptions allow more complex constraints to be imposed and verified locally on the contexts in which these words appear. This feature of lexicalized grammars can be taken advantage of, to further reduce the disambiguation task of the parser, as shown in supertag disambiguation. Hence supertag disambiguation can be used as a general pre-parsing component of lexicalized grammar parsers.

The degree of distinction between supertag disambiguation and parsing varies, depending on the lexicalized grammar being considered. For both LTAG and CCG, supertag disambiguation serves as a pre-parser filter that effectively weeds out inappropriate elementary structures (trees or categories), given the context of the sentence. It also indicates the dependencies among the elementary structures but not the specific operation to be used to combine the structures or the address at which the operation is to be performed – an *almost parse*. In cases where the supertag sequence for the given input string cannot be combined to form a complete structure, the *almost parse* may indeed be the best one can do.

In case of LTAG, even though no explicit substitutions or adjunctions are shown, the dependencies among LTAG trees uniquely identify the combining operation between the trees and the node at which the operation can be performed is almost always unique[11]. Thus supertag disambiguation is almost parsing for LTAGs. In contrast, the dependencies among the CCG categories do not result in directly identifying the combining operations between the categories since two categories can often be combined in more than one way. Hence for CCG further processing needs to be performed after supertag disambiguation to obtain the complete parse of the sentence.[12]

The dependency model of supertag disambiguation is more closer to parsing in dependency grammar formalism. Dependency parsers establish relationships among words, unlike the phrase-structure parsers which construct a phrase-structure tree spanning the words of the input. Since LTAGs are lexicalized and each elementary tree is associated with at least one lexical item, the supertag disambiguation for LTAG can therefore be viewed as establishing the relationship[13] among words as dependency parsers do. Then the elementary structures that the related words anchor are combined to reconstruct the phrase-structure tree similar to the result of phrase-structure parsers. Thus the interplay of both dependency and phrase-structure grammars can be seen in LTAGs. Rambow and Joshi [Rambow and Joshi, 1993] discuss in greater detail the use of LTAG in relating dependency analyses to phrase-structure analyses and propose a dependency-based parser for a phrase-structure based grammar.

4.7 Summary

In summary, we have presented a new technique that performs the disambiguation of supertags using local information such as lexical preference and local lexical dependencies. This technique, like part-of-speech disambiguation, reduces the disambiguation task that

[11] In some cases, the dependency information between an auxiliary and an elementary tree may be insufficient to uniquely identify the address of adjunction, if the auxiliary tree can adjoin to more than one node in the elementary tree, since the specific attachments are not shown.

[12] An approach similar to LTAG of including a supertag for each syntactic environment a lexical item appeared in was attempted in [Chytil and Karlgren, 1988]. However, this approach was not exploited for robust parsing.

[13] The relational labels between two words in LTAG is associated with the address of the operation between the trees that the words anchor.

needs to be done by the parser. After the disambiguation, we would have effectively completed the parse of the sentence and the parser need 'only' complete the adjunction and substitutions. This method can also serve to parse sentence fragments in cases where the supertag sequence after the disambiguation may not combine to form a single structure. We have implemented this technique of disambiguation using the n-gram models using the probability data collected from LTAG parsed corpus. The similarity between LTAG and Dependency grammars is exploited in the dependency model of supertag disambiguation.

Chapter 5

Exploiting LTAG representation for Explanation-based Learning

There are currently two philosophies for building grammars and parsers. Wide-coverage hand-crafted grammars such as [Grover *et al.*, 1993; Alshawi *et al.*, 1992; Karlsson *et al.*, 1994; XTAG-Group, 1995] are designed to be domain-independent. They are usually based on a particular grammar formalism, such as Context-Free Grammars, Constraint Grammars or Lexicalized Tree-Adjoining Grammars. Statistically induced grammars and parsers [Jelinek *et al.*, 1994; Magerman, 1995; Collins, 1996] on the other hand, are trained on specific domains using a manually annotated corpus of parsed sentences from the given domain.

Aside from the methodological differences in grammar construction, the two approaches differ in the richness of information contained in the output parse structure. Hand-crafted grammars generally provide a far more detailed parse than that output by a statistically induced grammar. Also, the linguistic knowledge which is overt in the rules of hand-crafted grammars is hidden in the statistics derived by probabilistic methods, which means that generalizations are also hidden and the full training process must be repeated for each domain.

Although hand-crafted wide-coverage grammars are portable, they can be made more efficient if it is recognized that language in limited domains is usually well constrained and certain linguistic constructions are more frequent than others. Hence, not all the structures

of the domain-independent grammar nor all the combinations of those structures would be used in limited domains. In this chapter, we view a domain-independent grammar as a repository of portable grammatical structures whose combinations are to be specialized for a given domain. We use Explanation-based Learning mechanism to identify the relevant subset of a hand-crafted general purpose grammar needed to parse in a given domain. We exploit the key properties of Lexicalized Tree-Adjoining Grammars to view parsing in a limited domain as Finite-State Transduction from strings to their dependency structures.

5.1 Explanation-based Learning

Explanation-based Learning (EBL) techniques were originally introduced in the AI literature by [Mitchell *et al.*, 1986; Minton, 1988; van Harmelen and Bundy, 1988]. The main idea behind explanation-based learning is that it is possible to form generalizations from single positive training examples if the system can explain why the example is an instance of the concept under study. The generalizer possesses the knowledge of the concept under study and the rules of the domain for constructing the required explanation. The definition of the EBL problem is summarized in Table 5.1.[1].

The objective of EBL in the AI domain is to keep track of suitably generalized solutions (explanations) to problems solved in the past and to replay those solutions to solve new but somewhat similar problems in the future. Although put in these general terms the approach sounds attractive, it is by no means clear whether it would actually improve the performance of the system, an aspect which is of great interest to us here.

Rayner [1988] was the first to investigate the usefulness of the EBL technique in the context of natural language parsing systems. The correspondence between the EBL terminology and parsing terminology is shown in Table 5.2. In parsing, the concept under study is *grammaticality* of an input. The *grammar* serves as the domain theory. A training *sentence* serves as an instance of the concept. A *parse* of a sentence represents an *explanation* of why the sentence is grammatical according to the grammar. A *derivation tree* represents the explanation structure. Parsing new sentences amounts to finding analogous explanations from the explanations for the training sentences.

[1]The definitions in this table are from [Mitchell *et al.*, 1986]

- Given
  - Goal Concept: A concept definition describing the concept to be learned.
  - Training Example (Instance): An example of the goal concept.
  - Domain Theory: A set of rules and facts to be used in explaining how the training example is an example of the goal concept.
  - Operationality Criterion: A predicate over concept definitions, specifying the form in which the learned concept definition must be expressed.
- Goal:
  - A generalization of the training example that is a sufficient concept definition for the goal concept and that satisfies the operationality criterion.
- Terminology:
  - Generalization of an example: A concept definition that describes a set containing that example.
  - Explanation: A proof of how an instance is an example of the concept.
  - Explanation Structure: The proof tree that leads to the proof that serves as an explanation.

Table 5.1: The explanation-based learning problem

As a special case of EBL, Samuelsson and Rayner [1991] specialize a CFG-based grammar for the ATIS domain by storing chunks of the parse trees present in a treebank of parsed examples. The idea is to reparse the training examples by letting the parse tree drive the rule expansion process and halting the expansion of a specialized rule if the current node meets a 'tree-cutting' criteria. However, the problem of specifying an optimal 'tree-cutting' criteria was not addressed in this work. Samuelsson [1994] used the information-theoretic measure of *entropy* to derive the appropriate sized tree chunks automatically. Neumann [1994] also attempts to specialize a grammar given a training corpus of parsed examples by generalizing the parse for each sentence and storing the generalized phrasal derivations under a suitable index.

Later in the chapter, we will make some additional comments on the relationship

EBL Terminology	→	Parsing Terminology
Concept	→	Grammaticality
Domain Theory	→	Grammar
Instance	→	Sentence
Explanation	→	Parse
Explanation Structure	→	Derivation Structure
Operationality Criterion	→	Elementary structures of the grammar

Table 5.2: Correspondence between EBL and parsing terminology.

between our approach and some of these earlier approaches. In particular, we show that generalization in previous approaches effectively computes the same structures that the LTAG representation provides to begin with.

5.2 Overview of our approach to using EBL

We are pursuing the EBL approach in the context of a wide-coverage grammar development system called XTAG [Doran *et al.*, 1994]. The details of XTAG are discussed in Section 3.2.

The training phase of the EBL process involves generalizing the derivation trees generated by XTAG for the training sentences and storing these generalized parses in the generalized parse database under an index computed from the morphological features of the sentence. The application phase of EBL is shown in the flowchart in Figure 5.1. An index using the morphological features of the words in the input sentence is computed. Using this index, a set of generalized parses is retrieved from the generalized parse database created in the training phase. If the retrieval fails to yield any generalized parse then the input sentence is parsed using the full parser. However, if the retrieval succeeds then the generalized parses are input to the "stapler". The "stapler" is a very impoverished parser that generates all possible attachment sites for all (if any) modifiers besides instantiating the features of nodes of the trees by term unification. The details of the "stapler" are discussed in Chapter 6.

The three key aspects of LTAG, as discussed in Chapter 3.2 are (a) lexicalization, (b) extended domain of locality and (c) factoring of recursion from the domain of dependencies.

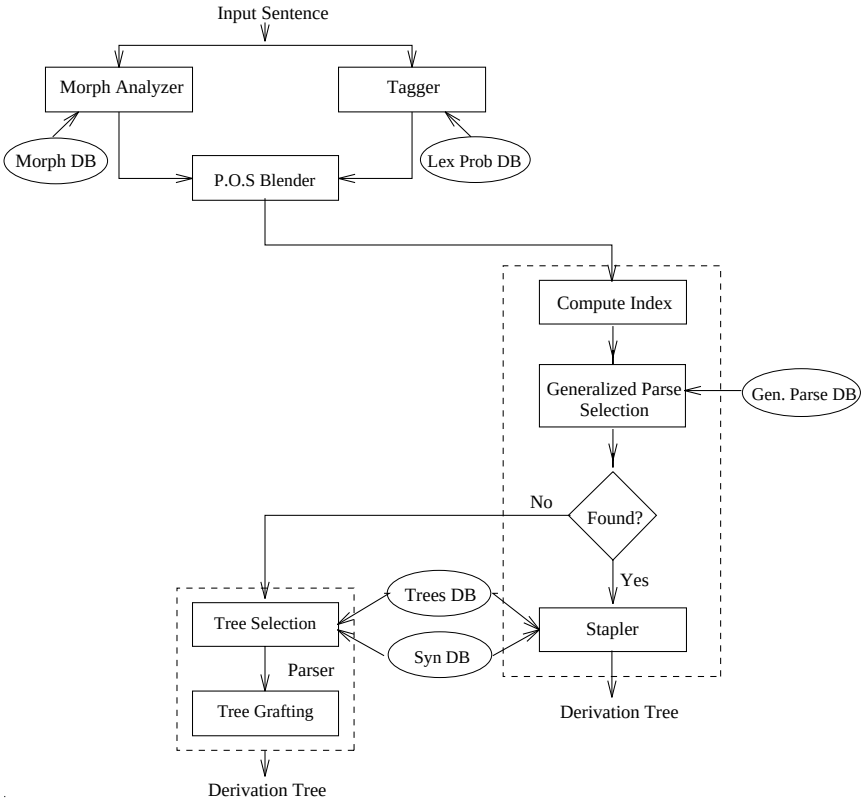


Figure 5.1: Flowchart of the XTAG system with the EBL component

In this chapter, we show that exploiting these key properties of LTAG (a) leads to an *immediate* generalization of parses for the training set of sentences, as discussed in Section 5.3, (b) achieves generalization over recursive substructures of the parses in the training set, as discussed in Section 5.4 and (c) allows for a finite state transducer (FST) representation of the set of generalized parses, as discussed in Section 5.5. In Section 5.6, we present experimental results evaluating the effectiveness of our approach on more than one kind of corpora. We contrast our approach of using EBL technique to previous approaches in Section 5.8.

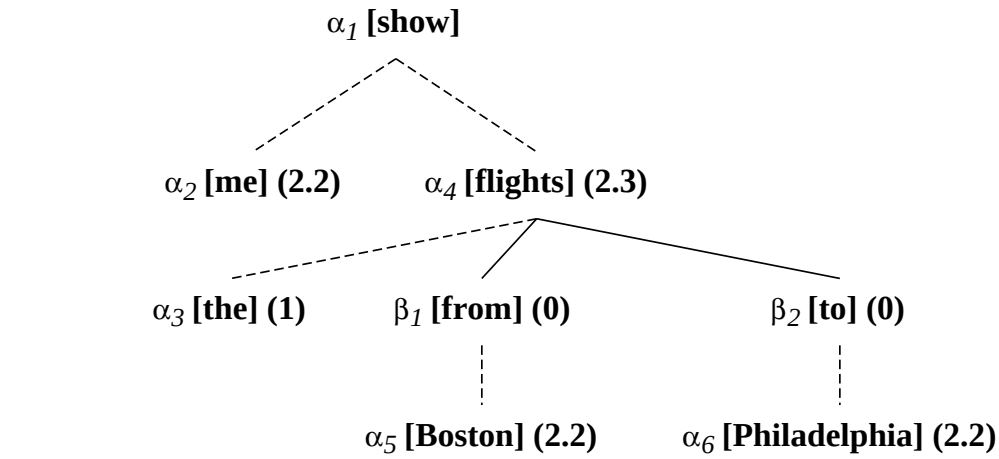
5.3 Feature-generalization

In LTAGs, a derivation tree uniquely identifies a parse for a sentence. A derivation tree is a sequence of elementary trees associated with the lexical items of the sentence along with substitution and adjunction links among the elementary trees. Also, the values for features at each node of every elementary tree is instantiated to a value by the parsing process. Given an LTAG parse, the generalization of the parse is truly *immediate* in that a generalized parse is obtained by (a) uninstantiating the particular lexical items that anchor the individual elementary trees in the parse and (b) uninstantiating the feature values contributed by the morphology of the anchor and the derivation process. This type of generalization is called *feature-generalization* and results in a feature-generalized parse.

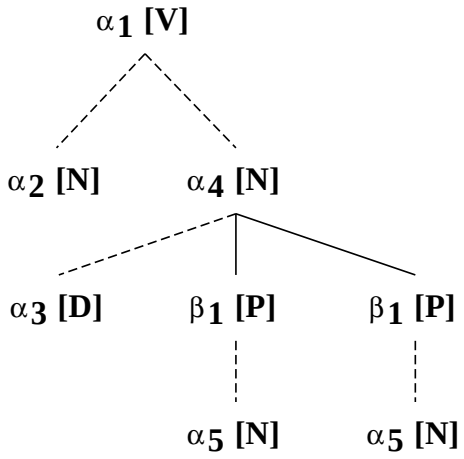
In other EBL approaches based on Context-Free-based grammars, [Rayner, 1988; Samuelsson, 1994] there is no independent notion of a derivation tree. The context-free parse tree corresponds to both the derivation tree and the derived tree. So to generalize a parse, it is necessary to walk up and down the parse tree to determine the appropriate subtrees to generalize on and to suppress the feature values. In contrast, in our approach, the process of generalization is *immediate*, once we have the output of the parser, since the elementary trees anchored by the words of the sentence define the subtrees of the parse for generalization. Replacing the elementary trees with uninstantiated feature values is all that is needed to achieve this generalization. The process of feature generalization is shown in Figure 5.2. After feature-generalization, the trees β_1 and β_2 are no longer distinct so we denote them by β. So also the trees α_5 and α_6 are no longer distinct, so we denote them by α in Figure 5.2(b).

5.3.1 Storing and retrieving feature-generalized parses

The feature-generalized parse of a sentence is stored in a generalized parse database under an index computed from the training sentence. In this case, the part-of-speech (POS) sequence of the training sentence is treated as the index to the feature-generalized parse. The index for the example sentence *show me the flights from Boston to Philadelphia* is shown in Figure 5.2(c). In the application phase, the POS sequence of the input sentence is used to retrieve a generalized parse(s) which is then instantiated to the features of the



(a)



(b)

V N D N P N P N

(c)

Figure 5.2: (a) Derivation structure (b) Feature Generalized Derivation structure (c) Index for the generalized parse for the sentence *show me the flights from Boston to Philadelphia*

sentence.

We have chosen here to index the generalized parse under the POS sequence of the training sentence. However, it must be noted that depending on the degree of abstraction from the lexical items, the generalized parse may be stored under different indices containing varying amounts of lexical specific information. The degree of abstraction strongly influences the amount of training material required for adequate coverage of the test sentences and the number of parses assigned to the test sentences.

This method of retrieving a generalized parse allows for parsing of sentences of the same lengths and the same POS sequence as those in the training corpus. However, in our approach there is another generalization that falls out of the LTAG representation which allows for flexible matching of the index to allow the system to parse sentences that are not necessarily of the same length as some sentence in the training corpus.

5.4 Recursive-generalization

Auxiliary trees in LTAG represent recursive structures. So if there is an auxiliary tree that is used in an LTAG parse, then that tree with the trees for its arguments can be repeated any number of times, or possibly omitted altogether, to get parses of sentences that differ from the sentences of the training corpus only in the number of modifiers. This type of generalization can be called *recursive-generalization*. Figure 5.3 illustrates the process of recursive generalization. The two Kleene star regular expressions in Figure 5.3(b) can be merged into one and the resulting recursive generalized parse is shown in Figure 5.3(c).

Recursive generalization in other approaches to EBL first requires identifying the appropriate subderivations that are recursive. This is achieved by cutting up the parse tree, as is done in [Rayner, 1988] or by taking arbitrary cuts of the parse tree as is done in [Neumann, 1994]. These operations effectively rediscover the set of auxiliary trees that the LTAG provides to begin with.

5.4.1 Storing and retrieving recursive-generalized parses

Due to recursive-generalization, the POS sequence covered by the auxiliary tree and its arguments can be repeated zero or more times. As a result, the index of a generalized

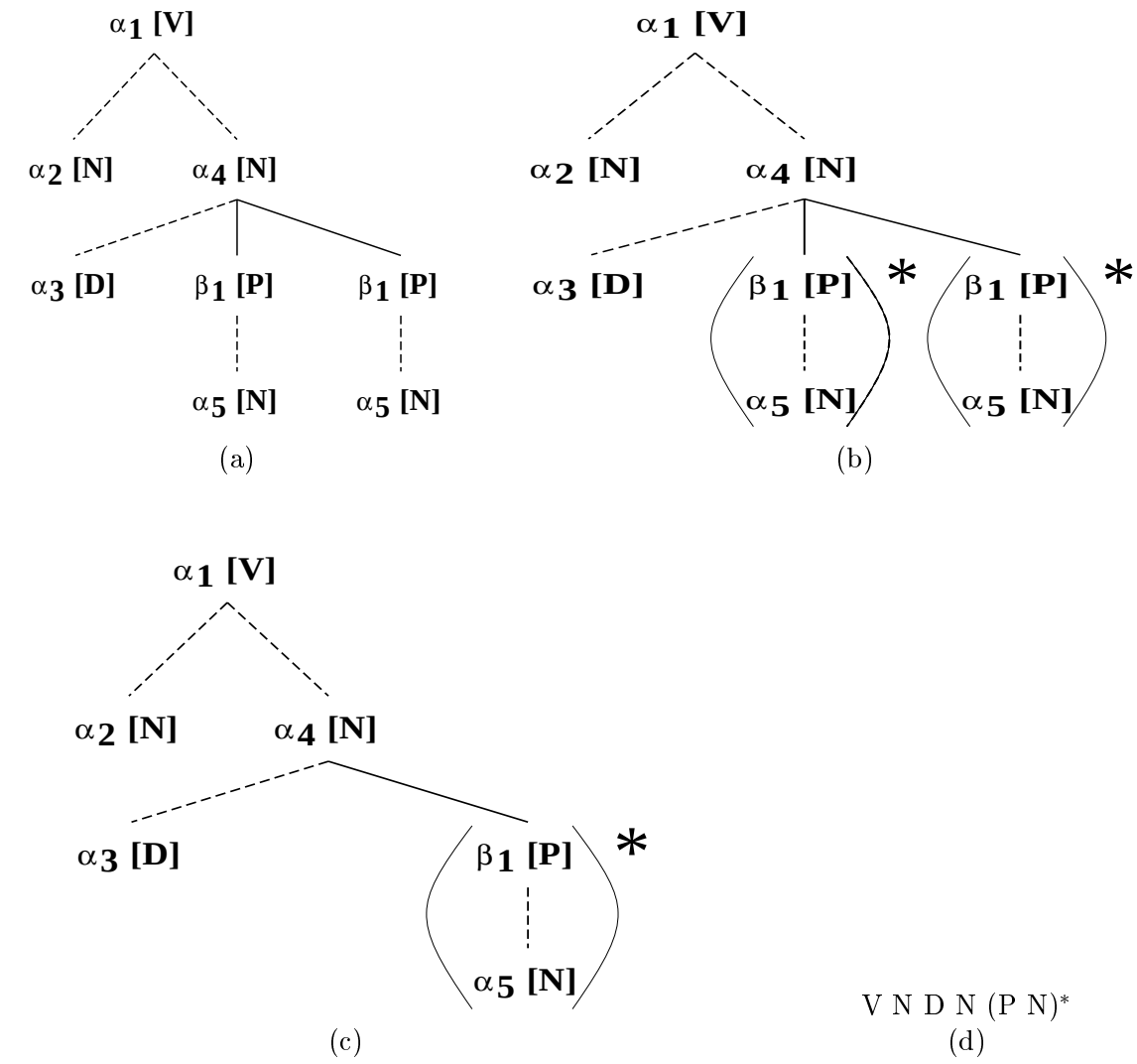


Figure 5.3: (a) Feature-generalized derivation tree (b) Recursive-generalized derivation tree (c) Recursive-generalized derivation tree with the two Kleene-stars collapsed into one (d) Index for the generalized parse for the sentence *show me the flights from Boston to Philadelphia.*

parse of a sentence with modifiers is no longer a string but a regular expression pattern on the POS sequence and retrieval of a generalized parse involves regular expression pattern matching on the indices. The index for the example sentence is shown in Figure 5.3(d), since the prepositions in the parse of this sentence would anchor auxiliary trees.

The most efficient method of performing regular expression pattern matching is to construct a *finite state machine* for each of the stored patterns and then traverse the machine using the given test pattern. If the machine reaches the final state, then the test pattern matches one of the stored patterns.

Given that the index of a test sentence matches with one of the indices from the training phase, the generalized parse retrieved will be a parse of the test sentence, modulo the modifiers. For example, if the test sentence, tagged appropriately, is

(5.1) Show/V me/N the/D flights/N from/P Boston/N to/P Philadelphia/N on/P Monday/N.

then, although the index of the test sentence matches the index of the training sentence, the generalized parse retrieved needs to be augmented to accommodate the additional modifier *on/P Monday/N*.

To accommodate the additional modifiers and their arguments that may be present in the test sentences, we need to provide a mechanism that assigns the additional modifiers and their arguments the following:

1. The elementary trees that they anchor and
2. The substitution and adjunction links to the trees they substitute or adjoin into.

We assume that the additional modifiers along with their arguments would be assigned the same elementary trees and the same substitution and adjunction links as were assigned to the modifier and its arguments of the training example. Hence the Kleene-star around the subderivation rooted by the preposition in Figure 5.3(b). This, of course, means that we may not get all the possible attachments of the modifiers at this time. (but see the discussion of the "stapler" in Chapter 6)

The recursive-generalized parse shown in Figure 5.3(c) can be stored in the generalized parse database indexed on the regular expression shown in Figure 5.3(d). However,

in the application phase, matching the index requires keeping track of the number of instantiation of the Kleene-star in the regular expression index so as to instantiate the corresponding modifier structure in the derivation tree. In contrast, the Finite-State transducer representation overcomes this problem by combining the generalized parse and the POS sequence (regular expression) that it is indexed by in a uniform representation.

5.5 Finite-State Transducer Representation

As mentioned earlier, a *finite state machine* (FSM) provides the most efficient method of performing regular expression pattern matching over the space of index patterns. An index in this representation is a path in the FSM. The generalized parse associated with an index is stored along the path that represents the index. The idea is to annotate each arc of the FSM with the elementary tree associated with that POS and also indicate which elementary tree it would be adjoined or substituted into. This results in a *Finite State Transducer* (FST) representation, illustrated by the example below. Consider the example sentence (5.2) with the generalized derivation tree in Figure 5.4.

(5.2) show me the flights from Boston to Philadelphia.

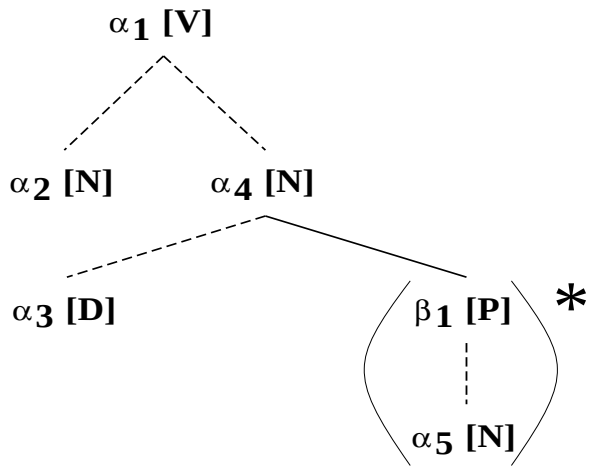


Figure 5.4: Generalized derivation tree for the sentence: *show me the flights from Boston to Philadelphia*

Item		Description
this_tree	:	the elementary tree that the word anchors
head_word	:	the word on which the current word is dependent on; "–" if the current word does not depend on any other word.
head_tree	:	the tree anchored by the head word; "–" if the current word does not depend on any other word.
number	:	a signed number that indicates the direction and the ordinal position of the particular head elementary tree from the position of the current word *OR*
	:	an unsigned number that indicates the gorn-address (i.e., the node address) in the derivation tree to which the word attaches *OR*
	:	"–" if the current word does not depend on any other word.

Table 5.3: Description of the components in the tuple representation associated with each word.

An alternate representation of the derivation tree that is similar to the *dependency representation*, is to associate with each word a tuple (this_tree, head_word, head_tree, number) where the description for the components of the tuple is given in Table 5.3.

Following this notation, the derivation tree in Figure 5.4 is represented as in (5.3).

(5.3) V/(α_1, –, –, –) N/(α_2, V,α_1,-1) D/(α_3, N, α_4,+1) N/(α_4,V, α_1, -1)
(P/(β, N, α_4, 2) N/(α, P, β, -1))*

which can be seen as a path in an FST as in Figure 5.5.

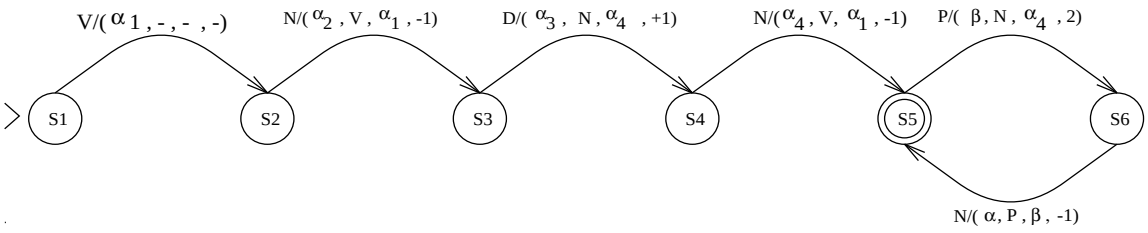


Figure 5.5: Finite State Transducer Representation for the sentences: *show me the flights from Boston to Philadelphia, show me the flights from Boston to Philadelphia on Monday, ...*

This FST representation is possible due to the lexicalized nature and the extended domain of locality of elementary trees because of which lexical dependencies are localized

to with in a single elementary structure. This representation makes a distinction between head-modifier dependencies and head-complement dependencies. For a head-complement dependency relation, the number in the tuple associated with a word is a signed number that indicates the ordinal position of its head in the input string. For a head-modifier dependency relation, the number in the tuple associated with a word is an unsigned number that represents the tree-address (Gorn address) of its head in the derivation tree.

5.5.1 Extending the Encoding for Clausal complements

Long distance dependencies in LTAGs arise out of adjunction of predicative auxiliary trees into other elementary trees. The method of recursive generalization discussed in Chapter 5 does not distinguish predicative auxiliary trees from modifier auxiliary trees. Consider the sentence 5.4 as a training example.

(5.4) who did you say flies from Boston to Washington

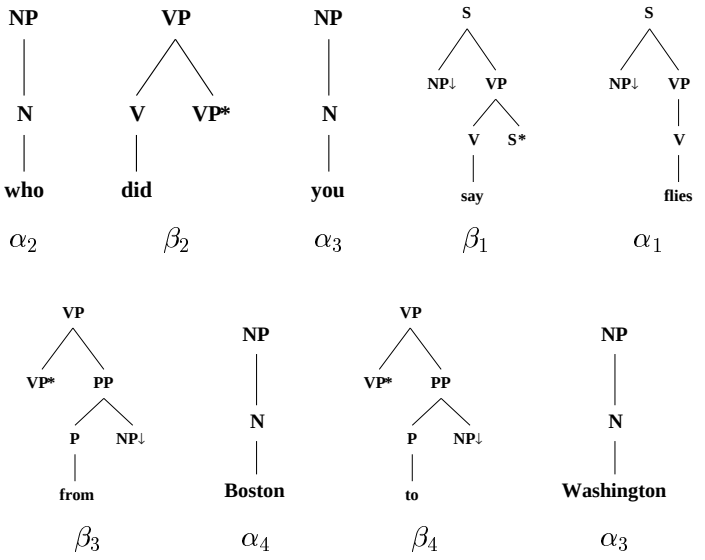


Figure 5.6: The elementary trees for the sentence *who did you say flies from Boston to Washington*

The elementary trees for the words of this sentence are shown in Figure 5.6 and the corresponding derivation tree and generalized versions are shown in Figure 5.7. The

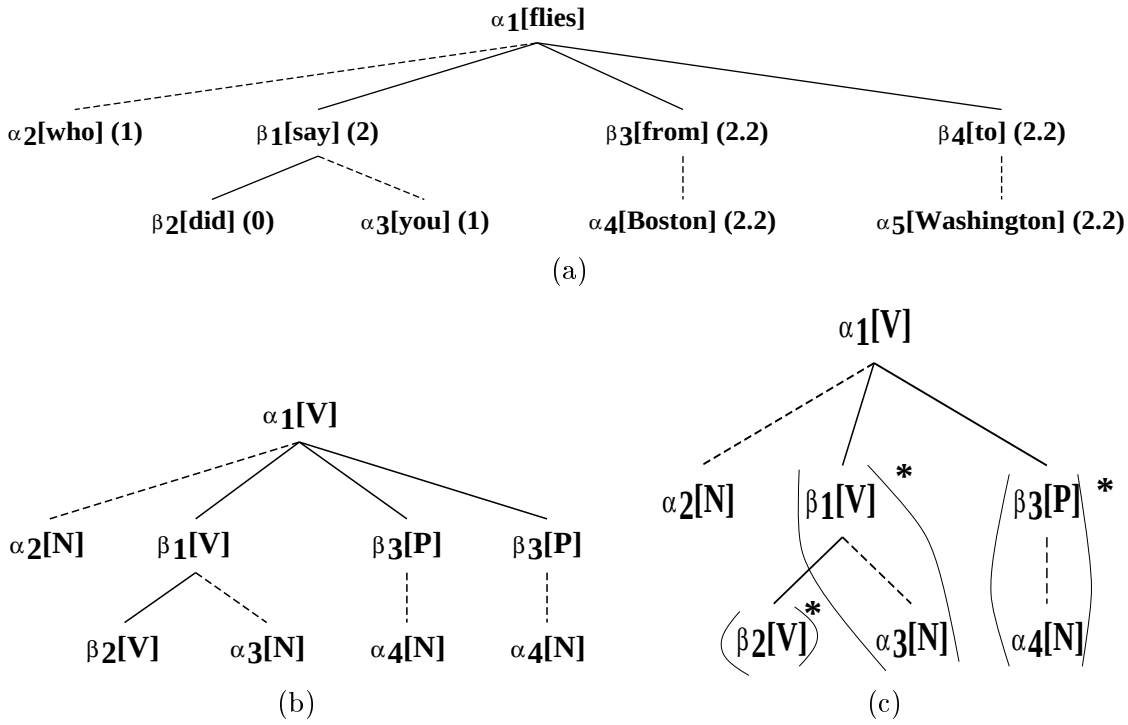


Figure 5.7: (a) Derivation structure (b) Feature Generalized Derivation structure and (c) Recursive Generalized Derivation structure for the sentence *who did you say flies from Boston to Washington*

FST representation of this derivation structure is shown in Figure 5.8. Due to recursive-generalization, generalized derivations for the following sentences can be obtained as well.

(5.5) Long Distance dependency: who did you think I said flies from Boston to Washington.

(5.6) Subject Extraction: who flies from Boston to Washington.

However, in the example for the long distance dependency, although a generalized derivation tree can be assembled by traversing the FST, the dependency relation between *think* and *flies* is incorrect. This is because the recursive generalization assumes that the successive instances of recursion attach to the same head as the initial one. To remedy this situation, the following observation is in order.

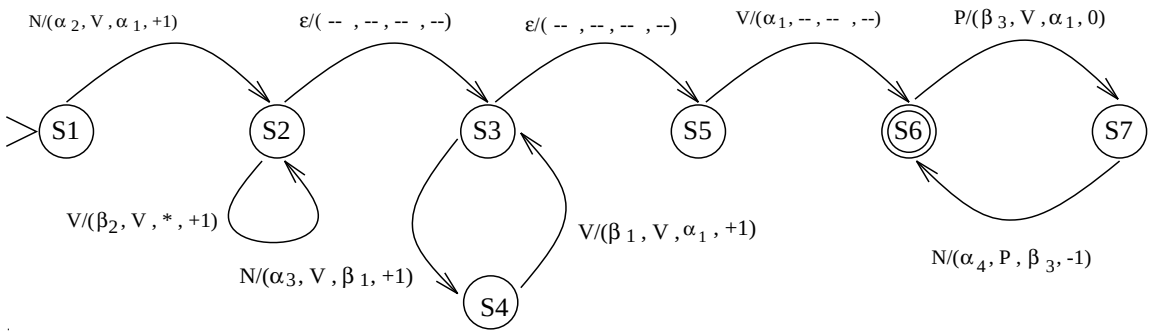


Figure 5.8: FST for the sentence *who did you say flies from Boston to Washington*

5.5.2 Types of auxiliary trees

Auxiliary trees in LTAG have been distinguished as modifier auxiliary trees and predicative auxiliary trees. While the modifier auxiliary trees modify the constituent they adjoin on to, the predicative auxiliary trees subcategorize for the constituent they adjoin on to. Examples of the modifier auxiliary trees and predicative auxiliary trees are shown in Figure 5.9.

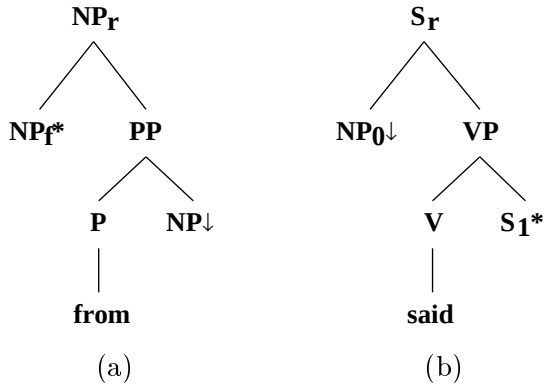


Figure 5.9: (a): Modifier Auxiliary Tree, (b): Predicative Auxiliary Tree

The additional auxiliary trees that result from recursive generalization display different adjunction behaviour depending on the type of the auxiliary tree as shown in sentences

Item		Description
this_tree	:	the elementary tree that the word anchors
head_word	:	the word on which the current word is dependent on; "–" if the current word does not depend on any other word.
head_tree	:	the tree anchored by the head word;
	:	"T_{clause}" if the type of the tree anchored by the head word is a clausal tree
	:	"–" if the current word does not depend on any other word;
	:	"$*$" if the tree does not matter.
number	:	a signed number that indicates the direction and the ordinal position of the particular head elementary tree from the position of the current word *OR*
	:	an unsigned number that indicates the gorn-address (i.e., the node address) in the derivation tree to which the word attaches *OR*
	:	"–" if the current word does not depend on any other word.

Table 5.4: Description of the components in the tuple representation associated with each word.

(5.7) and (5.8). The successive repetitions of the modifier in (5.7) can modify the same head (*flies*) as did the modifiers in the training example. However, successive repetition of the predicative auxiliary trees take as complement the clause following it. Thus, in (5.8), *think* adjoins on to *said* and not to *flies*.

(5.7) who did you say flies from Boston to Washington on Monday.

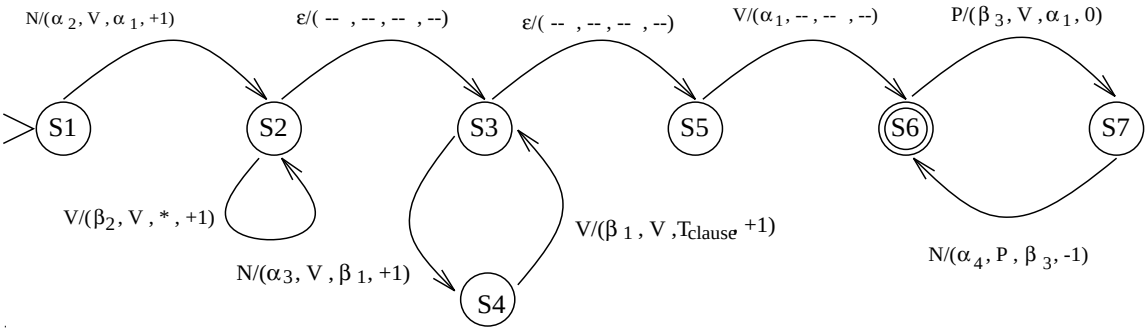
(5.8) who did you think I said flies from Boston to Washington.

Based on the preceding observations we assume that the additional auxiliary trees along with their arguments would be assigned elementary trees along with substitution and adjunction links as follows:

- The additional auxiliary trees along with their arguments would be assigned the same generalized elementary trees as they were assigned in the training example.
- The arguments of the auxiliary trees will be assigned the same substitution and adjunction links as they were assigned the training example.

N/(α_2, V, α_1, +1) V/(β_2, V,*,+1) (N/(α_3, V, β_1,+1) V/(β_1,V,T$_{clause}$,+1))*
V/(α_1, –, –, –) (P/(β_3, V, α_1, 0) N/(α_4, P, β_3, -1))*

(a)



(b)

Figure 5.10: Finite State Transducer Representation for the sentences: *who did you say flies from Boston to Washington, who did you say flies from Boston to Washington on Monday, who did you think I said flies from Boston to Washington, ...*

- If the auxiliary tree is a modifier auxiliary tree then it is assumed that it modifies the same head as it did in the training example.

- If the auxiliary tree is a predicative auxiliary tree then it is assumed to adjoin to the immediately following clause, in terms of string position.

Based on these assumptions, the derivation tree in Figure 5.7(c) is encoded by associating each word with a tuple (this_tree, head_word, head_tree, number) where the description for the components of the tuple is given in Table 5.4. With this extended encoding and by traversing the FST shown in Figure 5.10, the sentence 5.5 is assigned a derivation tree with the correct dependency links.

It is interesting to note that the complement linkages of a tree are specified in terms string positions irrespective of the nature of the tree (recursive or non-recursive) while the adjunct linkages are specified in terms of tree positions.

5.6 Experiments and Results

We now present experimental results to show the effectiveness of our approach. Experiments (a) through (c), are intended to measure the coverage of the FST representation of the parses of sentences from a range of corpora, ATIS, IBM-Manual and Alvey corpus. The results of these experiments provide a measure of repetitiveness of patterns as described in this chapter, at the sentence level, in each of these corpora.

Experiment (a): The details of the experiment with the ATIS corpus are as follows. A total of 400 sentences, average length of 10 words per sentence, which had been completely parsed by the XTAG system was randomly divided into two sets, a training set of 300 sentences and a test set of 100 sentences, using a random number generator. For each of the training sentences, the correct parse was generalized and stored as a path in a FST. The FST was then minimized. The minimized FST was tested for retrieval of a generalized parse for each of the test sentences that were pretagged with the correct POS sequence . When a match is found, the output of the EBL component is a generalized parse that associates with each word the elementary tree that it anchors and the elementary tree into which it adjoins or substitutes into – an *almost parse*.[2]

Experiment (b) and (c): Similar experiments were conducted using the IBM-manual corpus and a set of noun definitions from the LDOCE dictionary that were used as the Alvey test set [Carroll, 1993].

Corpus	Size of Training set	# of States	% Recall	Avg. # of parses	Avg. Response Time (secs)
ATIS	300	1162	80%	2	0.35 sec/sent
IBM	1500	9078	47%	4	3.50 sec/sent
Alvey	80	500	50%	2	0.20 sec/NP

Table 5.5: Recall percentage, Average number of parses, Response times and Size of the FST for various corpora

Results of these experiments are summarized in Table 5.5. The size of the FST obtained for each of the corpora, the recall percentage, the average number of generalized parses and the traversal time per input are shown in this table. The recall percentage of the FST

[2]See Chapter 4 for the role of almost parse in supertag disambiguation.

is the number of inputs that were assigned a correct generalized parse. Owing to the high degree of repetitive patterns in the ATIS domain, the size of the FST and the resulting recall are better than that for IBM manual sentences.

In Chapter 6, we discuss the performance improvement gained by incorporating the EBL component into the XTAG system.

5.7 Phrasal EBL and Weighted FST

Our approach to using EBL is ideally suited for domains where the patterns are repetitive only at the sentence level. However, this method can be extended for domains where the patterns are repetitive at the phrasal level as well. The idea would be to identify the repetitive phrasal subtrees from the derivation trees and create an FST as described in the previous sections for each of those subtrees, such as an FST for NPs, FST for PPs and so on. In the test phase these FSTs are applied as a sequence of transductions with each transduction resulting in the type of the phrase that is recognized along with the phrase internal substitution and adjunction links. The head of the phrase (word that does not depend on any other word in the phrase) identified by the current transduction is used in the next transduction for linking with other words.

Another extension to our approach is to associate weights to the arcs of the FST so as to produce ranked generalized derivations. Traversing the FST created during the training phase could result in a set of generalized derivation structures. These derivations can be ranked according to their likelihood if each arc of the FST is weighted by the probability of traversing that arc. This can be achieved by simply counting the number of times each arc is traversed during the training phase and normalizing the arc weights so that the weights of all the outgoing arcs at each node sum to one. In the test phase, a path probability is computed as the product of the probabilities of each arc that is a component arc of that path.

5.8 Discussion

In this section, we will compare and contrast our approach of using EBL to the previous approaches and highlight the representational richness provided by LTAGs.

In contrast to our approach, Samuelsson and Rayner [1991] specialize a CFG-based grammar for the ATIS domain by 'cutting up' the parse tree and identifying the most frequent subderivations using treebank of parsed examples. They then derive a specialized grammar in which the pre-terminal yield of these frequent subderivations are compiled out. The number of derivation steps in the specialized grammar is relatively smaller than in the original grammar and hence the parsing process is relatively faster.

Rayner and Samuelsson use the following two rules to cut up the parse tree and recover the subderivations. These rules seem sufficient in the ATIS domain but are not exhaustive in general for other domains.

- If a node is labeled as an NP and is not dominated by any other node labeled NP, then cut above it.

- If a node is labeled as an NP and does not dominate any other node labeled NP, then cut above it.

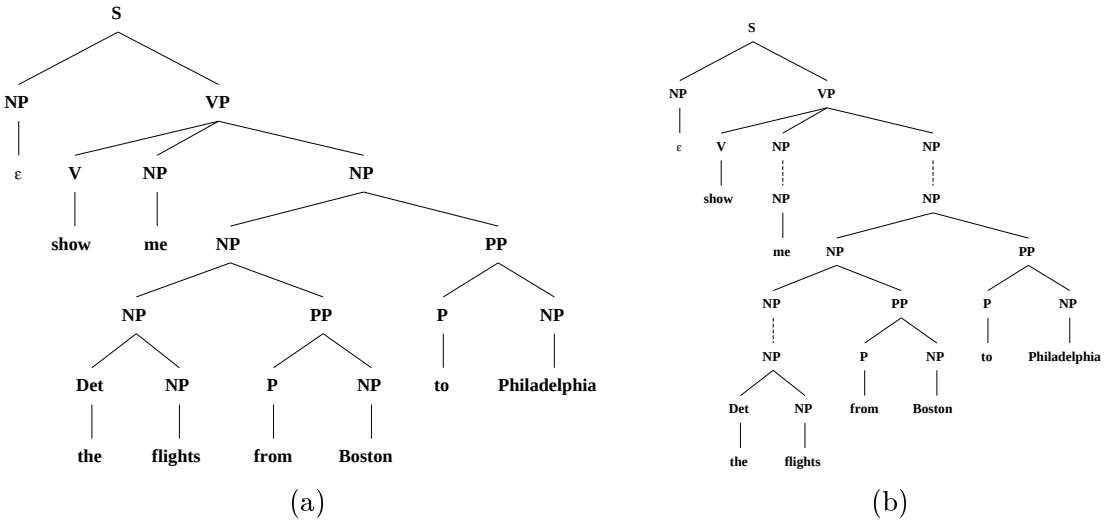


Figure 5.11: (a) Parse tree and (b) Tree-chunks for the sentence *show me the flights from Boston to Philadelphia*

Applying these two rules to the parse of the example sentence *show me the flights from Boston to Philadelphia* results in the three tree chunks shown in Figure 5.11. The dotted line in the Figure 5.11 shows the nodes at which the parse tree is cut. Notice that the resulting trees are of the following three types each corresponding to a LTAG elementary tree.

- A tree rooted at S with lexical items and NP substitution nodes on the frontier, corresponding to an LTAG initial tree with NP arguments.

- A tree rooted at NP with lexical items on the frontier, corresponding to an LTAG initial tree rooted at the NP node.

- A tree to express recursion, that is rooted at NP with NP nodes and lexical items on the frontier corresponding to auxiliary trees.[3]

As seen in this example, the tree-cutting rules effectively rediscover the LTAG elementary tree structures. Samuelsson [1994] in continuation with his previous work uses entropy as a metric to identify locations to cut up the parse tree. This method can identify certain subderivations that may be missed out in [Samuelsson and Rayner, 1991].

Neumann ([1994]), in an attempt to achieve recursive-generalization, allows arbitrary cuts through the parse tree to serve as indices thus allowing non-terminals, in particular NPs to be part of the index. As a result, matching the index in the application phase becomes much more complex and requires scanning, prediction and completion steps. In contrast, recursive-generalization in our approach is triggered by lexical items that anchor auxiliary trees and hence is quite immediate.

Another aspect that is not addressed in previous work is the issue of efficient indexing and retrieval of generalized parses. The FST representation used in our approach provides an optimal method for indexing and retrieving the generalized parses. Lexicalization and Extended Domain of Locality of LTAGs serve as the key properties for this efficient representation.

[3]The auxiliary trees may not represent minimal recursive structure, as in this case.

5.9 Summary

In this chapter, we have shown some novel applications of Explanation-Based Learning techniques to parsing with Lexicalized Tree-Adjoining Grammars (LTAG) in limited domains. The three key aspects of LTAG (a) lexicalization, (b) extended domain of locality and (c) factoring of recursion from the domain of dependencies (1) lead to an *immediate* generalization of parses for the training set of sentences, (2) achieve generalization over recursive substructures of the parses, and (3) allow for a finite state transducer (FST) representation of the set of generalized parses. We have shown a methodology to speed-up the performance of a domain-independent, wide-coverage, hand-crafted grammar when used in limited domains, without sacrificing the portability of the grammar. In the next chapter, we present a novel device called *Stapler* that in conjunction with the EBL component functions as a lightweight parser.

Chapter 6

Stapler

A parser is a device that checks the well-formedness of an input string given a set of elementary structures (a grammar) and the rules for combining them. The output of a parser is a parse that serves as a proof of the well-formedness of the input string given the grammar. A parse is the result of searching, selecting and combining appropriate elementary structures from the set of possible structures defined in the grammar, so as to span the input string. This fact is represented by the *derivation structure* for the input string. This aspect of searching through the space of elementary structures of the grammar to determine the appropriate structures to combine is the single most time-expensive component of the parsing process. The search space of elementary structures grows rapidly as the degree of ambiguity of words in the input string increases. This fact is further compounded in wide-coverage grammars where a variety of elementary structures are included in the grammar for the range of linguistic constructions. Due to this enormous size of the search space, parsers for wide-coverage grammars are excruciatingly slow and often perform at speeds that make them impractical.

Lexicalized grammars, however, provide an opportunity to reduce the ambiguity in the elementary structures even before parsing begins. Each lexical item in an input string contributes one elementary structure from the collection of structures it anchors, towards a parse for the string. If this task of selecting the correct elementary structure to contribute to the parse is done before parsing begins, then the parser's task is reduced to combining the selected structures using the operations of the formalism. In the context of LTAGs,

Supertag Disambiguation (as discussed in Chapter 4) and EBL-technique for parsing (as discussed in Chapter 5) are two approaches for selecting the suitable elementary trees even before parsing begins. In this chapter, we introduce a novel device called *Stapler* – an impoverished parser, whose only task is to combine the selected elementary trees to form a parse.

The following is the layout of this chapter. In Section 6.1 we discuss the nature of the input to the Stapler. The various tasks performed by a Stapler to complete a parse is discussed in Section 6.2. Some experimental results demonstrating the speed-up obtained by using a Stapler in contrast to a conventional parser are discussed in Section 6.3.

6.1 Input to the Stapler: *Almost Parse*

A stapler is a device used in conjunction with a supertagger or an EBL-lookup system. The output of a supertagger or an EBL-lookup system is one or more supertag sequences, each sequence annotated with dependency information – an *almost parse* or a generalized derivation tree. Figure 6.1(a) shows the generalized derivation trees resulting from the supertagger as discussed in Chapter 4 and Figure 6.1(b) shows the generalized derivation tree from EBL-look up as discussed in Chapter 5.[1] The following are the characteristics of an *almost parse*.

- Each word of the input is associated with a unique supertag.
- The supertags are linked by directed arcs originating at a dependent supertag and terminating at a head supertag.[2]

The stapler assembles the supertags to obtain a parse(s) of the input using the dependency information present in the *almost parse*.

[1] The lexical items are shown as instantiating the supertags in Figure 6.1. However, the features on the supertags are not yet instantiated with the morphological features of the lexical items that anchor them.

[2] A supertag γ is dependent on δ if γ substitutes or adjoins into δ. Supertag δ is called the head supertag and γ is called the dependent supertag.

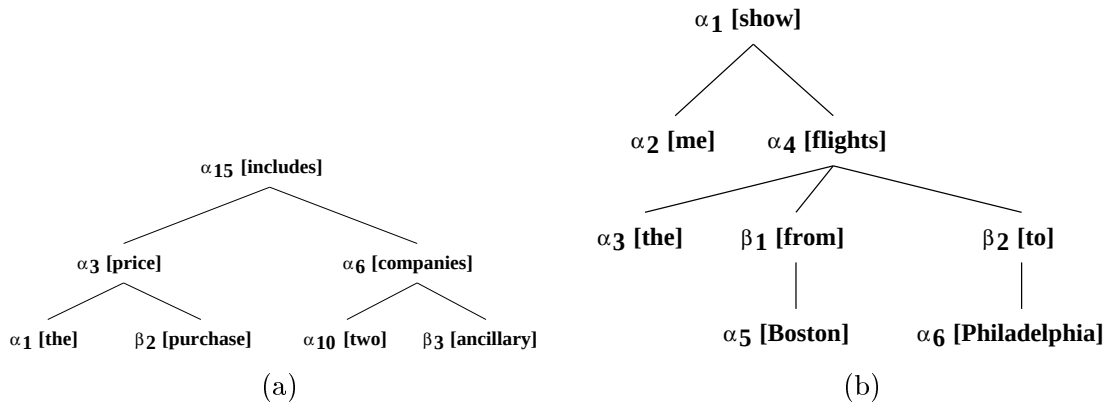


Figure 6.1: (a): Output of Supertagger for the sentence *The purchase price includes two ancillary companies* (b): Output of EBL-lookup for the sentence *Show me the flights from Boston to Philadelphia*

6.2 Stapler's Tasks

As can be seen from Figure 6.1, the *almost parse* needs to be augmented with the following information to be a complete parse. The stapler performs these tasks.

1. Identify the nature of operation: Dependency links are to be distinguished as either substitution or adjunction links.

2. Modifier Attachment: Alternate sites of attachments for modifiers needs to be computed.

3. Address of Operation: Node addresses are to be assigned to dependency links to indicate the location of operation.

4. Feature Instantiation: The values of the features on the nodes of the supertags have to be instantiated by a process of unification.

In the following sections we discuss each of the subtasks in greater detail.

6.2.1 Identify the nature of operation

Given the supertag sequence and the dependency links among them the task of this module is to label the dependency links as either substitution links or adjunction links. This task

is extremely straightforward since the type of the supertags (initial or auxiliary) that a dependency link connects determines the nature of the link. Auxiliary trees are adjoined while initial trees are substituted into other elementary trees.

6.2.2 Modifier Attachment

One of the generalizations made in EBL-lookup is *recursive-generalization*. It states that if an auxiliary supertag is used in an LTAG parse, then that supertag with the supertags for its arguments can be repeated any number of times, or possibly omitted altogether, to get parses of sentences that differ from the sentences of the training corpus in the number of modifiers only. However, an assumption is made that the additional instances of modifiers modify the same head as the training instance modified. Due to this assumption, EBL-lookup may not provide all possible attachments of the modifiers for a given input string. This is illustrated in Figure 6.2. Figure 6.2(a) is the generalized parse resulting from EBL-lookup for the sentence *Give me the cost of tickets on Delta*. Figure 6.2(b) shows the derivation tree instantiated to the words in the sentence. However, it can be clearly seen that the intended derivation tree is the one shown in Figure 6.2(c).

The task of this module is to generate the alternate sites of attachments for the modifiers in the input string, given an *almost parse* for that string. The algorithm proceeds as follows.

To produce alternate attachment sites for the modifiers, the sub-derivation rooted at the modifier auxiliary tree can be seen as sliding up and down the derivation tree. Each modifier auxiliary tree can slide up from its parent to its grandparent (if there is one) or can slide down from its parent to its sibling. The sliding process, however, has to be constrained so as to avoid crossing dependencies. To do so, each supertag that is associated with a modifier is distinguished as either left or right-anchor supertag based on the direction of the anchor node of the supertag to the foot node of the supertag. To avoid crossing-dependencies, left-anchor supertags can slide up, provided they do not have a right sibling in the derivation tree and right-anchor supertags can slide up, provided they do not have a left sibling in the derivation tree.

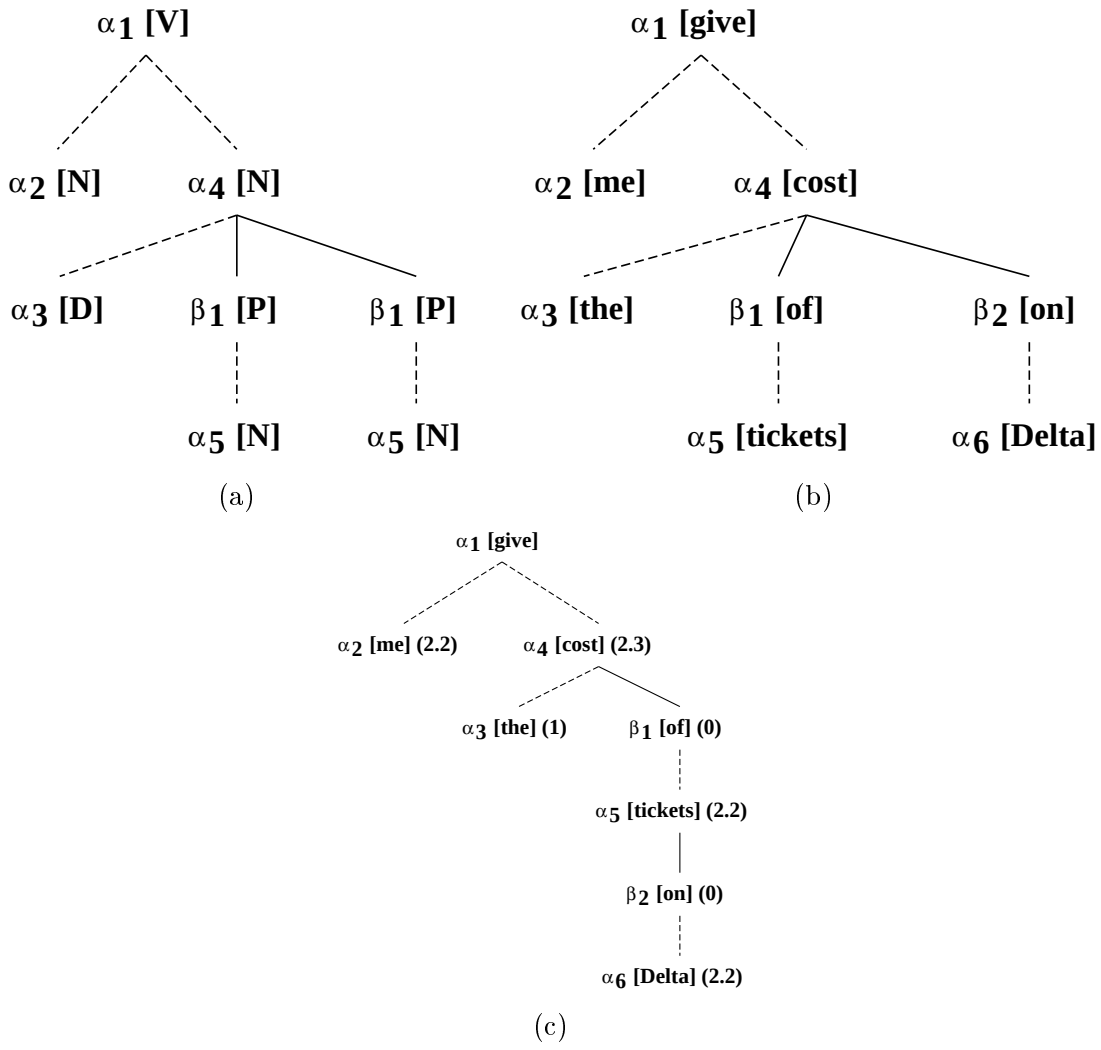


Figure 6.2: (a) Output of EBL-lookup (b) Instantiated derivation tree (c) Intended derivation tree for the sentence *Give me the cost of tickets on Delta*

6.2.3 Assigning the addresses to the links

The next step towards completing the *almost parse* is to assign node addresses to the dependency links to indicate the site of substitution or adjunction. To do this, we need to represent the internal structure of supertags. A supertag is represented by the list of nodes that appear in it – root node, anchor node, foot node, list of substitution nodes and list of internal nodes that do not have Null Adjunction constraints. Each node carries its label, its address and its features. Figure 6.3 illustrates this pictorially.

The process of assigning the addresses to the dependency links proceeds in two steps. In the first step, the substitution links are assigned node addresses and in the second step, the adjunction links are assigned node addresses. This allows for the possibility of exploiting more constraints when assigning the node addresses to adjunction links. The algorithm proceeds as follows.

For each substitution link, the possible substitution sites in the head supertag that match the root label of the dependent supertag are collected. If there is only one such node then that is the site for substitution. If there is more than one such node, we select the one that respects the linear order of the anchors of the head and dependent supertags as given in the input string.

For each adjunction link, the non-NA-internal nodes of the head supertag whose node labels match the root label of the dependent supertag are collected. If there is only one such node then that is the site for adjunction. If there is more than one such node, we select one, based on the linear order of the anchors of the head and dependent supertags given in the input string.

Once a unique node (internal or substitution) has been identified, the feature values are checked for compatibility according to the substitution and adjunction schema. In the next section, we discuss an efficient feature structure implementation system that exploits some of the properties of LTAGs.

6.2.4 Feature Structures and Unification

LTAGs localize the argument structure of lexical items and distinguish recursive structures from the domain of local dependencies. Owing to these properties, features in LTAGs have

the following properties.

- co-indexing among features is limited to those features that appear within a supertag.
- the feature values are finite and non-recursive.

These properties of features in LTAGs are exploited in this efficient implementation of the feature structures that avoids full-blown graph unification. Also owing to the finite valued features, it is possible to provide feature negation, feature disjunction and feature overriding mechanisms, without any computational calamities.

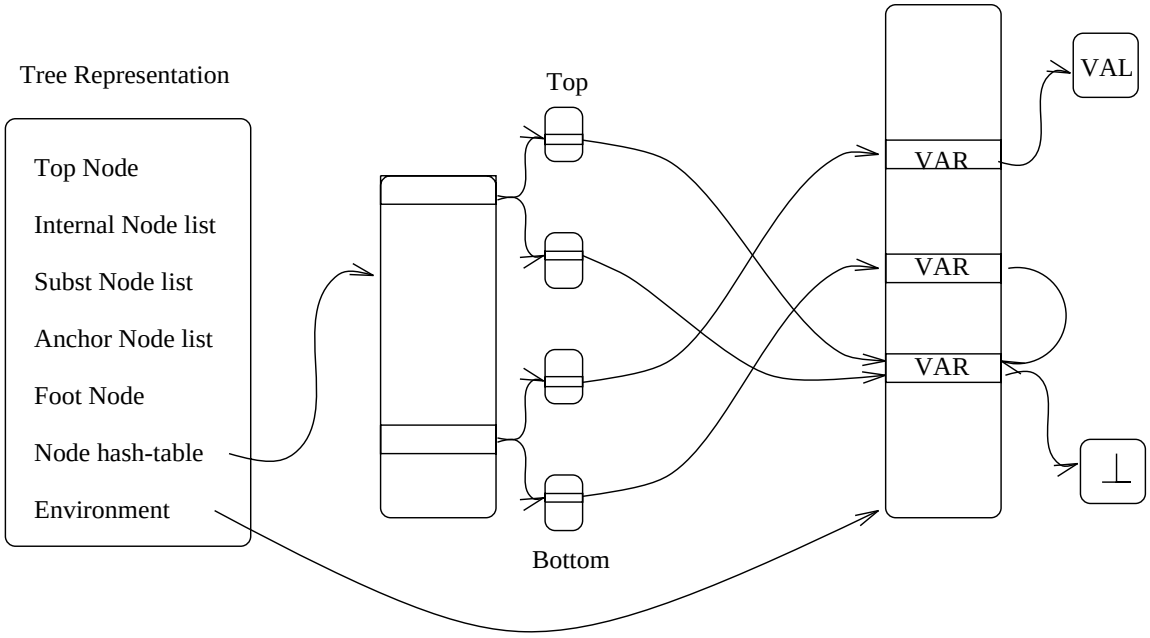


Figure 6.3: Data Structure for an LTAG elementary tree

6.2.5 Data Structure

The data structure for a supertag with feature information is shown in Figure 6.3. The various nodes of the supertag along with their addresses are represented in the fields of the *Supertag Structure*. The *Node Hash Table* is a hash table indexed by the node labels that have features associated with them. Each hash entry in the *Node Hash Table* is a pair of attribute tables corresponding to the top and bottom feature structures at that node. Each table is indexed by the attributes present in that feature structure. Each attribute points

to a meta-variable *VAR* which in turn is either bound to a value VAL, or is unbound ($\perp$), or is bound to another meta-variable. The bindings of the meta-variables to their values are stored in the *Environment* hash table which is indexed by the meta-variables. There is one *Environment* hash table for each supertag. The meta-variables are used to encode facts of unification/coindexation equations among attributes. Two coindexed attributes are bound to the same meta-variable.

In this representation, a feature structure associated with a supertag is a pair of hash tables – the attribute hash table consisting of the attributes appearing in that supertag and the environment hash table consisting of values of the features in the supertag. The unification algorithm takes two pairs of attribute and environment hash tables and returns a new pair of attribute and environment hash tables if unification succeeds, and *nil* otherwise.

6.3 Experimental Results

In an attempt to measure the speed-up obtained by a stapler in comparison to a full parser, we performed the following experiment. Parses of a set of 365 ATIS domain sentences were generalized and stored as FST as discussed in Chapter 5. A set of 100 sentences from ATIS domain was used as test data for the following experiments.

Experiment (a): The performance of XTAG on the 100 sentences is shown in the first row of Table 6.1. The coverage represents the percentage of sentences that were assigned a parse.

Experiment (b): The setup for this experiment is shown in Figure 6.4. The *almost parse* from the EBL lookup is input to the full parser of the XTAG system. The full parser does not take advantage of the dependency information present in the *almost parse*. However, it benefits from the supertag assignment to the words in the *almost parse*. This information helps the full parser, by reducing the ambiguity of assigning a correct supertag sequence for the words of the sentence.

The speed-up shown in the second row of Table 6.1 is entirely due to this reduction in ambiguity. If the EBL lookup fails to retrieve a parse, which happens for 20% of the sentences, then the ambiguity in supertag assignment is not reduced and the full parser parses with all the supertags for the words of the sentence. The drop in coverage is due

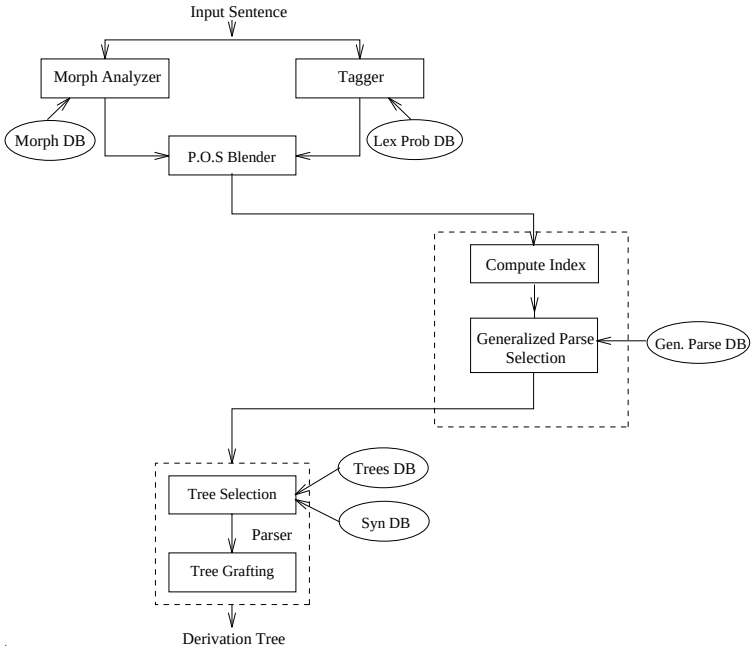


Figure 6.4: System setup used for Experiment (b).

to the fact that for 10% of the sentences, the retrieved generalized parse could not be instantiated to the features of the sentence.

System	# of sentences	Coverage %	Average time (in secs)
XTAG	100	100%	125.18
EBL+XTAG parser	100	90%	62.93
EBL+Stapler	100	70%	8.00

Table 6.1: Performance comparison of XTAG with and without EBL component

Experiment (c): The setup for this experiment is shown in Figure 6.5. In this experiment, the *almost parse* resulting from the EBL lookup is input to the stapler that generates all possible modifier attachments and performs the efficient unification, previously discussed, thus generating all the derivation trees. The stapler uses both the supertag assignment information and the dependency information present in the *almost parse* and speeds up the system even further, by a factor of about 15. However, it results in a further decrease in coverage by 10% due to the same reason as mentioned in Experiment (b). It must be

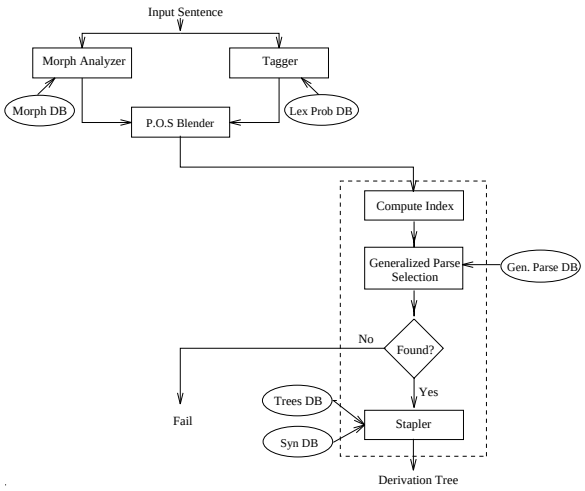


Figure 6.5: System setup used for Experiment (c).

noted that the coverage of this system is limited by the coverage of the EBL lookup. The results of this experiment are shown in the third row of Table 6.1.

6.4 Summary

In this chapter, we have presented a highly impoverished parser called the *stapler* that takes as input the *almost parse* structure resulting from the FST induced by the EBL mechanism and computes all possible parses and instantiates it to the particular sentence. Using this technique, the specialized grammar obtains a speed up of a factor of 15 over the unspecialized grammar on the Air Travel Information Service (ATIS) domain.

Chapter 7

Parser Evaluation Metrics

Performance evaluation of a parser can be distinguished into three kinds depending on the purpose they serve. [1] First, *intrinsic evaluation*, measures the performance of a parsing system in the context of the framework it is developed in. This kind of evaluation is applicable to both grammar based and statistical parsing systems since it helps system developers and maintainers to measure the performance of successive generations of the system. For grammar based systems, intrinsic evaluation helps identify the shortcomings and weaknesses in the grammar, and provides a direction for productive development of the grammar. For statistical parsers, intrinsic evaluation provides a measure of performance of the underlying statistical model and helps to identify improvements to the model. Since the evaluation is performed in the context of the framework that the parsing system is developed in, the metrics used for intrinsic evaluation can be made sensitive to the features and output representations of the parsing system.

A second method of evaluation of a parsing system is *extrinsic evaluation*. Extrinsic evaluation is meaningful when a parsing system is embedded in an application and it refers to the evaluation of the parsing system's contribution to the overall performance of the application. Extrinsic evaluation could be used as an indirect method of comparing parsing systems even if they produce different representations for their outputs as long as the output can be converted into a form usable by the application that the parser is embedded in.

[1] These evaluation methodologies are applicable to general purpose speech and natural language processing systems [Cole *et al.*, 1996; Jones and Galliers, 1995].

A third method of evaluation is *comparative evaluation*. The objective here is to directly compare the performance of different parsing systems that use different grammar formalisms and different statistical models. Comparative evaluation helps in identifying the strengths and weaknesses of different systems and suggests possibilities of combining different approaches. However, this evaluation scheme requires a metric that is insensitive to the representational differences in the output produced by different parsers. For this purpose, the metric may have to be sufficiently abstracted away from individual representations so as to reach a level of agreement among the different representations produced by parsers. However, as a result of the abstraction process, the strengths of representations of certain parsers might be lost completely.

In this chapter we focus on the comparative evaluation scheme. In Section 7.1, we discuss the methods of parser and grammar evaluations that have been suggested and used in the literature. We indicate the limitations of these metrics for the purpose of comparative evaluations in Section 7.2. In Section 7.3, we present our proposal, a Relation-based Model for Parser Evaluation, as an evaluation framework that overcomes the limitations of previous evaluation schemes. In Section 7.4, we present the results of evaluating the Supertagger and Lightweight Dependency Analyzer, using this scheme.

7.1 Methods for Evaluating a Parsing System

A parsing system can be evaluated along different dimensions ranging from grammatical coverage to average number of parses produced to average number of correct constituents in a parse produced by a system. Owing to this multi-dimensionality, there have been a variety of metrics that have been proposed for evaluating a parsing system. A comprehensive survey of different parsing metrics is provided in [Briscoe *et al.*, 1996]. These metrics can be divided into test suite-based and corpus-based methods. The corpus based methods are further divided into annotated and unannotated methods depending on whether the corpus is annotated for some linguistic information or not. In the sections that follow, we review each metric and discuss its strengths and weaknesses.

7.1.1 Test suite-based Evaluation

In this traditional method of parsing system evaluation, a list of sentences for each syntactic construction that is covered and not covered by the grammar is maintained as a database [Alshawi *et al.*, 1992; Grover *et al.*, 1993; XTAG-Group, 1995; Oepen *et al.*, forthcoming]. The test suite is used to track improvements and verify consistency between successive generations of the system that result from the addition of an analysis of a construction to the grammar or altering the analysis of a previously analyzed construction in the grammar. Although this method of evaluation has been mostly used for hand-crafted grammars, they could also be used to track the improvements in performance of statistical parsers with the changes in the underlying statistical model. The advantage of this method of evaluation is that it is relatively easy and straightforward and the negative information provides a direction for improving the system. However, the disadvantage is that it does not quantify how the performance of a parsing system would scale up when parsing unrestricted text data.

7.1.2 Unannotated Corpus-based Evaluation

The following methods also use unrestricted texts as corpora for evaluating parsing systems. However, the corpora consist of sentences which are not annotated with any linguistic information.

Coverage

Coverage is a measure of the percentage of sentences in the corpus that can be assigned one or more parses by a parsing system [Briscoe and Carroll, 1995; Doran *et al.*, 1994]. It is a weak measure since it does not guarantee that the analysis found is indeed the correct one. The output needs to be manually checked to determine this.

Average Parse Base

Average Parse Base (APB) [Black *et al.*, 1993a] is defined as the geometric mean of the number of analysis divided by the number of input tokens in each sentence parsed. This metric provides a method of predicting the average number of parses that a parsing system

would produce for a sentence of length n tokens. The metric is useful in comparing different versions of a parsing system that is under development, when tested on the same data. However, the disadvantage is that the metric does not measure the performance of correct analysis provided by the system and systems that have very low coverage would perform very well on this measure. Also, this metric cannot be used for cross-system comparative evaluation since the inherent ambiguity in the data interacts with the ambiguity of the grammar.

7.1.3 Annotated Corpus-based Evaluation

The following methods use unrestricted texts as corpora for evaluating parsing systems. The corpora consists of sentences which are annotated with information such as Part-of-speech tags, constituency labels, subject-verb-object triples. The annotation in the corpus is termed as the *gold standard*. The usefulness of the following methods is heavily dependent on the fidelity and consistency of annotation in the gold standard corpus. These methods also require that the output of the system be converted into the same representation as the gold standard.

Tagging Accuracy

This metric is designed with a view of evaluating the intermediate steps that are produced during the parsing process, such as part-of-speech and/or syntactic tags. The annotations are regarded as tags whose accuracy can be evaluated against a gold standard annotated with the correct tags. The accuracy with which a parsing system assigns the tag to a word in running text is measured in a variety of ways: as the ratio of correct tags per word, possibly divided into ambiguous and unambiguous and/or known and unknown words[Church, 1988; Karlsson *et al.*, 1994].

Structural Consistency

Structural Consistency metric [Black *et al.*, 1993b] is intended to measure the constituent accuracy of parses produced by a parsing system. Hence the gold standard annotation

consists of sentences with constituent brackets. Structural Consistency metric approximates is defined as the percentage of sentences for a test corpus which receive one or more global analyses one of which has no crossing brackets with the gold standard parse. Crossing bracket is the number of constituents that are inconsistent with the gold standard. Structural Consistency metric is an approximation of the more stringent metric of exact match of constituent structure. By itself, the structural consistency metric is insufficient since a grammar that assigns minimal structure (hence no inconsistent constituents) can score high on this metric as does a grammar that produces every possible bracketing for a sentence.

Ranked Consistency

One of the drawbacks of the Structural Consistency metric is that an a priori bound cannot be placed on how far down the list of parses produced can one expect to find the correct parse. Ranked Consistency metric is designed to overcome this limitation. It is defined as the percentage of sentences for a test corpus which receive one or more parses, one of the *top n ranked* of which has no crossing brackets with the gold standard parse. This metric gives a meaningful score as to how often a parsing system produces a consistent analysis with respect to a gold standard. The metric, however, assumes close compatibility of the output of the parsing system and the gold standard annotation. It also assumes that the parsing system produces a ranked order of parses.

Tree Similarity Measure

Tree similarity measures, employed by Sampson [Sampson *et al.*, 1989], uses a variety of metrics including the ratio of rules correctly deployed in a derivation to all rules in that derivation computed from the gold standard. This measure is more fine-grained than full identity of the parse in that a parsing system may not produce an entirely correct parse yet consistently produce analyses which are close to the correct one. This measure is also tolerant to errors in the gold standard.

Parseval

A metric that was designed with the aim of comparative evaluation of parsing system, Parseval [Harrison *et al.*, 1991] is an alternate mechanism of relaxing exact match to the parse in the gold standard as the success criterion. This scheme utilizes only bracketing information to compute three figures: crossing bracket, the number of brackets in the system's parse that cross the treebank parse; recall, a ratio of the number of correct brackets in the system's parse to the total number of brackets in the treebank parse and precision, a ratio of the number of correct brackets in the system's parse to the total number of brackets in the system's parse. The Parseval scheme served its purpose of providing a cross-representational evaluation metric using which the performance of various parsers could be quantified and compared – a hither-to-fore impossible task.

7.2 Limitations of Parseval

There have been several objections to the Parseval scheme. In this section, we summarize some of the objections and limitations of this scheme.

First, it was observed that crossing brackets measure penalizes mis-attachments more than once [Lin, 1995]. Consider for example, the sentence 7.1 with the annotation as the gold standard and the sentences 7.2 and 7.3 as parses from two parsing systems.

(7.1) [She [bought [[an incredibly expensive coat] [with [[gold buttons] and [fur lining]]]] [at [the store]]]]

(7.2) [She [bought [[an incredibly expensive coat] [with [[gold buttons] and [[fur lining] [at [the store]]]]]]]]

(7.3) [She [bought [an incredibly expensive coat] with [gold buttons] and [fur lining] [at [the store]]]]

Due to the mis-attachment of the phrase [at [the store]] to the phrase [fur lining] in the parse 7.2, three crossing brackets are created.

1. Between [[gold buttons] and [fur lining]] and [[fur lining] [at [the store]]]

2. Between [with [[gold buttons] and [fur lining]]] and [[gold buttons] and [[fur lining] [at [the store]]]]

3. Between [[an incredibly expensive coat] [with [[gold buttons] and [fur lining]]]] and [[an incredibly expensive coat] [with [[gold buttons] and [[fur lining] [at [the store]]]]]]

The recall of 7.2 is $\frac{6}{10}$=60% and the precision is $\frac{7}{10}$=63.6%. In contrast, the shallow parse in 7.3 with its minimal bracketing has a recall of 70%, a precision of 100% and no crossing brackets with the parse in 7.1. Although the parse 7.2 contains more information and is closer to the intended parse 7.1, it scores lower on the Parseval metric.

A second objection of the Parseval metric is that the precision measure penalizes a parser for inserting extra, possibly correct, brackets, if annotation in the treebank is skeletal [Srinivas *et al.*, 1995]. Consider a finer analysis of the phrase "an incredibly expensive coat" as shown in 7.4.

(7.4) [an [incredibly expensive] coat]

Since the structure in 7.4 is more detailed than the structure for the phrase in 7.1, the analysis in 7.4 results in a recall of 100% but a precision of 50%.

Due to the above mentioned limitations of the Parseval metric, it is unclear as to how the score on this metric relates to success in parsing. It is also not clear if this metric can be used for comparing parsers with different degrees of structural fineness since the score on this metric is tightly related to the degree of structural detail in the gold standard.

A significant limitation of the Parseval metric is that it has not been designed to evaluate parsers that produce partial parses. A partial parser could potentially produce constituents/chunks that may not be connected in a complete tree. In cases where an attachment of a constituent is hard to make, it might be equally useful to leave the constituent unattached than to force the parser to always attach it to some node. However, the bracket representation used by Parseval is inadequate to represent disconnected trees. In the bracket notation, there is an implicit assumption that the unattached constituents are attached to the root of the tree.

Another significant limitation of the Parseval metric is that it does not compare analyses to parsing tasks or applications. Parseval metric computes crossing brackets, recall and

precision as a way of approximating for the exact match criterion (identity of the output parse with the gold standard parse). Performance using the exact match criterion has a direct interpretation in terms of performance on tasks that a parser may be put to use. However, a measure that is an approximation to the exact match criterion, such as Parseval, has a less direct interpretation since the definition of approximation varies from task to task. Applications that use a parser may be interested in specific structures, Noun Phrases, Appositives, Predicative Nominatives, Subject-Verb-Object relations and so on. Parseval metric is not fine-grained enough to evaluate parses with respect to specific syntactic phenomena. It divorces the parser from the application the parser is embedded in.

7.3 Our Proposal

We present a proposal for parser evaluation that overcomes the limitations of the Parseval scheme and is also applicable for evaluating partial parsers. It is also intended to serve as a metric for evaluating parsers that are embedded in applications.

7.3.1 Application Independent Evaluation

The objective of this evaluation is to provide a measure that can be used to compare parsers irrespective of their output representation. Parsers are either constituency-based or dependency-based systems. Constituency-based parsers produce a hierarchically organized constituent structure, while dependency-based parsers produce a set of dependency linkages between the words of a sentence[2]. Furthermore, parsers could produce full parses that span the entire input string or partial parses that are a set of locally consistent parses for non-overlapping substrings of the input. Also, in terms of the detail of a parse, statistical constituency-based parsers produce relatively flat structures for Noun Phrases and Verb groups since the treebanks they are trained off of do not annotate for the internal structures of these elements. Hence, parsers that do produce internal structure for these elements need to be normalized first so that non-recursive phrasal constituents are flattened to chunks such as noun chunks, verb chunks, preposition chunks.

[2]Dekang Lin [Lin, 1995] proposed using a dependency linkage model for the complete sentence.

Also, hierarchically organized constituency notation does not have the provision to represent partial parses since an "unattached constituent" is implicitly assumed to attach to the highest constituent. Hence to facilitate the comparison of partial and full parsers, we propose that the accuracy of hierarchical structures be measured in terms of the accuracy of the relations between chunks that are expressed by dependency links between certain words (typically heads) of the chunks. Partial parses are thus penalized by their not being able to express certain dependency links due to unattached constituents.

Thus we propose a normalized representation that is based on chunks and dependency links. Evaluating this representation amounts to first evaluating the flat phrasal structures such as noun chunks, verb groups, preposition phrases (disregarding the attachment location) and then evaluating the correctness of hierarchical structures using dependencies between words (typically, heads) of the two chunks. If the dependency links are labeled then evaluation could be performed with and without including the labels.

7.3.2 Application Dependent Evaluation

The objective of this evaluation is to serve as an Adequacy Evaluation[3] of a parser. Adequacy Evaluation involves determining the fitness of a system for a purpose. It provides users (application developers) information to determine whether a parser is adequate for their particular task, and if so, whether it is more suited than others. The result of this evaluation is a report which provides comparative information of the parsers and does not necessarily identify the best parser, thus allowing the user to make an informed choice.

One of the dimensions of adequacy evaluation is to evaluate parsers from the point of view of specific grammatical constructions such as minimal and maximal Noun phrases, Appositives, Preposition Phrase modifiers, Predicative constructions, Relative Clauses, Parentheticals and Predicate-Argument relations. The degree of difficulty and accuracy of identifying these grammatical constructions, given a parse, depends on the representation adopted by the grammar underlying the parser. The appropriateness of the representation for a task can only be evaluated by evaluating the accuracy with which these grammatical constructions can be identified. Hence we recommend that parsers be evaluated based on their performance in identifying specific grammatical constructions.

[3] see [Cole *et al.*, 1996].

7.3.3 A General Framework for Parser Evaluation

In the preceding sections, we have identified two kinds of parser evaluations – application independent evaluation and application dependent evaluation. In this section, we propose a general framework called the Relation Model of parser evaluation that allows for the two kinds of evaluations to be instantiated in it. In this framework, a parse is viewed in multiple dimensions with each dimension expressing a relation R. A parser can be evaluated along any one (or all) of these dimensions. A gold standard used for evaluation is viewed as expressing one particular relation. Performance of the parser in expressing the relation R is measured in terms of recall, precision and F-measure. F-measure provides a parameter β that can be set accordingly, so as to measure the performance of a system depending on the relative importance of recall to precision. A summary of the evaluation framework is shown in 7.1.

- Let $x\ R\ y$ represent that x is in a relation R with respect to y .
- Let S_{gold} be the relation, R, expressed in the key (annotated corpus).
- Let S_{out} be the relation, R, expressed in the output of the parser.
- Recall = $(\mathrm{S}_{gold} \cap \mathrm{S}_{out})/\mathrm{S}_{gold}$
- Precision = $(\mathrm{S}_{gold} \cap \mathrm{S}_{out})/\mathrm{S}_{out}$
- F-Measure = $((\beta^2 + 1) * \mathrm{P} * \mathrm{R})/(\beta^2 * \mathrm{P} + \mathrm{R})$ where β is the relative importance of Recall and Precision.

Figure 7.1: Summary of the Relation Model of Parser Evaluation

A few instantiations of this general framework are as follows: For evaluating chunks of type t where t could be a noun chunk, verb chunk, preposition phrase and so on; R is defined as the relation: x *starts* and y *ends* the chunk type t. A similar instantiation could be used to evaluate the parser on specific constructions. For evaluating dependencies, the relation R is defined as x *depends on* y. The dependency links could be evaluated further based on their labels.

7.4 Evaluation of Supertag and LDA system

In this section, we discuss the performance of Supertagging for text chunking and performance of LDA for dependency analysis on a range of corpora.

7.4.1 Performance of Supertagging for Text Chunking

We have applied the supertag and Lightweight Dependency Analyzer (LDA) system for text chunking. Text chunking, proposed by Abney [1991] involves partitioning a sentence into a set of non-overlapping segments. Text chunking serves as a useful and relatively tractable precursor to full parsing. Text chunks primarily consist of non-recursive noun and verb groups. The chunks are assumed to be minimal and hence non-recursive in structure.

Noun Chunking

Supertag based text chunking is performed by the local application of functor-argument information encoded in supertags. Once the head noun of the noun chunk is identified, the prenominal modifiers that are functors of the head noun or functors of the functors of the noun are included in the noun chunk. The head NP is identified as the noun with a initial (non recursive) supertag (A_NXN). Thus the following simple algorithm identifies noun chunks, for example, shown in (7.5), (7.6) and (7.7).

Algorithm: *Scan right to left starting with the noun initial supertag (A_NXN) and collect all functors of a noun or a noun modifier.*

Some example noun chunks identified by this algorithm from supertagged WSJ texts are shown in below.

(7.5) New Jersey Turnpike Authority

(7.6) its increasingly rebellious citizens

(7.7) two $ 400 million real estate mortgage investment conduits

Ramshaw and Marcus [1995] present a transformation-based noun chunker using three specialized tags (I,O,B). Each word of the training corpus is annotated to indicate if the

word is included in a noun chunk (I), outside a noun chunk (O) or at the boundary of two noun chunks (B). A set of transformation rules are then learned using an error-driven transformation-based learning scheme presented in [Brill, 1993]. The performance of this scheme using rules defined on parts of speech and rules incorporating lexical information is shown in Table 7.4.1.

System	Training Size	Recall	Precision
R&M	Baseline	81.9%	78.2%
R&M (without lexical information)	200,000	90.7%	90.5%
R&M (with lexical information)	200,000	92.3%	91.8%
Supertags	Baseline	74.0%	58.4%
Supertags	200,000	93.0%	91.8%
Supertags	1,000,000	93.8%	92.5%

Table 7.1: Performance comparison of the transformation based noun chunker and the supertag based noun chunker

Table 7.4.1 also shows the performance of the supertag based noun chunking, trained and tested on the same texts as the transformation-based noun chunker. The supertag-based noun chunker performed better than transformation-based noun chunker with lexical templates, even though the supertag-based noun chunker does not use lexical context in determining the extent of a noun chunk. The supertagger uses lexical information on a per word basis only to pick an initial set of supertags for a given word. Moreover, the supertag-based noun chunking not only identifies the extents of the noun chunks but by the virtue of the functor-argument information, provides internal structure to the noun chunks. This internal structure of the noun chunks could be utilized during subsequent linguistic processing.

Verb Chunking: We also performed an experiment similar to noun chunking to identify verb chunks. We treat a sequence of verbs and verbal modifiers, including auxiliaries, adverbs, modals as constituting a verb group. Similar to the noun chunk identification algorithm discussed above, the verb chunking based on supertags can be performed using local functor-argument information encoded in supertags. However, for verb chunks, the scan is made from left to right starting with a verbal modifier supertag, either an auxiliary verb or an adverbial supertag and including all functors of a verb or a verb modifier. Thus

the following simple algorithm identifies verb chunks, for example, shown in (7.8), (7.9) and (7.10).

Algorithm: *Scan left to right starting with the verb or verbal modifier supertag and collect all functors of a verb or a verb modifier.*

(7.8) would not have been stymied

(7.9) did n't even care

(7.10) just beginning to collect

System	Training Size	Recall	Precision
R&M	Baseline	60.0%	47.8%
R&M (without lexical information)	200,000	83.6%	83.5%
R&M (with lexical information)	200,000	88.5%	87.7%
Supertags	Baseline	76.3%	67.9%
Supertags	200,000	86.5%	91.4%
Supertags	1,000,000	87.2%	92.2%

Table 7.2: Performance comparison of the transformation based verb chunker and the supertag based verb chunker

Table 7.4.1 shows the performance of the transformation based verb chunker and the supertag based verb chunker, both trained and tested on the same sentences of the Wall Street Journal Corpus. The supertag-based verb chunker had higher precision than the transformation based verb chunker with lexical information but had lower recall. The supertag-based verb chunking not only identifies the extents of the verb chunks but by the virtue of the subcategorization information, provides internal structure to the verb chunks. This internal structure of the verb chunks could be utilized during subsequent linguistic processing.

Preposition Phrase Attachment

Supertags distinguish a noun-attached preposition from a verb-attached preposition in a sentence, as illustrated by the two different supertags in (7.11) and (7.12). Due to this

distinction, the supertagging algorithm can be evaluated based on its ability to select the correct supertag for the prepositions in a text.

(7.11) sell 500 railcar platforms to/B_vxPnx Trailer Train Co. of/B_nxPnx Chicago

(7.12) begin delivery of/B_nxPnx goods in/B_vxPnx the first quarter

The task of preposition attachment has been worked on by a number of researchers in the past. Humans perform only at an accuracy of 88% accuracy [Ratnaparkhi *et al.*, 1994] which gives an indication of the complexity of the task. Table 7.3 presents a comparative evaluation of various approaches in the literature against the supertag based approach, for the preposition attachment task.

System	Accuracy
Ratnaparkhi, Reynar & Roukos	77.7%
Ratnaparkhi, Reynar & Roukos (with classes)	81.6%
Hindle & Rooth	78-80%
Brill & Resnik	81.9%
Collins & Brooks	84.5%
Supertags	81.1%

Table 7.3: Performance comparison of the supertag based preposition phrase attachment against other approaches

It must be pointed out that the supertagger makes the preposition attachment decision (choosing between B_vxPnx and B_nxPnx) based on a trigram context. Most often than not the preposition is not within the same trigram window as the noun and the verb due to modifiers of the noun and the verb. However, despite this limitation, the trigram approach performs as well as Ratnaparkhi, Reynar & Roukos [1994] and Hindle and Rooth [1991], and is outperformed only by methods (Brill and Resnik [1994], Collins and Brooks[1995]) that use four words: the verb, the object noun, the preposition and the complement of the preposition in making the attachment decision.

Other Constructions

In this section, we summarize the performance of the supertagger in identifying constructions such as appositives, parentheticals, relative clauses and coordinating conjunctions.

Appositives: In the XTAG grammar, the appositive construction has been analyzed using a Noun Phrase modifying supertag that is anchored by a comma which takes the appositive Noun Phrase as an argument; for example, the comma in (7.13) and the second comma in (7.14) anchor appositive supertags. Thus, a comma in a sentence could either anchor an appositive supertag or a set of coordination supertags, one supertag for each type of coordination. Further, a comma also anchors supertags that mark parentheticals as in (7.15). The task of the supertagger is to disambiguate among the various supertags and assign the appropriate supertag to the comma, in the appositive context. In Table 7.4, we present the performance results of the supertagger on such a task. The baseline of assigning the most likely supertag to comma results in zero percent recall.

Parentheticals: Propositional attitude verbs such as *say* and *believe* can not only appear in the canonical word order of subject-verb-complement, but can also appear at other positions in a sentence, as in (7.14) and (7.15). Also, the relative order of the subject and the verb can also be reversed as in (7.14). The XTAG grammar distinguishes such adverbial constructions from the sentence complement construction by assigning different supertags to the verb. A detailed analysis of this construction is presented in [Doran, 1996]. The task of the supertagger is to disambiguate among the sentence complement supertag and the various adverbial supertags for the verb given the context of the sentence. Table 7.4 presents the performance results of the supertagger on this task. Once again the baseline of selecting the most likely supertag of the verb results in zero percent recall.

(7.13) Van Pell , 44

(7.14) " The U.S. underestimated Noriega all along , " *says* Ambler Moss , a former Ambassador to Panama

(7.15) Mr. Noriega 's relationship to American intelligence agencies became contractual in either 1966 or 1967 , intelligence officials *say* .

Coordination Conjunctions: In Table 7.4, we also present the performance of the supertagger in identifying coordination constructions. Coordination conjunctions anchor a supertag that requires two conjuncts, one on either side of the anchor. There is one supertag for every possible pair of conjunct types. We not only have supertags that coordinate like

Construction	# of occurrences	# identified	# correct	Recall	Precision
Appositive	362	491	306	84.5%	62.3%
Parentheticals	225	200	170	75.5%	85%
Coordination	1886	1750	1329	70.5%	75.9%
Relative Clauses	297	269	116	39.0%	43.2%
Relative Clauses (ignoring valency)	297	269	156	52.5%	58%

Table 7.4: Performance of the trigram supertagger on Appositive, Parentheticals, Coordination and Relative Clause constructions.

types but we also include supertags that coordinate unlike types amounting to about 30 different supertags for coordination.

Relative Clauses: As discussed in Chapter 3.2, due to extended domain of locality of LTAGs, verbs are associated with one relative clause supertag for each of the arguments of the verb. This is unlike CCG, where the relative pronoun is associated with multiple supertags. A more detailed discussion about the differences in the distribution of ambiguity between the CCG and LTAG is presented in [Doran and Srinivas, 1994]. The task of the supertagger in LTAG is not only to identify that the verb is in a relative clause structure but also to identify the valency of the verb and the argument that is being relativized. The performance of the supertagger with and without the valency information being taken into account is present in Table 7.4.

We are unaware of any other quantitative results on WSJ data for identifying these constructions, for comparative evaluation.[4] It is interesting to note that the performance of the supertagger in identifying appositives and parenthetical construction is better than for coordination conjunctions and relative clause constructions. We believe that this might in part be due to the fact that Appositives and Parentheticals can mostly be disambiguated using relatively local contexts. In contrast, disambiguation of Coordinate constructions and Relative clauses typically require large contexts that are not available for a trigram supertagger.

[4]Lin [1996] provides results for some of these constructions on the Brown corpus.

7.4.2 Performance of Supertag and LDA system

In this section, we present results from two experiments using the supertagger in conjunction with the LDA system as a dependency parser. In order to evaluate the performance of this system, we need a dependency annotated corpus. However, the only annotated corpora we are aware of, the Penn Treebank [Marcus *et al.*, 1993] and the SUSANNE corpus [Sampson, 1994], annotate sentences with constituency trees. In the interest of time and the need for a dependency annotated corpus, we decided to transform the constituency trees of the two corpora using some rules and heuristics. It must be noted that the resulting corpora is only but an approximation to a manually annotated dependency corpus. However, although the annotation resulting from the transformation does not conform to a standard dependency annotation, it is nevertheless an invaluable resource for performance evaluation of dependency parsers.

The WSJ corpus from the Penn Treebank and the subset of the Brown corpus in SUSANNE were converted by two related but different mechanisms. The conversion process of the Penn Treebank proceeded as follows. For each constituent of a parse a head word is associated using the head percolation table introduced in [Jelinek *et al.*, 1994; Magerman, 1995]. The head percolation table associates with each possible constituent label an ordered list of possible children of the constituent whose head word is passed to the parent. The annotation of a parse tree with head word information proceeds bottom up. At any constituent, the percolation information is consulted to determine which of the constituent's children would pass the head word over to their parent. Once the head words are annotated, the dependency notation is generated by making the head words of non-head constituents to be dependent on the head of the head constituent. An example of this procedure is illustrated in Figure 7.2.

In a similar manner to the WSJ conversion process, we converted the subset of the LOB annotation of the SUSANNE corpus into a dependency notation. However, in this conversion process we could not use the head information table used in the WSJ conversion process since the annotation labels are different. Instead, we regard a constituent label to be a projection of one of its children, which was regarded as the head constituent. Once the heads are annotated the dependency notation is generated by making the head words

```
( (S
    (NP-SBJ
      (NP (NNP Pierre) (NNP Vinken) )
      (, ,)
      (ADJP
        (NP (CD 61) (NNS years) )
        (JJ old) )
      (, ,) )
    (VP (MD will)
      (VP (VB join)
        (NP (DT the) (NN board) )
        (PP-CLR (IN as)
          (NP (DT a) (JJ nonexecutive) (NN director) ))
        (NP-TMP (NNP Nov.) (CD 29) )))
    (. .) ))

1 Pierre 2 Vinken
2 Vinken  8 will
3 , 2 Vinken
4 61 5 years
5 years 6 old
6 old 2 Vinken
7 , Vinken
8 will 8 will
9 join 8 will
10 the 11 board
11 board 9 join
12 as 9 join
13 a 15 director
14 nonexecutive 15 director
15 director 12 as
16 Nov. 9 join
17 29 16 Nov.
18 . will
```

Figure 7.2: The Phrase Structure tree and the dependency linkage obtained from the phrase structure tree for the WSJ sentence *Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29.*

of non-head constituents to be dependent on the head word of the head constituent.

In the first experiment we use the dependency versions of the Penn Treebank annotation of the WSJ corpus and the LOB annotation of the SUSANNE corpus as gold standards. However, in the second experiment, we use derivation structures of WSJ sentences that were parsed using the XTAG system as the gold standard. The derivation structures serve as dependency structures that are closest in conventions to those assumed by the LDA system.

Experiment 1:

The trigram supertagger trained on 200,000 words of the WSJ corpus was used as in conjunction with the LDA to provide a dependency analysis for 2000 sentences of Section 20 of the WSJ corpus. The Penn Treebank parses for these sentences were converted into dependency notation which was used as the gold standard. A dependency link produced by the LDA was regarded to be correct if the words being related by the link were also present in the gold standard. However, to account for the differences in the annotation, the dependency relations among words in a noun group were treated as equivalent to each other. So also, dependency relations among words in a verb group were treated as equivalent to each other. There were 47,333 dependency links in the gold standard and the LDA produced 38,480 dependency links correctly, resulting in a recall score of 82.3%. Also, a total of 41,009 dependency links were produced by the LDA, resulting in a precision score of 93.8%.

We conducted a similar experiment using the SUSANNE corpus. Using the same trigram supertagger trained on the 200,000 words of the WSJ corpus in conjunction with the LDA, we provided a dependency analysis for the sentences in the SUSANNE corpus (125,000 words). The gold standard was created by converting the phrase structure annotations for these sentences into dependency notation using the procedure described above. As in the case of WSJ corpus, to account for the differences in the annotation, the dependency relations among words in a noun group were treated as equivalent to each other. So also, dependency relations among words in a verb group were treated as equivalent to each other. There were a total of 126,493 dependency links in the output of the LDA of which, 112,420 also present in the gold standard, resulting in a precision score

of 88.8%. There were a total of 140,280 links in the gold standard, resulting a recall of 80.1%. It is interesting to note that although the trigram model was trained on the WSJ corpus, the performance of the LDA on SUSANNE is comparable to the performance of the LDA on the WSJ corpus.

On analyzing the errors, we discovered that several of the errors were due to the approximate nature of the dependency corpus created by the conversion process. A second source of errors was due to the differences in the notion of dependency produced by the LDA system and that resulting from the conversion of the Treebank. This is largely due to a lack of a standard dependency notation.

Corpus	System	# of dependency links	# produced by LDA	# correct	Recall	Precision
Brown	LDA	140,280	126,493	112,420	80.1%	88.8%
WSJ	LDA	47,333	41,009	38,480	82.3%	93.8%

Table 7.5: Comparative Evaluation of LDA on Wall Street Journal and Brown Corpus

Experiment 2

In this experiment, we used the derivation structures produced by XTAG for 1350 WSJ sentences[5] as the gold standard. The correct derivation was hand picked for each of the 1350 WSJ sentences. These sentences were supertagged using the supertagger trained on 8000 sentences of WSJ and dependency annotated using the LDA system. The metric for evaluation was Recall and Precision computed based on the number of dependency links present in both the gold standard and the output produced by the system. Table 7.6 shows the performance of the system. Although, the derivation trees produced by the XTAG system are closest in terms of conventions used in the dependency output of the LDA system; there are a few points of divergences, a major one being the annotation for sentence complement verbs. While in the LDA system, the embedded verb depends on the matrix verb the reverse is the case in an LTAG derivation structure [Rambow *et al.*, 1995]. Also, since the XTAG system, as it is implemented does not permit two auxiliary trees to be adjoined at the same node, certain valid derivation structures are not possible in XTAG. A precise formulation of possible derivation structures in LTAG framework is

[5]sentences of length less than 16

presented in [Schabes and Shieber, 1992].

Training Size (words)	Test Size (words)	Recall	Precision
200,000	12,000	83.6%	83.5%

Table 7.6: Performance of LDA system compared against the XTAG derivation structures

% sentences with 0 errors	% sentences with with $\leq$1 error	% sentences with with $\leq$2 errors	%sentences with with $\leq$3 errors
35%	60.3%	78%	89.8%

Table 7.7: The percentage of sentences with zero, one, two and three dependency link errors.

Next, we present the performance of the LDA in terms of its accuracy at producing a dependency linkage at the level of a sentence, when evaluated against an LTAG derivation tree. Table 7.7 tabulates the percentage of sentences that have zero, one, two and three dependency link errors. In contrast to evaluation against a skeletally bracketed treebank, evaluation against LTAG derivation trees is much more strict. For example, an LTAG derivation contains detailed annotation in terms of the internal structure of the nominal modifiers in Noun Phrases and verbal modifiers in Verb groups. Also, a derivation structure has a stricter imposition of the argument-adjunct distinction and distinguishes readings that have similar phrase structure trees such as predicative and equative readings and idiomatic and non-idiom readings of a sentence. Further, the derivation structure is much more closer to semantic interpretation of a sentence than the phrase structure. Hence, the performance figures that are shown in Table 7.6 and Table 7.7 are more strict and hence more significant than the crossing bracket, precision and recall figures measured against skeletally bracketed corpora.

7.5 Summary

This chapter can be seen as consisting of two parts. In the first part, we reviewed some of the currently prevalent evaluation metrics for parsers and indicated their limitations for comparing parsers that produce different representations. We proposed an alternate evaluation metric based on relation model of evaluation that overcomes the limitations of

the existing evaluation metrics. In the second part of this chapter, we provided results from extensive evaluation of the supertagger and the LDA system on a range of tasks and compared its performance against the performance of other systems on those tasks.

Chapter 8

Applications of Supertagging

Supertagging has been used in a variety of applications including information retrieval and information extraction, text simplification and language modeling. Some of these applications have been developed in collaboration with researchers at IRCS (Dr. Baldwin, Dr. Chandrasekar and Dr. Niv) and students (Christine Doran and Jeffrey Rayner) in the Department of Computer and Information Sciences at University of Pennsylvania.

Although it is certainly true that a full parser can be used in all the applications discussed in this chapter, our goal in using a supertagger was to exploit the strengths of supertags as the appropriate level of lexical description needed for most applications. Also, due to their local nature, supertags were amenable to spot retraining. By spot retraining we mean that if an application needs to make a certain distinction say for example supertagging a time noun phrase (eg. last July) differently from a common noun phrase (eg. primary elections), the training material for the supertagger can be easily changed and the supertagger can be retrained quickly. The high speed and low memory requirements of the supertagger proved to be useful advantages as well.

The outline of this chapter is as follows. In Section 8.1 we discuss the use of syntactic information in the form of supertags in an information retrieval system. We compare the retrieval efficiency of a system based on part-of-speech tags against a system based on supertags. In Section 8.2 we discuss the application of supertags in an information extraction task. The application of supertags as syntactically motivated class labels in a class-based language modeling task is presented in Section 8.3. The distinction between

recursive and non-recursive supertags is exploited in a sentence simplification application that is discussed in Section 8.4. The ability to bring to bear document-level and domain-specific constraints during supertagging of a sentence is discussed in Section 8.5.

8.1 Information Retrieval

This work has been carried out in collaboration with Dr. Chandrasekar, Visiting Scholar, IRCS, University of Pennsylvania.

The availability of vast amounts of useful textual information in machine-readable form has led to a resurgence of interest in Information Retrieval (IR). There is considerable academic and business interest now in analyzing unrestricted text to extract information of relevance, and in the development of information retrieval and information extraction systems.

Although the ultimate goal in Information Retrieval is to have a system which 'understands' all the text that it processes, and responds to users' queries 'intelligently', there are many open problems in syntactic processing, semantic analysis and discourse processing that need to be solved before tools that are required for 'understanding' texts could be developed.

As a result of the limitation of not being able to automatically 'understand' texts, most IR systems approximate linguistic information by using keywords and features such as proximity and adjacency operators. But the standard problems of synonymy and polysemy adversely affect recall and precision of information retrieval. Users typically have to scan a large number of potentially relevant items to get to the information that they are looking for. (See [Salton and McGill, 1983], [Frakes and Baeza-Yates, 1992] for details on work in information retrieval.)

Clearly, it is inadequate to just retrieve documents which contain keywords of interest. Since any coherent text contains significant latent information, such as syntactic structure and patterns of language use, this can be exploited to make information retrieval more efficient.

In this section, we demonstrate quantitatively how syntactic information is useful in filtering out irrelevant documents. In contrast to earlier approaches which used syntactic

information during information retrieval stage, for example [Croft *et al.*, 1991], we use it in a filtering stage, after basic information retrieval. We also compare two different syntactic labelings – simple Part-of-speech (POS) labeling and Supertag labeling and show how the richer (more fine-grained) representation of Supertags leads to more efficient and effective document filtering.

Our task is to develop a system which uses syntactic information inherent in text to improve the efficiency of retrieval (given any basic IR tool) by automatically or semi-automatically identifying patterns of relevance.

8.1.1 Methodology

In this section, we describe how augmented-patterns for the domain of appointments (of people to posts) are extracted from a corpus of news text, and applied to filter out irrelevant information. In this description, all training and testing is done with sentences as the units of information. However, the methodology applies to larger units of text, where the relevant sentences are extracted using simple keyword matching.

Identifying the Training Set

A large textual corpus is first segmented into sentences. All sentences related to the word(s) of interest (in this case, *appoint* or a morphological variant) are extracted using some simple tool. These sentences are examined to see which of them are relevant to the domain of interest. Some decisions about the relevance of sentences may not be very easy. These decisions determine the scope of the filtering that is achieved using this system. 'Augmented-patterns' are then induced from the relevant sentences.

Inducing Patterns from Training Data

The training data is first tokenized into sentences and the relevant sentences are processed to identify phrases that denote names of people, names of places or designations. These phrases are converted effectively to one lexical item. The chunked relevant sentences are then tagged (with POS tags or supertags) and the tags associated with the words in the sentences are used to create noun-groups (involving prenominal modifiers) and verb-groups

(involving auxiliaries, modals, verbs and adverbs). At this stage, we have an abstract view of the structure of each sentence.

We look at a small window around the word(s) of our interest (in this case, one chunk on either side of the word *appoint* or its morphological variant), skipping punctuation marks. The word and the syntactic labels in this window are then generalized to a small set of augmented patterns, where each augmented pattern is a description involving tags, punctuation symbols and some words. The patterns for all sentences are then sorted, and duplicates removed. The resulting patterns can be used for filtering. Once a set of patterns is identified for a particular class of query, it can be saved in a library for later use.

Generalization brings out the syntactic commonality between sentences, and permits an economical description of most members of a set of sentences. We expect that a few patterns will suffice to describe the majority of sentences, while several patterns may be necessary to describe the remaining sentences. We could also sort patterns by the number of sentences that each of them describes, and ignore all patterns below some reasonable threshold. Note that generalization could increase recall while reducing precision, while thresholding decreases recall.

Pattern Application

The task in the pattern application phase is to employ the patterns induced in the pattern training phase to classify new sentences into relevant and irrelevant ones. The new sentences could be part of documents retrieved from news-wire texts, from the World Wide Web (WWW), etc. The relevance of a document is decided on the basis of the relevance of the sentences contained in it.

In this phase, the sentences of each document are subjected to similar stages of processing as were the training sentences. Each sentence in each document is chunked based on simple named-entity recognition and then labeled with syntactic information (POS tags or supertags). The syntactic labels for the words in each sentence are used to identify noun and verb chunks. At this stage, the sentence is ready to be matched against the patterns obtained from the training phase. A sentence is deemed to be relevant if it matches one or more of these patterns. Since the pattern matching is based on simple regular expressions specified over words and syntactic labels, it is extremely fast and robust. A document is

deemed relevant if it contains at least one relevant sentence.

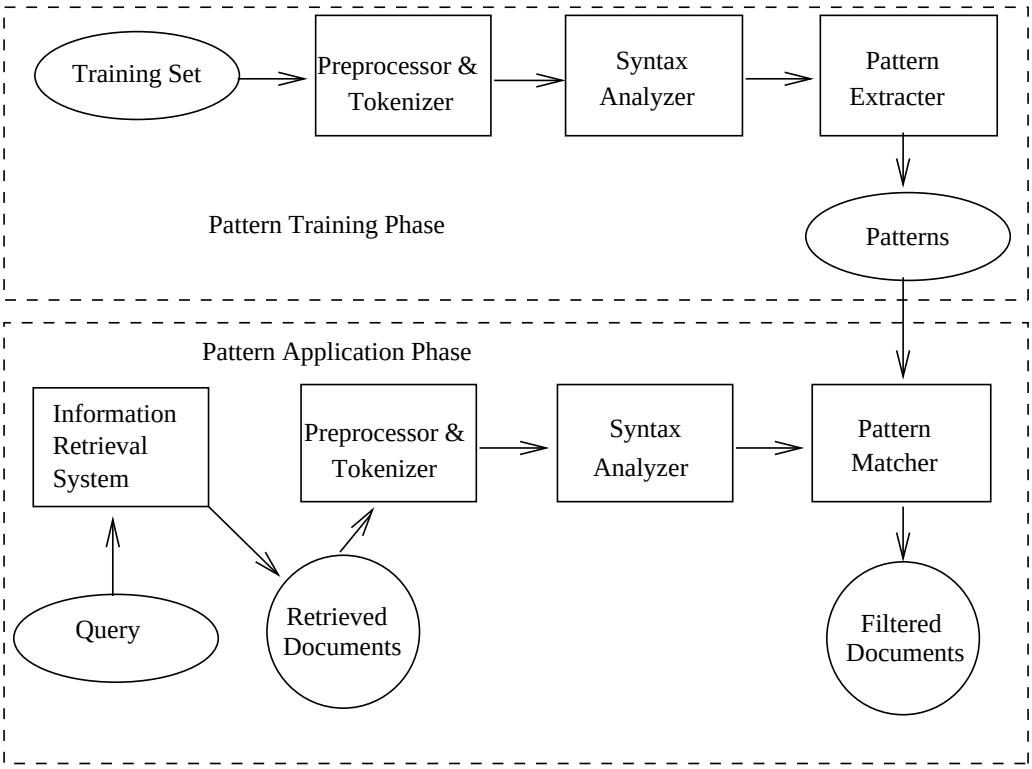


Figure 8.1: Overview of the Information Filtering scheme

Such a tool for information filtering, named Glean [Chandrasekar and Srinivas, 1996], is being developed in a research collaboration between the National Centre for Software Technology (NCST), Bombay, the Institute for Research in Cognitive Science (IRCS) and the Center for the Advanced Study of India (CASI), University of Pennsylvania. Glean seeks to innovatively overcome some of the problems in IR mentioned above, and is predicated on the idea that any coherent text contains significant latent information, such as syntactic structure and patterns of language use, which can be used to enhance retrieval efficiency. In particular, Glean uses the notion of 'agents', a combination of augmented textual-patterns and code, to identify specific structural features in the text.

In the next section, we describe experiments on retrieval efficiency using two syntactic labeling schemes: POS tagging and supertagging. We also discuss another experiment that uses this methodology to filter information retrieved from the World Wide Web.

8.1.2 The Experiment: POS Tagging *vs* Supertagging

The objective of this experiment is to quantitatively measure the performance improvement achieved by richer syntactic information for filtering out irrelevant documents.

The experiment: Training Phase

The text corpus constituted of approximately 52 MB of New York Times (NYT) data comprising of the August 1995 wire service output.[1] The corpus was sentence-segmented, and all sentences from this corpus that contained the word *appoint* or any of its morphological variants were extracted using the Unix tool `grep`. We plan to handle other equivalent words and phrases later.

The 494 sentences containing *appoint** were examined manually, and a subset of them (56 sentences) which were actually relevant to appointments being announced were identified. This constituted the training corpus. This included sentences about the appointments of groups of people such as panel, commissions etc. We rejected sentences where *appointed* was used as an adjectival or in a relative clause, and where a variant of *appoint* was used in the noun sense (eg. *appointee/appointment*). Most of the 56 acceptable sentences came from sentences with the word *appointed*; a few came from *appoint* and *appointment*.

Patterns obtained using POS tags The 56 relevant sentences were preprocessed to normalize punctuation. Named-entities were identified and grouped into single tokens. These sentences were then POS tagged and the tags were used to chunk verb groups and noun phrases. The chunked sentences were then processed to obtain 20 generalized patterns.

Patterns obtained using supertags In a similar manner, the training sentences were processed to obtain 21 distinct patterns using supertags instead of POS tags.

Sample patterns involving POS tags and supertags respectively, which match sentences that contain a noun phrase, followed by the transitive verb *appointed*, possibly qualified by auxiliaries and preverbal adverbs, and followed by a noun phrase are shown in Figure 8.2. Using such patterns, sentences from a variety of domains were categorized into relevant and irrelevant sentences.

[1] NYT text was chosen since it was felt that it would have more variety than, for instance, the Wall Street Journal.

POS: \S*/E_NG \S*:E_VGQUAL appointed:VBN/E_VG \S*/E_NG
Supertag: \S*/A_NXN \S*:E_VGQUAL appointed:A_nx0Vnx1/E_VG \S*/A_NXN
Key: \S* refers to any word/phrase; E_VGQUAL is any set of verbal qualifiers; E_VG is a verb group. A_NXN is a noun-phrase supertag, and A_nx0Vnx1 refers to a verb preceded and followed by a noun-phrase: a transitive verb. E_NG is a tag for a noun phrase, and VBN is a POS tag for a past participle verb.

Figure 8.2: Sample patterns involving POS tags and Supertags

The experiment: Testing Phase

From the July 1995 NYT wire service text, all sentences (a total of 529 sentences) containing the word *appoint* or its variant were extracted using `grep`. This constituted the base case, where sentences are retrieved using a simple retrieval mechanism (such as `grep`), with no filtering applied.

Domain	Total Sents	Rel Sents	Classified as relevant			Classified as irrelevant		
			Total	Correct	Incorrect	Total	Correct	Incorrect
NYT July95 (Supertags)	529	95	168	77	91	361	343	18
NYT July95 (POS)	529	95	73	42	31	456	392	64
Base Case	529	95	529	95	434	0	0	0

Table 8.1: Classification of appoint* sentences

These 529 sentences also constituted the test set. The gold standard was independently created by manually examining these sentences and classifying them into 95 relevant sentences and 434 irrelevant sentences (with respect to the task).

The patterns obtained from the training phase were applied to the 529 sentences. As before, these sentences were processed in a manner similar to the training data. All sentences which matched the augmented-patterns were deemed relevant by the program. The relevant and irrelevant sets for each method (POS tags and Supertags) were compared to the standard relevant and irrelevant sets.

The results of these experiments are summarized in Table 8.1, for the supertagging

Domain	Recall	Precision	F-score ($\beta = 1.0$)	F-score ($\beta = 0.5$)	F-score ($\beta = 1.5$)
NYT July95 (Supertagging)	(77/95) = 81%	(77/168) = 49%	61	72	56
NYT July95 (Part of Speech)	(42/95) = 44%	(42/73) = 58%	50	46	53
Base Case (all appoint*)	(95/95) = 100%	(95/529) = 18%	31	52	24

Table 8.2: Precision and Recall of different filters, for **relevant sentences**

Domain	Recall	Precision	F-score ($\beta = 1.0$)	F-score ($\beta = 0.5$)	F-score ($\beta = 1.5$)
NYT July95 (Supertagging)	(348/434) = 80%	(348/373) = 94%	86	82	89
NYT July95 uniq (Part of Speech)	(380/434) = 88%	(380/452) = 84%	88	89	87
Base Case (all appoint*)	(0/434) = 0%	(0/0) –	–	–	–

Table 8.3: Precision and Recall of different filters, for **irrelevant sentences**

method, the POS tag method and for the base case. The second column in the table gives the count of sentences judged relevant by humans. The columns that follow list judgments made by the program, and the overlap they have with the standard set.

We have computed the recall, precision and F-score measure for the three methods shown in Table 8.1. F-score [Hobbs *et al.*, 1992] is defined as follows:

$$\text{F-score} = ((\beta^2 + 1) * \text{P} * \text{R})/(\beta^2 * \text{P} + \text{R})$$

F-score provides a method of combining recall and precision. It provides a parameter β that can be set to measure the performance of a system depending on the relative importance of recall to precision.

Table 8.2 shows the recall and precision scores for relevant sentences, for each of the three methods shown in Table 8.1. It also shows F-scores for the three cases: precision is as important as recall (β=1), precision is less important than recall (β=0.5) and precision is more important than recall (β=1.5). Similar results for irrelevant sentences are summarized in Table 8.3.

POS Tagging & Supertagging: Merits and Demerits

In this section, we briefly summarize the merits and demerits of POS and supertag labels used in the experiments above.

Granularity of description:

The tags (POS and supertag labels) employed in our approach serve to categorize different syntactic contexts of a word. In general, a richer set of tags leads to a better discrimination of the context. The supertags provide an optimal descriptive granularity, which is neither at the level of complete parses (which may often be hard or impossible to obtain), nor at the simpler level of parts of speech. Since supertags are richer in representation when compared to POS tags, it is expected that supertags would provide better discrimination.

Sources of Error:

Both POS tagging and supertagging use statistical methods, and their accuracy and completeness depend on the material that was used to train them. The genre of the training material also introduces lexical biases. In addition, the vastly bigger tagset for supertagging makes it more prone to mistakes than POS tagging. Further, errors in tagging during the pattern training and pattern application phases can cause erroneous patterns to be created and lead to wrong categorization of documents respectively.

8.1.3 Gleaning Information from the Web

In this section, we describe another experiment which uses the techniques discussed in the Section 8.1.1 on documents about appointments retrieved from the World Wide Web.

Design of the Experiment

Given a search expression, URLs (Uniform Resource Locators) of the documents that match the search expression are retrieved, using a publicly available search engine. The URLs are then filtered for duplicates and the document corresponding to each URL is then retrieved. Each retrieved document is then processed using the tools described in Section 8.1.1. A document is deemed relevant if it matches at least one of the patterns induced for that domain.

For the particular experiment we performed, we used the Alta Vista web search engine (see `http://altavista.digital.com/`) to retrieve the URLs matching a search expression, using the WWW::Search and WWW::Search::AltaVista Perl modules distributed from ISI
(`http://www.isi.edu/lsam/tools/WWW_SEARCH/`). The document corresponding to each matching URL was downloaded using a simple socket application program, with timeouts to account for busy networks or failed connections.

There are several hundred documents that mention events about appointments on the Web. To restrict the set retrieved to a manageable number, we searched the Web using the Alta Vista search expression shown below, where we require that the document retrieved contain the expression *Fortune 500 company* as well as the word *CEO*:

```
+appoint* +"Fortune 500 company" +CEO
```

Retrieval and Filtering Performance

A total of 100 URLs matched this query. Documents corresponding to 16 of these URLs were not retrieved due to problems not uncommon on the Web, such as network failure and sites that had moved. The 84 documents that were retrieved were hand-checked for relevance and 28 documents were found to be relevant. The 84 retrieved documents were also subjected to the filtering process described in Section 8.1.1. This classified 29 documents as relevant, of which 23 documents matched the hand-classified data. Tables 8.4 to 8.6 show the performance of the system in terms of Recall and Precision for relevant and irrelevant documents. The first row shows the performance of the web search engine by itself, while the second row shows the performance of Glean's filtering on the output of the web search engine. It is interesting to note that this method performs better in filtering out irrelevant documents than in identifying relevant documents.

As is seen from Tables 8.4 to 8.6, Glean filtering increases precision significantly. Compared to the number of sentences in the column under *Total Docs* (which is the number of documents retrieved by the plain search), the number of sentences marked relevant is about a third of the total number. The number of documents in this experiment is small, but we intend to collect similar figures for other experiments involving larger numbers of

System	Total Docs	Rel Docs	Classified as relevant			Classified as irrelevant		
			Total	Correct	Incorrect	Total	Correct	Incorrect
Plain Web Search (No Glean)	84	28	84	28	56	0	0	0
With Glean filtering	84	28	29	23	6	55	50	5

Table 8.4: Classification of the documents retrieved for the search query

System	Recall	Precision
Plain Web Search (Without Glean)	(28/28) = 100%	(28/84) = 33.3%
With Glean filtering	(23/28) = 82.1%	(23/29) = 79.3%

Table 8.5: Precision and Recall of Glean for retrieving relevant documents.

documents.

As briefly noted above, the performance of this mechanism is much better for filtering out irrelevant material than it is for identifying relevant material. This was also noticed in our experiments with New York Times data. There are many possible reasons for this, including extra-syntactic phenomena which we have not accounted for, inadequate size of window used in creating patterns, unusual patterns of word use, etc.

Note that errors in supertagging could also lead to wrong categorization of documents,

System	Recall	Precision
Plain Web Search (Without Glean)	–	–
With Glean filtering	(50/56) = 89.3%	(50/55) = 90.9%

Table 8.6: Precision and Recall of Glean for filtering out irrelevant documents

as could errors in spelling. Errors in supertagging during the creation of patterns may cause extremely specific patterns to be created, which may not be a serious problem, since these patterns are not likely to match any of the input.

We are addressing some of these problems in order to reduce the overall error rate. We are also testing the system with other domains and test areas, with very encouraging results for information refining.

8.2 Information Extraction

Supertagging has also been used for information extraction tasks. In contrast to information retrieval tasks, information extraction requires not only identifying the relevant sentences containing the information requested by the user but also requires to fill out the fields of an attribute-value matrix typically called a *template*. The attributes of a template vary based on the type of information summarized by the template. A template can be regarded as a summarized representation of the relevant document.

During the summer of 1996, an information extraction project that was sponsored by Lexis-Nexis was undertaken at the University of Pennsylvania. Two different kinds of templates were required to be filled: management changeovers[2] and new product introductions. The documents were drawn from a variety of news sources such as Reuters, Business wire, PR news, LA Times and San Fransisco Chronicle.

To do this task, we employed the EAGLE (An Extensible Architecture for General Linguistic Engineering) system[Baldwin *et al.*, 1996] which has been under development since 1995 at the University of Pennsylvania. The EAGLE system provides a flexible architecture to integrate a variety of Natural Language tools in novel ways for text processing. It contains document preprocessing tools for sentence segmentation, case normalization; syntactic analysis tools such as a morphological analyzer, a combination of three part-of-speech taggers, supertagger and statistical parser; and discourse processing tools for detecting coreference information. A detailed description of each of these components of the system is provided in [Baldwin *et al.*, 1996]. A pattern description language called

[2]The fields of this template were not identical to those in the corresponding MUC-6 template [Committee, 1995].

Mother of PERL (MOP) [Doran *et al.*, 1996] has also been developed in conjunction with the EAGLE system. MOP allows a user to specify patterns using the different levels of syntactic descriptions provided by the EAGLE system to spot information relevant to fill the fields of a template.

The supertagger is used in EAGLE as one of the two syntactic analyzers, the other being a statistical parser [Collins, 1996]. The objective of using two syntactic analyzers was to benefit from the strengths of the two different representations and approaches to syntactic processing. The supertagger was used to rapidly and robustly detect chunks such as Noun groups (both minimal and maximal Noun groups) and Verb groups in a document. These chunks were used by the Lightweight Dependency Analyzer to compute subject-verb-object triples for a sentence. MOP patterns based on the supertag annotation, the noun and verb chunk annotation and the subject-verb-object representation are specified to spot information that serve to fill the management and new product introduction templates.

An example MOP pattern that employs supertags and maximal Noun phrases is shown in Figure 8.3.

```
[maxNP;] ![] /``|''/ >0..3
st:/B_nxVpus|B_nx0Vs1/ ( that )
tok:/,|:/ `` [>>.. !~ /``/ ] ''  ->
quoted_speech1 [[ Comment Brackets1
                  Commenter Brackets0 ]] $$
```

Figure 8.3: A sample MOP pattern using Supertags and Maximal Noun Phrases

This is a pattern meant to match a text as follows:[3]

> Whitaker stated, "We are delighted to have this exciting product group as part of QSound and to further solidify the successful business relationships between SEGA and all facets of the QSound operation. We are in negotiations with the present AGES Direct management staff and expect to have them onboard with us immediately."

The pattern works as follows:

[3]We are thankful to Christine Doran for this example.

1. Match a maximal NP, which does not itself contain any quotation marks (`[maxNP;] ![] /‘‘|’’/`)

2. Go forward 0 to 3 tokens (`>0..3`)

3. Match one of the two supertags for a verb of saying in this position (`st:/B_nxVpus|B_nx0Vs1/`)

4. Optionally match the word *that* (`( that )`)

5. Match a comma or a colon, followed by an open quote (`tok:/,|:/ ‘‘`)

6. Match any number of words up to the end of the document, without crossing another open quote, until hitting a close quote (`[>>..  !  /‘‘/ ] ’’`)

This pattern makes use of various levels of representations. It uses tokens, for the obligatory punctuation marks, the maximal Noun Phrases for commenter, and the supertags to identify the relevant sub-class of main verbs. The supertags in particular offers just the right level of generalization, as listing the verbs one by one is not practical for a class of this size, and yet allowing any verb at all would be too permissive. The two pieces of information extracted by this pattern are the Commenter and the Comment itself.

EAGLE was evaluated on 153 documents from 23 different publications. A hand-analysis revealed the system's performance to be 84.6% precision and 45.6% recall on fields filled for the management changeover task. It should be noted that EAGLE was more consistent in filling fields than human annotators, and 12% of the templates filled were completely missed by human annotators on the first pass. No evaluation was conducted for new product announcements. Due to the complex nature of EAGLE, it was difficult to quantitatively measure the contribution of each component of EAGLE to the overall performance of the system.

8.3 Language Modeling

N-gram language models have proved to be very successful in the language modeling task. They are fast, robust and provide state-of-the-art performance that is yet to be

surpassed by more sophisticated models. Further, n-gram language models can be seen as weighted finite state automata which can be tightly integrated within speech recognition systems [Pereira and Riley, 1994]. However, these models fail to capture even relatively local dependencies that exist beyond the order of the model. We expect that the performance of a language model could be improved if these dependencies can be exploited. However, extending the order of the model to accommodate these dependencies is not practical since the number of parameters of the model is exponential in the order of the model, reliable estimation of which needs enormous amounts of training material.

In order to overcome the limitation of the n-gram language models and to exploit syntactic knowledge, many researchers have proposed structure-based language models [Zue *et al.*, 1991; Kenji and Ward, 1991]. Structure-based language models employ grammar formalisms richer than weighted finite-state grammars such as Probabilistic LR grammars, Stochastic Context-Free Grammars (SCFG), Probabilistic Link Grammars (PLG) [Lafferty *et al.*, 1992] and Stochastic Lexicalized Tree-Adjoining Grammars (SLTAGs) [Resnik, 1992; Schabes, 1992]. Formalisms such as PLGs and SLTAGs are more readily applicable for language modeling than SCFGs due to the fact that these grammars encode lexical dependencies directly. Although all these formalisms can encode arbitrarily long-range dependencies, tightly integrating these models with a speech recognizer is a problem since most parsers of these formalisms only accept and rescore N-best sentence lists. A recent paper [Jurafsky *et al.*, 1995] attempts at a tight integration of syntactic constraints provided by a domain-specific SCFG in order to work with lattice of word hypothesis. However, the integration is computationally expensive and the word lattice pruning is suboptimal. Also, most often the utterances in a spoken language system are ungrammatical and may not yield a full parse spanning the complete utterance.

Supertags present an alternate technique of integrating structural information into language models without really parsing the utterance. It brings together the advantages of a n-gram language model – speed, robustness and the ability to closely integrate with a speech recognizer.

In this section, we discuss a supertag-based n-gram language model that is similar to a class-based n-gram language model except that the classes are defined by the supertags. A related work that employs part-of-speech categories as classes is presented in [T.R.

Niesler and P.C. Woodland, 1996]. Since the supertags encode both lexical dependencies and syntactic and semantic constraints in a uniform representation, supertag-based classes are more fine-grained than part-of-speech based classes.

8.3.1 Performance Evaluation

The performance of a language model is ideally measured in terms of its ability to decrease the word error rate of a recognizer. However, to avoid this expensive metric, an alternate measure of performance, *perplexity (PP)* [Bahl *et al.*, 1977] is used:

(8.1) $PP = 2^{(-1/n)*log_2(Pr(W))}$

where

$Pr(W)$ is the probability of the word sequence W.

In a class-based n-gram language model, the probability of a k-word word sequence W is given by

(8.2) $Pr(W) = \sum_{c^k} \prod_{i=1}^{k} P(w_i|c_i)p(c_i|c_{i-2}, c_{i-1})$

where

$Pr(W)$ is the probability of the word sequence W.

Perplexity, roughly speaking, measures the average size of the set from which the next word is chosen from. The smaller the perplexity measure the better the language model at predicting the next word. The perplexity measure for a language model depends on the domain of discourse.

In this section, we present perplexity results for two class-based models on the the Wall Street Journal corpus: a part-of-speech based and a supertag-based trigram language model. A set of 200,000 words from the Wall Street Journal was randomly split into a set of 180,000 words for training and 20,000 words for the test corpus. The corpus was tagged with the correct part-of-speech tags[4] and supertags to train the part-of-speech based language model and supertagger based language model respectively. Then the performance of the two models[5] was measured on the test corpus. The results are shown in Table 8.7.

[4]The UPenn treebank tagset with 40 tags was used for this purpose

[5]Only the best class (POS or supertag) sequence was used in the perplexity calculation.

Model	Perplexity
Part-of-speech based trigram model	203
Supertag based trigram model	101

Table 8.7: Word perplexities for the Wall Street Journal Corpus using the part-of-speech and supertag based langauge models.

It is interesting to note that the performance of the supertag model is better than the part-of-speech model which suggests that the supertags define a better level of contextual generalization and provide more fine-grained classes than part-of-speech tags. Table 8.8 presents class perplexity results for the part-of-speech tag model and the supertag model. Class perplexity measures the average size of the set from which the next class is chosen from, given the previous history of the sentence. As can be seen, even though there is much higher ambiguity on a per word basis for supertags than for part-of-speech tags, the number of possible combinations for supertags is much more restricted than for part-of-speech tags.

Model	Perplexity
Part-of-speech	7.61
Supertag	6.23

Table 8.8: Class perplexities for the Wall Street Journal Corpus

8.4 Simplification

Long and complicated sentences prove to be a stumbling block for current Natural Language systems. These systems stand to gain from methods that syntactically simplify such sentences. In applications such as Machine Translation, simplification results in simpler sentence structures and reduced ambiguity. In Information Retrieval systems, retrieving information from texts with simplified sentences rather than with complex sentences can help improve the precision of the retrieval system. Simplification can also be looked at as a summarization process where complex sentences are 'summarized' by removing extraneous clauses.

Simplification is viewed as a two step process. The first stage identifies the structure of the sentence and the second stage applies the rules of simplification on the computed structure. These rules identify the positions at which a sentence can be split based on lexical and structural information surrounding that position. Obviously a parser could be used to obtain the complete structure of the sentence. However, full parsing is slow and prone to failure, especially on complex sentences. Supertagging provides an alternate method which is both fast and yields hierarchical structure (constituent information) that can be used for the purpose of simplification.

8.4.1 Simplification with Dependency links

The output provided by the dependency analyzer not only contains dependency links among words but also indicates the constituent structure as encoded by supertags. The constituent information is used to identify whether a supertag contains a clausal constituent and the dependency links are used to identify the span of the clause. Thus, embedded clauses can easily be identified and extracted, along with their arguments. Punctuation can be used to identify constituents such as appositives which can also be separated out. As with the finite-state approach, the resulting segments may be incomplete as independent clauses. If the segments are to be reassembled, no further processing need be done on them.

Figure 8.4 shows a rule for extracting relative clauses, in dependency notation. We first identify the relative clause tree (Z), and then extract the verb which anchors it along with all of its dependents. The right hand side shows the two resulting trees. The gap in the relative clause (Y) need only be filled if the clauses are not going to be recombined. Examples (8.3) and (8.4) show a sentence before and after this rule has applied.

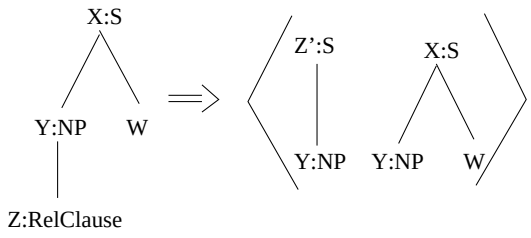


Figure 8.4: Rule for extracting relative clauses

(8.3) ... an issue [that **generated** unusual heat in the House of Commons] ...

(8.4) An issue [**generated** unusual heat in the House of Commons]. The issue ...

8.4.2 Evaluation and Discussion

Preliminary results using the model presented in the previous section are very promising. Using a corpus of Press Trust of India (PTI) newswire data, and only considering relative clause and appositive simplification, we correctly recovered 25 out of 28 relative clauses and 14 of 14 appositives. We generated 1 spurious relative clause and 2 spurious appositives.

Dependency-based simplification provides many advantages over simplification methods that employ Finite-State Grammar for the analysis of a complex sentence [Chandrasekar *et al.*, 1996]. Simplification rules in these methods manipulate noun groups and verb groups provided by the sentence analysis phase. As a result, the rules for the simplifier have to identify basic predicate argument relations to ensure that the right chunks remain together in the output. The simplifier in the Dependency-based simplification model has access to information about argument structure, which makes it much easier to specify simplification patterns involving complete constituents. The Finite-State Grammar based simplification approach is forced to enumerate all possible variants of the LHS of each simplification rule (eg. Subject versus Object relatives, singular versus plural NPs). In contrast, in the Dependency-based Simplification model, the rules exploit the argument/adjunct distinctions encoded in supertags and the associated constituent types and hence are more general.

We are presently working on automatically inducing simplification rules [Chandrasekar and Srinivas, 1996] given a parallel corpus of complex sentences and their simplified counterparts. The idea is to identify the points of divergences between the dependency structure for the complex sentence and the dependency structure for the simple sentences and to store that divergence information indexed on the surrounding context. In the test phase, given a complex sentence, the context information is used to retrieve the appropriate rule to be applied for simplification.

8.5 Exploiting Document-level Constraints for Supertagging

Both statistical parsing systems as well as hand-crafted grammar-based parsing systems model sentence level constraints and provide syntactic analysis of sentences. There are, however, other constraints that are present at the document level as well as domain specific constraints that are not integrated at the level of sentence processing. Integrating document level constraints into hand-crafted grammar-based parsing systems greatly complicates the model of processing and incorporating domain specific constraints into these systems requires elucidating the domain constraints in the grammar framework used by the system; a tedious task. For statistical parsing systems, domain constraints might be modeled by retraining the statistical model using data from that domain. However, this is a very expensive proposition since training data for any given domain is hard to collect. Further, integrating discourse level constraints increases the number of parameters and thus complicates the statistical model. Instead of a monolithic approach for integrating these constraints, we recommend a modular approach. In this section, we illustrate how the supertagging approach is amenable to a modular approach of incorporating document level and domain specific constraints.

The lexicalized nature of supertags combined with the fact that every syntactic context is represented by a different supertag, makes it is possible to bring to bear document level constraints as well as domain specific constraints when supertagging a sentence. To do this, we simply pre-supertag the words of the sentence that are influenced by the higher level constraints with a supertag that meets those constraints. For example, if we had an oracle that decided if a preposition attaches to the Verb Phrase or to the Noun Phrase, then the preposition could be pre-supertagged with the appropriate supertag that indicates the type of attachment.

Another example is to use coreference information to pre-supertag noun phrases [Srinivas and Baldwin, 1996]. An interesting aspect of the coreference system [Baldwin, 1995] is that there is an early pass of proper noun resolution that does not use any linguistic analysis other than the case of the words and what sentence they are in. The algorithm starts with a sparse matrix representation of all uppercase string matches, and attempts to

System	Total # of NPs	Recall	Precision
With Pretagging	235	91.0%	87.3%
Without Pretagging	235	87.9%	86.4%

Table 8.9: Performance results on extracting Noun Phrases with and without pre-supertagging.

grow the length of the matches while maintaining that they still match as a person name or company name. For example 'Smith' and 'Smith' could be expanded into 'Bernard J. Smith' and 'Mr. Smith' by the algorithm. Eventually the subroutine will consider matches of greater scope than person names and company names.

The pre-tagging afforded by the pre-linguistic coreference recognition was successful in improving the performance of the linguistically based Noun Phrase recognition component. The proper noun recognizer collapses all the lexical items in a proper noun into a single lexical item, so 'Federal Railway Labor Act' becomes pretagged as 'Federal_Railway_Labor_Act//A_NXN'. The advantage of collapsing the Noun Phrase is twofold. Besides improving the performance of the Noun Phrase recognition component, it allows the trigram model of supertagging to skip irrelevant words in the Noun Phrase and perhaps gain access to constraints that it otherwise would not reach given a three word window.

The pre-supertagging improved the performance of the Noun Phrase detection system. In particular, the recall of the system was helped by 3%, with no damage to precision. In looking at the output, it is clear that fairly simple steps could be made to improve performance further.

It is conceivable to have an architecture that employs experts of various specialties (domain, Simple Coreference, Noun group, PP etc.) which pretag words in a sentence with appropriately chosen supertags based on their information.

8.6 Summary

In this chapter, we discuss several applications of the supertagger and the LDA system including information retrieval and information extraction, text simplification and language modeling. Our goal in using a supertagger is to exploit the strengths of supertags as the appropriate level of lexical description needed for most applications. In an *information retrieval* application, we compare the retrieval efficiency of a system based on part-of-speech tags against a system based on supertags and show that the supertag-based system performs at higher levels of precision. In an *information extraction* task, supertags are used in specifying extraction patterns. For *language modeling* applications, we view supertags as syntactically motivated class labels in a class-based language model. The distinction between recursive and non-recursive supertags is exploited in *a sentence simplification application*. The ability to bring to bear document-level and domain-specific constraints during supertagging of a sentence is explored in improving the performance of a noun phrase recognizer.

Chapter 9

Conclusions

9.1 Contributions

In this dissertation, we have proposed novel methods for robust parsing that integrate the flexibility of linguistically motivated lexical descriptions with the robustness of statistical techniques. We have shown that the computation of linguistic structure can be localized if lexical items are associated with rich descriptions (*Supertags*) that impose complex constraints in a local context. The supertags are designed such that only those elements on which the lexical item imposes constraints appear within the same supertag. Further, each lexical item is associated with as many supertags as the number of different syntactic contexts in which the lexical item can appear in. This makes the number of different descriptions for each lexical item much larger, than when the descriptions are less complex; thus increasing the local ambiguity for a parser.

In Chapter 4, we have presented several models for disambiguating supertags that use statistical distributions of supertag co-occurrences, collected from a corpus of parses, to resolve this local ambiguity. We have provided extensive evaluation results for all the models for supertagging with the best rate of supertag accuracy of 92.2%. We have shown that using the supertagger as a preprocessor for a lexicalized grammar parser results in a speed-up of a factor of about 30 by simply reducing the search space for the parser even before parsing begins. We have also shown that besides being used as a preprocessor for a parser, a supertagger can also be used as a stand-alone parser. Supertag disambiguation

results in a representation that is effectively a parse (*almost parse*), and in conjunction with a lightweight dependency analyzer, a supertagger serves as an efficient and robust parser.

In Chapter 5 and Chapter 6, we have exploited the representation of the supertags in conjunction with Explanation-based learning (EBL) to improve the efficiency of parsing in limited domains. Language in limited domains is usually well constrained and certain syntactic patterns appear more frequently than others. EBL improves the parsing efficiency in such domains by relying on domain specific sentence constructions. In addition, by exploiting the representation of supertags in conjunction with EBL, parsing in limited domains can be modeled as a Finite-State Transducer. We have implemented such a system for the Air Travel Information Service (ATIS) domain which improves parsing efficiency by a factor of 15 over a system not tuned to the domain.

We have attempted to develop a uniform evaluation metric for parsers that are either statistical induced or grammar-based and produce full or partial parses. We advocate a two pronged evaluation metric, an application independent glass box evaluation metric and an application dependent black box evaluation metric. We argue that a mixed representation of text chunks and dependencies serves as a better level of representation for a glass box evaluation metric as opposed to a purely constituency based or a purely dependency based evaluation metric. For a black box evaluation, we propose that parsing systems should be measured on their ability to correctly identify various grammatical constructions and relations such as maximal noun phrases, appositives, predicative constructions, PP modifiers and Predicate-Argument relations. We obtained a recall and precision figures of 93.8% and 92.5% respectively for noun chunking, 87.2% and 92.2% for verb chunking, 84.5% and 62.3% for appositives, 75.5% and 85% for parentheticals and 81.1% accuracy for preposition phrase attachment using the supertagger in conjunction with the Lightweight dependency analyzer. In terms of performance of pairwise dependency links, we obtained a recall and precision of 82.3% and 93.8% on 47,000 words of Wall Street Journal corpus. At the sentence level, 89.8% of the sentences had three or less pairwise dependency link errors.

As mentioned before, we have demonstrated the usefulness of supertagger as a preprocessor for a parser by achieving a speedup of a factor of 30. Moreover, supertagger has

been applied in a variety of applications including information retrieval and information extraction, text simplification and language modeling. The supertagger in conjunction with the LDA has been used as a partial parser in information extraction applications. We have also used the fact that supertags provide richer and more fine-grained classes than part-of-speech tags in applications of information retrieval and language modeling.

9.2 Future Work

In this section, we present some of the directions for future work we intend to pursue.

Supertagging Models: A number of models for supertag disambiguation have been presented in Chapter 4 which vary in terms of the size of context that they take into account, such as the trigram model, the head trigram model and the dependency model and the representation for supertags – symbols or vector of features. Each of these models have different strengths and limitations depending on the nature of the application domain. We have begun to evaluate these models for their supertag accuracy. Further evaluation of these models in terms of accuracy of predicting specific constructions would be very useful. We would also like to apply more powerful statistical classification techniques such as statistical decision trees and maximum entropy models to supertagging.

Lightweight Dependency Analyzer (LDA): In Chapter 4, we presented LDA, a simple model for producing dependency analysis, once the supertagging is complete. The LDA uses the dependency requirements encoded in the supertags and some simple heuristics to produce a dependency analysis for a sentence. The deterministic nature of LDA crucially depends on the fact that each word is associated with only one supertag. However, the performance of the LDA can be improved if it is extended to deal with N-best supertags for each word.

Training Corpus: A distinct advantage for the statistical parsing research is the availability of large annotated corpora in the form of Penn Treebank and SUSANNE. The training material for supertag disambiguation models was obtained from converting the Penn Treebank parses using heuristics. There are several errors in the resulting supertag annotation, mostly due to errors of translation but also due to inadequacy of annotation in the Penn Treebank for this purpose. A more appropriate corpus would be to collect

LTAG derivation structures based on the XTAG grammar.

Balancing Act: One of the goals of this research has been to show that a hand-crafted grammar combined with domain specific statistics can perform as well as or better than a purely statistical grammarless parsing system. Besides the fine-grained nature of the output, we claim that the advantage of a hand-crafted grammar system is the ease of its portability to limited domains. We are in the process of performing experiments to establish this claim by porting the hand-crafted grammar to a weather reporting domain.

Psycholinguistic Implications: There has been increased emphasis on the role of lexical mechanisms in theories of sentence comprehension in the psycholinguistic literature [MacDonald *et al.*, 1994; Trueswell and Tanenhaus, 1994]. By providing multiple structural representation of words, these theories explain syntactic ambiguity resolution in term of lexical ambiguity resolution. This view is highly compatible with the view of Supertagging in lexicalized grammars. We have begun experiments to model supertag disambiguation using cognitively plausible architecture and processing models. The attempt is to see to what extent the model's processing preferences for syntactically ambiguous phrases mimic those found in human sentence processing.

Supertag-based Applications: Synchronous TAGs [Shieber and Schabes, 1990] provide a well-developed framework for Machine Translation. In this framework, elementary trees of the source language are mapped to elementary trees of the target language via synchronous mappings. Although this framework has been studied for its representational capacity, there has not been a robust translation system based on Synchronous TAGs. Supertagging in conjunction with synchronous mappings provides a natural direction for a robust translation system.

A number of applications reported in Chapter 8 have employed supertags as labels of richer classes than POS tags. This view has been very profitable but does not exploit the representation of supertags completely. The syntactic and the argument information encoded in the supertags is entirely ignored. We hope to improve the performance of some of the applications in Chapter 8 using the syntactic and argument information encoded in the supertags.

Applicability of Supertagging to other Lexicalized Grammars: As discussed in Chapter 3, the idea of supertagging has wider applicability than LTAGs. Any lexicalized

grammar assigns multiple descriptions to a lexical item, although only one (or relatively small number) of those descriptions would be used in the context of a given sentence. Thus the idea of supertagging, minimizing the ambiguity in terms of the number of descriptions assigned to a lexical item even before parsing begins, is applicable to other lexicalized grammars as well.

CCGs and LTAGs are similar in that they are both lexicalized grammars, however they differ in the extent of the domain of locality. There is on-going work on the development of a wide-coverage CCG system [Doran and Srinivas, 1994]. We intend to investigate the applicability of the partial parsing methods discussed here to CCG and study the tradeoffs adopted by CCG and LTAG with regard to notions such as constituency, dependency and locality.

Appendix A

List of Supertags

The following is a the list of supertags used in the experiments described in Chapter 4.

...EOS...	End of Sentence
A_ARB	Adverb as argument
A_AXA	Adjective as argument
A_AXAs	Adjective with sentence complement
A_CONJ	Conjunction as argument
A_D	Determiner as argument (in coordination)
A_N	Noun as argument (in coordination)
A_NXG	Genitive as argument
A_NXN	Noun phrase as argument
A_NXNs	Noun phrase with sentence complement
A_P	Preposition as argument (in coordination)
A_PL	Particles
A_PU	Punctuation
A_PXARBPnx	Complex preposition argument
A_PXP	Exhaustive Preposition argument
A_PXPNaPnx	Complex preposition argument
A_PXPPnx	Complex preposition argument
A_PXPnx	Preposition argument

A_by	*by* in passives
B_APnxs	Sentence initial complex preposition modifier
B_ARBCONJs	Sentence initial complex conjunction modifier
B_ARBPnxs	Sentence initial complex preposition modifier
B_ARBPs	Sentence initial complex preposition modifier
B_ARBa	Adverb modifying an adjective
B_ARBarb	Adverb modifying an adverb
B_ARBarbs	Sentence initial complex adverb modifier
B_ARBd	Adverb modifying a determiner
B_ARBn	Adverb modifying a noun
B_ARBnx	Adverb modifying a noun phrase
B_ARBpx	Adverb modifying a preposition phrase
B_ARBs	Adverb modifying a sentence
B_ARBvx	Adverb modifying a verb phrase
B_An	Adjective modifying a noun
B_COMPs	Complementizer
B_CONJACONJd	Complex determiner modifier
B_CONJDCONJd	Complex determiner modifier
B_CONJPnxs	Sentence initial complex preposition modifier
B_CONJarbCONJs	Sentence initial complex conjunction
B_CONJnx1conjnx2	Complex noun phrase conjunction
B_CONJs	Sentence initial conjunction
B_CONJs1conjs2	Complex sentence conjunction
B_CONJvx1conjvx2	Complex verb phrase conjunction
B_Dnx	Determiner
B_NEGarb	Negation on adverb
B_NEGnx	Negation on noun phrase
B_NEGpx	Negation on preposition phrase
B_NEGvx	Negation on verb phrase
B_Nn	Noun modifier
B_Ns	Temporal noun modifying a sentence

B_Nvx	Temporal noun modifying a verb phrase
B_PARBPnxs	Sentence initial complex preposition modifier
B_PARBarb	Complex preposition modifying an adverb
B_PARBd	Complex preposition modifying a determiner
B_PARBnx	Complex preposition modifying a noun phrase
B_PARBpx	Complex preposition modifying a preposition
B_PARBs	Complex preposition modifying a sentence
B_PARBvx	Complex preposition modifying a verb phrase
B_PDNPnxs	Sentence initial complex preposition modifier
B_PNPnxs	Sentence initial complex preposition modifier
B_PNaPnxs	Sentence initial complex preposition modifier
B_PPnxs	Sentence initial complex preposition modifier
B_PUapu	Punctuation around adjective
B_PUarbpu	Punctuation around adverb
B_PUnpu	Punctuation around noun
B_PUnxpu	Punctuation around noun phrase
B_PUpu	Punctuation on another punctuation
B_PUpx	Punctuation before preposition phrase
B_PUpxpu	Punctuation around preposition phrase
B_PUpxpunx	Punctuation around preposition phrase
B_PUs	Punctuation before a sentence
B_PUspu	Punctuation around a sentence
B_PUvpu	Punctuation around a verb
B_PUvxpu	Punctuation around a verb phrase
B_Pnxs	Sentence initial preposition modifier
B_UCONJUCONJs	Complex Sentence initial conjunction
B_Vn	Participle verb, prenominal modifier
B_Vnxpus	Parenthetical sentence complement verb
B_Vnxpuvx	Parenthetical sentence complement verb
B_Vnxs	Parenthetical sentence complement verb
B_Vs	Auxiliary inversion

B_Vvx	Auxiliary verb
B_a1CONJa2	Adjective coordination
B_aARB	Adverb modifying an adjective
B_arb1CONJarb2	Adverb coordination
B_arbARB	Adverb modifying an adverb
B_ax1CONJax2	Adjective phrase coordination
B_conjAd1conjAd2	Multi-anchor adverb coordination
B_d1CONJd2	Determiner coordination
B_n1CONJn2	Noun coordination
B_nA	Postnominal adjective modifier
B_nARB	Postnominal adverb modifier
B_nPUnx	Appositive for postal addresses without final punctuation
B_nPUnxpu	Appositives for postal addresses
B_nx1CONJARBCONJnx2	Multi-anchor noun phrase conjunction
B_nx1CONJARBnx2	Multi-anchor noun phrase conjunction
B_nx1CONJnx2	Noun phrase conjunction
B_nx1conjCONJnx2	Multi-anchor noun phrase conjunction
B_nxAPnx	Noun phrase modifying complex preposition
B_nxARB	Noun phrase modifying adverb
B_nxARBPnx	Noun phrase modifying complex preposition
B_nxGnx	Possessive *'s*
B_nxN	Noun phrase modifying time noun phrase
B_nxP	Noun phrase modifying exhaustive preposition
B_nxPDNPnx	Noun phrase modifying complex preposition
B_nxPNPnx	Noun phrase modifying complex preposition
B_nxPNaPnx	Noun phrase modifying complex preposition
B_nxPPnx	Noun phrase modifying complex preposition
B_nxPUnx	Appositives without final punctuation
B_nxPUnxpu	Appositives

B_nxPUpx	Noun phrase modifying preposition
B_nxPnx	Noun phrase modifying preposition
B_p1CONJp2	Preposition coordination
B_px1CONJARBCONJpx2	Multi-anchor preposition phrase coordination
B_px1CONJpx2	Preposition phrase coordination
B_pxARB	Preposition phrase modifying adverb
B_s1CONJs2	Sentence coordination
B_sARB	Sentence modifying adverb
B_sPU	Sentence final punctuation
B_sPUnx	Sentence modifying noun phrase
B_sPUs	Punctuation connecting sentences
B_sPnx	Sentence modifying noun phrase
B_v1CONJv2	verb coordination
B_vnxVpu	Parenthetical sentence complement verb
B_vx1CONJARBCONJvx2	Multi-anchor verb phrase coordination
B_vx1CONJvx2	Verb phrase coordination
B_vx1conjCONJvx2	Multi-anchor verb phrase coordination
B_vxAPnx	Verb phrase modifying complex preposition
B_vxARB	Post verbal adverbial modifier
B_vxARBPnx	Verb phrase modifying complex preposition
B_vxCONJPnx	Verb phrase modifying complex preposition
B_vxN	Verb phrase modifying time noun phrase
B_vxNPnx	Verb phrase modifying complex preposition
B_vxP	Verb phrase modifying exhaustive preposition
B_vxPDNPnx	Verb phrase modifying complex preposition
B_vxPNPnx	Verb phrase modifying complex preposition
B_vxPNaPnx	Verb phrase modifying complex preposition
B_vxPPnx	Verb phrase modifying complex preposition
B_vxPnx	Verb phrase modifying preposition

B_vxPs	Verb phrase modifying preposition with sentence complement
A_EW1nx1V	Wh-question on subject of ergative verb
A_Enx1V	Active ergative verb
A_Gnx0Ax1	Gerundive adjectival predicate
A_Gnx0V	Gerundive intransitive
A_Gnx0Varb1	Gerundive verb with adverb complement
A_Gnx0Vax1	Gerundive verb with adjective complement
A_Gnx0Vnx1	Gerundive transitive verb
A_Gnx0Vnx1nx2	Gerundive ditransitive verb
A_Gnx0Vnx1pl	Gerundive transitive particle verb (shifted)
A_Gnx0Vnx1s2	Gerundive verb with noun phase and sentence complement
A_Gnx0Vplnx1	Gerundive transitive particle verb (unshifted)
A_Gnx0Vpnx1	Gerundive verb with preposition complement
A_Gnx0Vs1	Gerundive verb with sentence complement
A_Inx0V	Imperative intransitive verb
A_Inx0Vax1	Imperative adjectival predicate
A_Inx0Vnx1	Imperative transitive
A_Inx0Vnx1pnx2	Imperative verb with noun phrase and preposition complement
A_Inx0Vnx1s2	Imperative verb with noun phrase and sentence complement
A_Inx0Vpl	Imperative intransitive verb particle
A_Inx0Vplnx1	Imperative transitive verb particle (unshifted)
A_W0nx0Vnx1	Wh-question on subject of transitive verb
A_W1nx0N1	Wh-question on predicate of a nominal predicate
A_nx0ARBPnx1	Complex preposition predicate
A_nx0Ax1	Adjective predicate
A_nx0Ax1pnx1	Adjective predicate with preposition complement
A_nx0Ax1s2	Adjective predicate with sentence complement
A_nx0BEnx1	Equative
A_nx0CONJPnx1	Complex preposition predicate

A_nx0N1	Nominal predicate
A_nx0N1s1	Nominal predicate with sentence complement
A_nx0PDNPnx1	Complex preposition predicate
A_nx0PNPnx1	Complex preposition predicate
A_nx0PNaPnx1	Complex preposition predicate
A_nx0PPnx1	Complex preposition predicate
A_nx0Pnx1	Preposition predicate
A_nx0Px1	Exhaustive Preposition predicate
A_nx0Px1s2	Preposition predicate with sentence complement
A_nx0V	Intransitive
A_nx0Varb1	Verb with adverb complement
A_nx0Vax1	Verb with adjective complement
A_nx0Vnx1	Transitive verb
A_nx0Vnx1Pnx2	Dative shifted double object verb
A_nx0Vnx1nx2	Ditransitive verb
A_nx0Vnx1pl	Transitive verb particle (shifted)
A_nx0Vnx1pnx2	Verb with noun phrase and preposition phrase complement
A_nx0Vnx2nx1	Ditransitive verb
A_nx0Vpl	Intransitive verb particle
A_nx0Vplnx1	Transitive verb particle (unshifted)
A_nx0Vplpnx1	Verb particle with preposition complement
A_nx0Vpnx1	Verb with preposition complement
B_nx0Vs1	Verb with sentence complement
A_nx0lVN1	Intransitive light verb
A_nx0lVN1Pnx2	Light verb with preposition complement
A_nx1V	Transitive passive verb with no by-phrase
A_nx1Vbynx0	Transitive passive verb with by-phrase
A_nx1Vnx2	Ditransitive passive verb with no by-phrase
A_nx1Vpl	Transitive passive verb particle with no by-phrase
A_nx1Vplbynx0	Transitive passive verb particle with by-phrase

A_nx1Vpnx2	Dative passive verb with no by-phrase
A_pW1nx0Pnx1	Wh-question on object of preposition predicate
A_s0N1	Sentence subject nominal predicate
B_Inx0Vs1	Imperative verb with sentence complement
B_N0nx0Ax1	Relative clause on subject of adjective predicate
B_N0nx0Ax1s2	Relative clause on subject of adjective predicate with sentence complement
B_N0nx0BEnx1	Relative clause on subject of equative
B_N0nx0N1	Relative clause on subject of nominal predicate
B_N0nx0Pnx1	Relative clause on subject of preposition predicate
B_N0nx0Px1	Relative clause on subject of exhaustive preposition predicate
B_N0nx0V	Relative clause on subject of intransitive verb
B_N0nx0Varb1	Relative clause on subject of verb with adverb complement
B_N0nx0Vax1	Relative clause on subject of verb with adjective complement
B_N0nx0Vnx1	Relative clause on subject of transitive verb
B_N0nx0Vnx1Pnx2	Relative clause on subject of dative verb
B_N0nx0Vnx1arbx2	Relative clause on subject of verb with noun phrase and adverb complement
B_N0nx0Vnx1nx2	Relative clause on subject of double object verb
B_N0nx0Vnx1pl	Relative clause on subject of transitive verb particle (unshifted)
B_N0nx0Vnx1pnx2	Relative clause on subject of verb with noun phrase and preposition phrase complement
B_N0nx0Vnx1s2	Relative clause on subject of verb with noun phrase and sentence complement
B_N0nx0Vnx2nx1	Relative clause on subject of double object verb
B_N0nx0Vpl	Relative clause on subject of intransitive verb particle
B_N0nx0Vplnx1	Relative clause on subject of transitive verb particle
B_N0nx0Vplpnx1	Relative clause on subject of verb particle with preposition complement

B_N0nx0Vpnx1	Relative clause on subject of verb with preposition phrase complement
B_N0nx0Vs1	Relative clause on subject of verb with sentence complement
B_N1nx0BEnx1	Relative clause on object of equative verb
B_N1nx0Vnx1	Relative clause on object of transitive verb
B_N1nx0Vnx1Pnx2	Relative clause on object of dative verb
B_N1nx0Vnx1nx2	Relative clause on object of double object verb
B_N1nx0Vnx1pnx2	Relative clause on object of verb with noun phrase and preposition phrase complement
B_N1nx0Vnx1s2	Relative clause on object of verb with noun phrase and sentence complement
B_N1nx0Vplnx1	Relative clause on object of transitive verb particle
B_N1nx0Vpnx1	Relative clause on object of verb preposition complement
B_N1nx1V	Relative clause on object of passive transitive verb
B_N1nx1VPnx2	Relative clause on object of passive dative verb
B_N1nx1Vax2	Relative clause on object of passive verb with adjective complement
B_N1nx1Vbynx0	Relative clause on object of passive transitive verb with by-phrase
B_N1nx1Vnx2	Relative clause on object of passive double object verb
B_N1nx1Vpl	Relative clause on object of passive verb particle
B_N1nx1Vpnx2	Relative clause on object of passive verb with preposition phrase complement
B_N1nx1Vs2	Relative clause on object of passive verb with sentence complement
B_N2nx0Vnx1pnx2	Relative clause on indirect object of dative verb
B_N2nx1Vpnx2	Relative clause on indirect object of passive dative verb
B_Nnx0Ax1	Adjunct relative clause for an adjective predicate
B_Nnx0BEnx1	Adjunct relative clause for an equative
B_Nnx0N1	Adjunct relative clause for a nominal predicate

B_Nnx0Pnx1	Adjunct relative clause for a prepositional predicate
B_Nnx0V	Adjunct relative clause for an intransitive
B_Nnx0Vnx1	Adjunct relative clause for a transitive
B_Nnx0Vnx1nx2	Adjunct relative clause for double object verb
B_Nnx0Vnx1pl	Adjunct relative clause for a transitive verb particle
B_Nnx0Vnx1s2	Adjunct relative clause for a verb with noun phrase and sentence complement
B_Nnx0Vpl	Adjunct relative clause for a verb particle
B_Nnx0Vpnx1	Adjunct relative clause for a verb with preposition phrase complement
B_Nnx0Vs1	Adjunct relative clause for a verb with sentence complement
B_Nnx1V	Adjunct relative clause for a passive transitive verb
B_Nnx1Vpnx2	Adjunct relative clause for a passive verb with preposition phrase complement
B_Nnx1Vs2	Adjunct relative clause for a passive verb with sentence complement
B_nx0A1s1s	Presentential clausal adjunct of adjective predicate with sentence complement
B_nx0Ax1s	Presentential clausal adjunct of adjective predicate
B_nx0Vnx1pnx2s	Presentential clausal adjunct of dative verb
B_nx0Vnx1s	Presentential clausal adjunct of transitive verb
B_nx0Vpls1	Presentential clausal adjunct of intransitive verb particle
B_nx0Vpnx1s	Presentential clausal adjunct of transitive verb particle
B_nx0Vs	Presentential clausal adjunct of intransitive verb
B_nx0Vs1	Verb with sentence complement
B_nx1Vbynx0s	Presentential clausal adjunct of passive transitive verb with by-phrase
B_nx1Vps	Presentential clausal adjunct of passive verb with preposition complement

B_nx1Vs	Presentential clausal adjunct of passive verb
B_nx1Vs2	Passive verb with sentence complement
B_nxVnxpus	Parenthetical sentence complement verb
B_nxVpus	Parenthetical sentence complement verb
B_nxVpuvx	Parenthetical sentence complement verb
B_nxVs	Parenthetical sentence complement verb
B_pN1nx0Vpnx1	Relative clause on the object of verb with preposition complement
B_pN2nx1Vpnx2	Relative clause on the indirect object of dative verb
B_sVnx	Parenthetical sentence complement verb
B_snxV	Parenthetical sentence complement verb
B_snxVnx	Parenthetical sentence complement verb
B_vxnx0ARBPnx1	Postsentential clausal adjunct of complex preposition predicate
B_vxnx0Ax1	Postsentential clausal adjunct of adjective predicate
B_vxnx0V	Postsentential clausal adjunct of intransitive verb
B_vxnx0Vax1	Postsentential clausal adjunct of verb with adjective complement
B_vxnx0Vnx1	Postsentential clausal adjunct of transitive verb
B_vxnx0Vnx1nx2	Postsentential clausal adjunct of double object verb
B_vxnx0Vnx1pl	Postsentential clausal adjunct of transitive verb particle (shifted)
B_vxnx0Vpnx1	Postsentential clausal adjunct of verb with preposition phrase complement
B_vxnx1V	Postsentential clausal adjunct of passive transitive verb
B_vxnx1Vbynx0	Postsentential clausal adjunct of passive transitive verb with by-phrase
B_vxnx1Vnx2	Postsentential clausal adjunct of passive ditransitive verb
B_vxnx1Vpnx2	Postsentential clausal adjunct of passive dative verb

Bibliography

[Abney, 1990a] Steven Abney. Rapid Incremental parsing with repair. In *Proceedings of the 6th New OED Conference: Electronic Text Research*, pages 1–9, University of Waterloo, Waterloo, Canada, 1990.

[Abney, 1990b] Steven P. Abney. Rapid Incremental Parsing with Repair. In *Proceedings of the 6^{th} New OED Conference*, University of Waterloo, Waterloo, Ontario, 1990.

[Abney, 1991] Steven Abney. Parsing by chunks. In Robert Berwick, Steven Abney, and Carol Tenny, editors, *Principle-based parsing*. Kluwer Academic Publishers, 1991.

[Abney, 1994a] Steven Abney. Dependency Grammars and Context-Free Grammars. Manuscript, University of Tubingen, March 1994.

[Abney, 1994b] Steven Abney. Partial Parsing. Tutorial given at ANLP-94, Stuttgart, October 1994.

[Alshawi and Carter, 1994] Hiyan Alshawi and David Carter. Training and scaling preference functions for disambiguation. *Computational Linguistics*, 20(4), 1994.

[Alshawi *et al.*, 1992] Hiyan Alshawi, David Carter, Richard Crouch, Steve Pullman, Manny Rayner, and Arnold Smith. *CLARE – A Contextual Reasoning and Cooperative Response Framework for the Core Language Engine*. SRI International, Cambridge, England, 1992.

[Anttila, 1994] A. Anttila. *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*, chapter How to recognize subjects in English. Mourton de Gruyter, Berlin and New York, 1994.

[Appelt *et al.*, 1993] D. Appelt, J. Hobbs, J. Bear, D. J. Israel, and M. Tyson. FASTUS: a finite-state processor for information extraction from real-world text. In *Proceedings of IJCAI-93*, Chambery, France, September 1993.

[Bahl *et al.*, 1977] L.R. Bahl, J.K. Baker, F. Jelinek, and R.L. Mercer. Perplexity - a measure of the difficulty of speech recognition tasks. *Program of the 94th Meeting of the Acoustical Society of America J. Acoust. Soc. Am.*, 62, 1977.

[Baldwin *et al.*, 1996] Breckenridge Baldwin, Christine Doran, Jeffrey Reynar, Michael Niv, B. Srinivas, and Mark Wasson. EAGLE: An Extensible Architecture for General Linguistic Engineering. Manuscript, Department of Computer and Information Sciences, University of Pennsylvania, 1996.

[Baldwin, 1995] F. Breckenridge Baldwin. *CogNIAC – A Discourse Processing Engine.* PhD thesis, Department of Computer and Information Sciences, University of Pennsylvania, 1995.

[Black *et al.*, 1993a] Ezra Black, R. Garside, and G. Leech (eds.). *Statistically-driven computer grammars of English: The IBM/Lancaster approach.* Rodopi, Amsterdam, 1993.

[Black *et al.*, 1993b] Ezra Black, Fred Jelinek, John Lafferty, David M. Magerman, Robert Mercer, and Salim Roukos. Towards History-based Grammars: Using Richer Models for Probabilistic Parsing. In *Proceedings of the* 31^{st} *Conference of Association of Computational Linguistics*, 1993.

[Bod, 1995] Rens Bod. *Enriching Linguistics with Statistics: Performance Models of Natural Language.* PhD thesis, ILLC Disertation Series 1995-14, University of Amsterdam, 1995. ftp://ftp.fwi.uva.nl/pub/theory/illc/dissertations/DS-95-14.text.ps.gz.

[Brill and Resnik, 1994] E. Brill and P. Resnik. A rule-based approach to prepositional phrase attachment disambiguation. In *Proceedings of the International Conference on Computational Linguistics (COLING '94)*, Kyoto, Japan, 1994.

[Brill, 1993] Eric Brill. Automatic grammar induction and parsing free text: A transformation-based approach. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, Columbus, Ohio, 1993.

[Briscoe and Carroll, 1995] Ted Briscoe and John Carroll. Developing and Evaluating a Probabilistic LR Parser of Part-of-Speech and Punctuation Labels. In *Proceedings of the Fourth International Workshop on Parsing Technologies (IWPT95)*, Prague, Czech Republic, 1995.

[Briscoe *et al.*, 1996] Ted Briscoe, John Carroll, Nicoletta Calzolari, Stefano Federici, Simonetta Montemagni, Vito Pirrelli, Greg Grefenstette, Antonio Sanfilippo, Glenn Carroll, and Mats Rooth. Shallow parsing and knowledge extraction for language engineering – work package 1. Specification of Phrasal Parsing, Prefinal Report, May 1996.

[Carroll, 1993] John Carroll. *Practical Unification-based Parsing of Natural Language*. University of Cambridge, Computer Laboratory, Cambridge, England, 1993.

[Chandrasekar and Srinivas, 1996] R. Chandrasekar and B. Srinivas. Using syntactic information in document filtering: A comparative study of part-of-speech tagging and supertagging. Technical Report IRCS 96–29, University of Pennsylvania, 1996.

[Chandrasekar *et al.*, 1996] R. Chandrasekar, Christine Doran, and B. Srinivas. Motivations and methods for text simplification. In *Proceedings of the Sixteenth International Conference on Computational Linguistics (COLING '96)*, Copenhagen, Denmark, August 1996.

[Chomsky, 1992] Noam Chomsky. A Minimalist Approach to Linguistic Theory. *MIT Working Papers in Linguistics*, Occasional Papers in Linguistics No. 1, 1992.

[Church, 1988] Kenneth Ward Church. A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. In *2nd Applied Natural Language Processing Conference*, Austin, Texas, 1988.

[Chytil and Karlgren, 1988] Michal P. Chytil and Hans Karlgren. Categorial grammars and list automata for strata of non-cf languages. In Wojciech Buszkowski, Witold Marciszewski, and Johan van Benthem, editors, *Categorial Grammar*. Benjamin Cummings, Philadelphia and Amsterdam, 1988.

[Cole *et al.*, 1996] Ronald A. Cole, Joseph Mariani, Hans Uszkoreit, Annie Zaenen, and Victor Zue. Survey of the state of the art in human language technology, 1996. http://www.cse.ogi.edu/CSLU/HLTsurvey/.

[Collins and Brook, 1995] Michael Collins and James Brook. Prepositional phrase attachment through a backed-off model. In *Proceedings of the Third Workshop on Very Large Corpora*, MIT, Cambridge, Boston, 1995.

[Collins, 1996] Michael Collins. A New Statistical Parser Based on Bigram Lexical Dependencies. In *Proceedings of the 34^{th} Annual Meeting of the Association for Computational Linguistics*, Santa Cruz, 1996.

[Committee, 1995] The MUC-6 Program Committee. The information extraction task definition, v2.1. In *Proceedings of the Sixth Message Understanding Conference*, 1995.

[Croft *et al.*, 1991] Bruce W. Croft, Howard R. Turtle, and David D. Lewis. The use of phrases and structured queries in information retrieval. In *Proceedings of the 14^{th} Annual International Conference on Research and Development in Information Retrieval (SIGIR '91)*, pages 32–45, Chicago, USA, 1991.

[Doran and Srinivas, 1994] Christine Doran and B. Srinivas. A Wide-Coverage CCG Parser. In *Proceedings of the 3rd TAG+ Conference*, Paris, France, 1994.

[Doran *et al.*, 1994] Christy Doran, Dania Egedi, Beth Ann Hockey, B. Srinivas, and Martin Zaidel. XTAG System - A Wide Coverage Grammar for English. In *Proceedings of the 17^{th} International Conference on Computational Linguistics (COLING '94)*, Kyoto, Japan, August 1994.

[Doran *et al.*, 1996] Christine Doran, Michael Niv, Breckenridge Baldwin, Jeffrey Reynar, and B. Srinivas. Mother of Perl: A Multi-tier Pattern Description Language. Manuscript, Department of Computer and Information Sciences, University of Pennsylvania, 1996.

[Doran, 1996] Christy Doran. Punctuation in Quoted Speech. In *Proceedings of the SIGPARSE96*, Santa Cruz, California, June 1996.

[Frakes and Baeza-Yates, 1992] W. B. Frakes and R. S. Baeza-Yates. *Information Retrieval: Data Structures and Algorithms.* Prentice Hall, 1992.

[Fujisaki *et al.*, 1989] T. Fujisaki, Jelinek J. Cocke, E. Black, and T. Nishino. A Probabilistic Parsing Method for Sentence Disambiguation. In *Proceedings of the* 1^{st} *Annual International Workshop of Parsing Technologies*, Pittsburgh, 1989.

[Gazdar *et al.*, 1985] G. Gazdar, E. Klein, G. Pullum, and I. Sag. *Generalized Phrase Structure Grammar.* Harvard University Press, Cambridge, Massachusetts, 1985.

[Gee and Grosjean, 1983] James Gee and François Grosjean. Performance structures: a psycholinguistic and linguistic appraisal. *Cognitive Psychology*, 15:411–458, 1983.

[Good, 1953] I.J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika 40 (3 and 4)*, 1953.

[Grishman, 1995] Ralph Grishman. Where's the Syntax? The New York University MUC-6 System. In *Proceedings of the Sixth Message Understanding Conference*, Columbia, Maryland, 1995.

[Gross, 1984] Maurice Gross. Lexicon-Grammar and the Syntactic Analysis of French. In *Proceedings of the* 10^{th} *International Conference on Computational Linguistics (COLING'84)*, Stanford, California, 1984.

[Grover *et al.*, 1993] Claire Grover, John Carroll, and Ted Briscoe. *The Alvey Natural Language Tools Grammar*, 4th release edition, 1993.

[Harris, 1962] Zelig Harris. *String Analysis of Language Structure.* Mouton and Co., The Hague, Netherlands., 1962.

[Harrison *et al.*, 1991] P. Harrison, S. Abney, D. Fleckenger, C. Gdaniec, R. Grishman, D. Hindle, B. Ingria, M. Marcus, B. Santorini, and T. Strzalkowski. Evaluating syntax performance of parser/grammars of English. In *Proceedings of the Workshop on Evaluating Natural Language Processing Systems, ACL.*, 1991.

[Hindle and Rooth, 1991] Don Hindle and Mats Rooth. Structural ambiguity and lexical relations. In *29th Meeting of the Association for Computational Linguistics*, Berkeley, CA, 1991.

[Hindle, 1983] D. Hindle. Deterministic Parsing of Syntactic Non-Fluencies. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, 1983.

[Hindle, 1993] D. Hindle. Prediction of lexicalized tree fragments in text. In *ARPA Workshop on Human Language Technology*, March 1993.

[Hobbs and Bear, 1994] Jerry R. Hobbs and John Bear. Two Principles of Parse Preference. In *Current Issues in Natural Language Processing: In Honor of Don Walker*. Giardini with Kluwer, 1994.

[Hobbs *et al.*, 1991] Jerry Hobbs, Douglas E. Appelt, John S. Bear, Mabry Tyson, and David Magerman. The TACITUS system: The MUC-3 experience. Technical report, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, October 1991.

[Hobbs *et al.*, 1992] Jerry Hobbs, Doug Appelt, John Bear, David Israel, and W. Mary Tyson. FASTUS: a system for extracting information from natural language text. Technical Report 519, SRI, 1992.

[Hobbs *et al.*, 1995] Jerry R. Hobbs, Douglas E. Appelt, John Bear, David Israel, Andy Kehler, Megumi Kamayama, David Martin, Karen Myers, and Marby Tyson. SRI International FASTUS system MUC-6 test results and analysis. In *Proceedings of the Sixth Message Understanding Conference*, Columbia, Maryland, 1995.

[Jelinek *et al.*, 1994] Fred Jelinek, John Lafferty, David M. Magerman, Robert Mercer, Adwait Ratnaparkhi, and Salim Roukos. Decision Tree Parsing using a Hidden Derivation Model. In *Proceedings from the ARPA Workshop on Human Language Technology Workshop*, March 1994.

[Jensen *et al.*, 1993] Karen Jensen, George E.Heidorn, and Stephen D. Richardson. *Natural Language Processing: The PLNLP Approach.* Kluwer Academic Publishers, Boston/Dordrecht/London, 1993.

[Jones and Galliers, 1995] Karen Sparck Jones and Julia R. Galliers. *Evaluating natural language processing systems : an analysis and review.* Number 1083 in Lecture notes in computer science. Lecture notes in artificial intelligence. Springer, Berlin ; New York, 1995.

[Joshi and Hopely, 1997] Aravind Joshi and Philip Hopely. A parser from antiquity. *Natural Language Engineering*, 2(4), 1997.

[Joshi and Kulick, 1997] Aravind Joshi and Seth Kulick. Partial proof trees as building blocks for a categorial grammar. *Linguistics and Philosophy*, 1997. To appear.

[Joshi and Schabes, 1996] Aravind Joshi and Yves Schabes. *Handbook of Formal Lanaguages and Automata*, chapter Tree-Adjoining Grammars. Springer-Verlag, Berlin, 1996.

[Joshi and Srinivas, 1994] Aravind K. Joshi and B. Srinivas. Disambiguation of Super Parts of Speech (or Supertags): Almost Parsing. In *Proceedings of the* 17^{th} *International Conference on Computational Linguistics (COLING '94)*, Kyoto, Japan, August 1994.

[Joshi *et al.*, 1975] Aravind K. Joshi, L. Levy, and M. Takahashi. Tree Adjunct Grammars. *Journal of Computer and System Sciences*, 1975.

[Joshi *et al.*, 1990] Aravind K. Joshi, K. Vijay-Shanker, and David Weir. The Convergence of Mildly Context Sensitive Grammatical Formalisms. In Peter Sells, Stuart Shieber, and Tom Wascow, editors, *Foundational Issues in Natural Language Parsing.* MIT Press, Cambridge, Massachusetts, 1990.

[Joshi, 1960] Aravind K. Joshi. *Advances in Documentation and Library Science*, volume III, Part 2, chapter Computation of Syntactic Structure. Interscience Publishers, Inc., New York, 1960.

[Joshi, 1985] Aravind K. Joshi. Tree Adjoining Grammars: How much context Sensitivity is required to provide a reasonable structural description. In D. Dowty, I. Karttunen, and A. Zwicky, editors, *Natural Language Parsing*, pages 206–250. Cambridge University Press, Cambridge, U.K., 1985.

[Joshi, 1987] A. K. Joshi. An introduction to tree adjoining grammars. In A. Manaster-Ramer, editor, *Mathematics of Language*. John Benjamins, Amsterdam, 1987.

[Jurafsky *et al.*, 1995] D. Jurafsky, Chuck Wooters, Jonathan Segal, Andreas Stolcke, Eric Fosler, Gary Tajchman, and Nelson Morgan. Using a Stochastic CFG as a Language Model for Speech Recognition. In *Proceedings, IEEE ICASSP*, Detroit, Michigan, 1995.

[Kaplan and Bresnan, 1983] Ronald Kaplan and Joan Bresnan. Lexical-functional Grammar: A Formal System for Grammatical Representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, Massachusetts, 1983.

[Karlsson *et al.*, 1994] Karlsson, Voutilainen, Heikkilä, and Anttila. *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*. Mourton de Gruyter, Berlin and New York, 1994.

[Katz, 1987] Slava M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, speech and SignalProcessing*, 35(3):400–401, 1987.

[Kenji and Ward, 1991] Kita Kenji and Wayne Ward. Incorporating LR Parsing into SPHINX. In *Proceedings, IEEE ICASSP*, 1991.

[Kroch and Joshi, 1985] Anthony S. Kroch and Aravind K. Joshi. The Linguistic Relevance of Tree Adjoining Grammars. Technical Report MS-CIS-85-16, Department of Computer and Information Science, University of Pennsylvania, 1985.

[Kuno, 1966] S. Kuno. Harvard predictive analyzer. In David G.Hays, editor, *Readings in automatic language processing*. American Elsevier Pub. Co., New York, 1966.

[Lafferty *et al.*, 1992] John Lafferty, Daniel Sleator, and Davy Temperley. Grammatical Trigrams: A Probabilistic Model of Link Grammar. Technical Report CMU-CS-92-181, School of Computer Science, Carnegie Mellon University, 1992.

[Lin, 1995] Dekang Lin. A Dependency-based Method for Evaluating Broad-Coverage Parsers. In *Proceedings of IJCAI-96*, Montreal, Canada, August 1995.

[Lin, 1996] Dekang Lin. Evaluation of principar with the susanne corpus. In *Proceedings of the Workshop on Robust Parsing at European Summer School in Logic, Language and Information*, Prague, August 1996.

[MacDonald *et al.*, 1994] Maryellen MacDonald, Neal Pearlmutter, and Mark Seidenberg. The lexical nature of syntactic ambiguity resolution. *Psychological Review*, 101:676–703, 1994.

[Magerman, 1995] David M. Magerman. Statistical Decision-Tree Models for Parsing. In *Proceedings of the 33^{rd} Annual Meeting of the Association for Computational Linguistics*, 1995.

[Marcken, 1990] Carl G. De Marcken. Parsing the LOB Corpus. In 28^{th} *Meeting of the Association for Computational Linguistics*, 1990.

[Marcus *et al.*, 1993] Mitchell M. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19.2:313–330, June 1993.

[Marcus, 1983] Mitchell P. Marcus. *A theory of syntactic recognition for natural language*. Cambridge Massachusetts and London, England, The MIT Press, 1983.

[McDonald, 1993] David McDonald. Sparser. In *Text-based Intelligent Systems*. Hillsdale, N.J. : L. Erlbaum Associates, 1993.

[Minton, 1988] Steve Minton. Quantitative Results concerning the utility of Explanation-Based Learning. In *Proceedings of 7^{th} AAAI Conference*, pages 564–569, Saint Paul, Minnesota, 1988.

[Mitchell *et al.*, 1986] Tom M. Mitchell, Richard M. Keller, and Smadar T. Kedar-Carbelli. Explanation-Based Generalization: A Unifying View. *Machine Learning 1*, 1:47–80, 1986.

[Nagao, 1994] Makoto Nagao. Varieties of Heuristics in Sentence Processing. In *Current Issues in Natural Language Processing: In Honour of Don Walker*. Giardini with Kluwer, 1994.

[Neumann, 1994] Günter Neumann. Application of Explanation-based Learning for Efficient Processing of Constraint-based Grammars. In 10^{th} *IEEE Conference on Artificial Intelligence for Applications*, San Antonio, Texas, 1994.

[Ney *et al.*, 1995] Herman Ney, Ute Essen, and Reinhard Kneser. On the estimation of 'small' probabilities by leaving-one-out. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(2), 1995.

[Oepen *et al.*, forthcoming] Stephan Oepen, Klaus Netter, and Judith Klein. *TSNLP – Test Suites for Natural Language Processing.* CSLI Lecture Notes, forthcoming. http://cl-www.dfki.uni-sb.de/tsnlp/.

[Pereira and Riley, 1994] Fernando C.N. Pereira and Michael D. Riley. Speech recognition by composition of weighted finite automata. In *Proceedings of the ARPA Workshop on Human Language Technology Workshop*, March 1994.

[Pollard and Sag, 1987] Carl Pollard and Ivan A. Sag. *Information-Based Syntax and Semantics. Vol 1: Fundamentals.* CSLI, 1987.

[Rambow and Joshi, 1993] O. Rambow and A.K. Joshi. Dependency parsing for phrase-structure grammars. University of Pennsylvania, 1993.

[Rambow *et al.*, 1995] Owen Rambow, David Weir, and K. Vijay-Shanker. D-Tree Grammars. In *Proceedings of the* 33^{rd} *Conference of Association of Computational Linguistics*, 1995.

[Ramshaw and Marcus, 1995] Lance Ramshaw and Mitchell P. Marcus. Text chunking using transformation-based learning. In *Proceedings of the Third Workshop on Very Large Corpora*, MIT, Cambridge, Boston, 1995.

[Ratnaparkhi *et al.*, 1994] A. Ratnaparkhi, J. Reynar, and S. Roukos. A maximum entropy model for prepositional phrase attachment. In *Proceedings of ARPA Workshop on Human Language Technology*, Plainsboro, NJ, March 1994.

[Ratnaparkhi, 1996] Adwait Ratnaparkhi. A Maximum Entropy Part-of-speech Tagger. In *Proceedings of the Emperical Methods in Natural Language Processing Conference*, Philadelphia, 1996.

[Rayner, 1988] Manny Rayner. Applying Explanation-Based Generalization to Natural Language Processing. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, Tokyo, 1988.

[Resnik, 1992] Philip Resnik. Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *Proceedings of the Fourteenth International Conference on Computational Linguistics (COLING '92)*, Nantes, France, July 1992.

[Robinson, 1981] Jane Robinson. Perspectives on parsing issues. In *Proceedings of ACL '81*, pages 95–106, 1981.

[Roche, 1993] Emmanuel Roche. *Analyse syntaxique transformationelle du francais par transducteurs et lexique-grammaire.* PhD thesis, Universite Paris 7., 1993.

[Salton and McGill, 1983] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, 1983.

[Sampson *et al.*, 1989] G. Sampson, R. Haigh, and E. Atwell. Natural language analysis by stochastic optimization: a progress report on project april. *Journal of Experimental and Theoretical Artificial Intelligence*, 1:271–287, 1989.

[Sampson, 1994] G. Sampson. SUSANNE: a Doomsday book of English Grammar. In *Corpus-based Research into Language.* Rodopi, Amsterdam, 1994.

[Samuelsson and Rayner, 1991] Christer Samuelsson and Manny Rayner. Quantitative Evaluation of Explanation-Based Learning as an Optimization Tool for Large-Scale Natural Language System. In *Proceedings of the 12^{th} International Joint Conference on Artificial Intelligence*, Sydney,Australia, 1991.

[Samuelsson, 1994] Christer Samuelsson. Grammar Specialization through Entropy Thresholds. In *32nd Meeting of the Association for Computational Linguistics*, Las Cruces, New Mexico, 1994.

[Scha, 1990] Remko Scha. Taaltheorie en taalthechnologie; competence en performance. In Q.A.M. de Kort and G.L.J. Leerdam, editors, *Computertoepassingen in de Neerlanistiek*. Almere: Landelijke Verening van Neerlandici, 1990.

[Schabes and Joshi, 1991] Yves Schabes and Aravind K. Joshi. Parsing with Lexicalized Tree Adjoining Grammar. In M. Tomita, editor, *Current Issues in Parsing Technologies*. Kluwer Academic Publishers, 1991.

[Schabes and Shieber, 1992] Yves Schabes and Stuart Shieber. An Alternative Conception of Tree-Adjoining Derivation. In *Proceedings of the* 20^{th} *Meeting of the Association for Computational Linguistics*, 1992.

[Schabes *et al.*, 1988] Yves Schabes, Anne Abeillé, and Aravind K. Joshi. Parsing strategies with 'lexicalized' grammars: Application to Tree Adjoining Grammars. In *Proceedings of the* 12^{th} *International Conference on Computational Linguistics (COLING'88)*, Budapest, Hungary, August 1988.

[Schabes *et al.*, 1993] Y. Schabes, M. Roth, and R. Osborne. Parsing the Wall Street Journal with the Inside-Outside Algorithm. In *Proceedings of the European ACL*, 1993.

[Schabes, 1990] Yves Schabes. *Mathematical and Computational Aspects of Lexicalized Grammars*. PhD thesis, Computer Science Department, University of Pennsylvania, 1990.

[Schabes, 1992] Yves Schabes. Stochastic lexicalized tree-adjoining grammars. In *Proceedings of the Fourteenth International Conference on Computational Linguistics (COLING '92)*, Nantes, France, July 1992.

[Seneff, 1992] Stephanie Seneff. A relaxation method for understanding spontaneous speech utterances. In *Proceedings, Speech and Natural Language Workshop*, San Mateo, CA, 1992.

[Shieber and Schabes, 1990] Stuart Shieber and Yves Schabes. Synchronous Tree Adjoining Grammars. In *Proceedings of the* 13^{th} *International Conference on Computational Linguistics (COLING'90)*, Helsinki, Finland, 1990.

[Sleator and Temperley, 1991] Daniel Sleator and Davy Temperley. Parsing English with a Link Grammar. *Technical report CMU-CS-91-196, Department of Computer Science, Carnegie Mellon University*, 1991.

[Soong and Huang, 1990] Frank K. Soong and Eng-Fong Huang. Fast Tree-Trellis Search for Finding the N-Best Sentence Hypothesis in Continuous Speech Recognition. *Journal of Acoustic Society, AM.*, May 1990.

[Srinivas and Baldwin, 1996] B. Srinivas and Breckenridge Baldwin. Exploiting supertag representation for fast coreference resolution. In *Proceedings of the International Conference on Natural Language Processing and Industrial Applications (NLP+IA '96)*, Moncton, Canada, June 1996.

[Srinivas *et al.*, 1995] B. Srinivas, Christine Doran, and Seth Kulick. Heuristics and parse ranking. In *Proceedings of the* 4^{th} *Annual International Workshop on Parsing Technologies*, Prague, September 1995.

[Steedman, 1987] Mark Steedman. Combinatory Grammars and Parasitic Gaps. *Natural Language and Linguistic Theory*, 5:403–439, 1987.

[Steedman, 1996] Mark Steedman. *Surface Structure and Interpretation*. MIT Press, 1996.

[Steedman, 1997] Mark Steedman, editor. *The Syntactic Interface*. MIT Press, Cambridge Massachusetts and London, England, 1997.

[T.R. Niesler and P.C. Woodland, 1996] T.R. Niesler and P.C. Woodland. A variable-length category-based n-gram language model. In *Proceedings, IEEE ICASSP*, 1996.

[Trueswell and Tanenhaus, 1994] John Trueswell and Mike Tanenhaus. Toward a lexicalist framework for constraint-based syntactic ambiguity resolution. In K. Rayner C. Clifton and L. Frazier, editors, *Perspectives on sentence processing*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1994.

[van Harmelen and Bundy, 1988] Frank van Harmelen and Allan Bundy. Explanation-Based Generalization = Partial Evaluation. *Artificial Intelligence*, 36:401–412, 1988.

[Vijay-Shanker and Joshi, 1991] K. Vijay-Shanker and Aravind K. Joshi. Unification Based Tree Adjoining Grammars. In J. Wedekind, editor, *Unification-based Grammars*. MIT Press, Cambridge, Massachusetts, 1991.

[Vijay-Shanker, 1987] K. Vijay-Shanker. *A Study of Tree Adjoining Grammars*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1987.

[Vijay-Shanker, 1992] K. Vijay-Shanker. Using Descriptions of Trees in a Tree Adjoining Grammar. *Computational Linguistics*, 18, 1992.

[Voutilainen, 1983] Atro Voutilainen. *Two-level Morphology. A General Computation Model for Word-form Production and Generation*. Publications of the Department of General Linguistics, University of Helsinki, 1983.

[Voutilainen, 1994] Atro Voutilainen. *Designing a parsing grammar*. Publications of the Department of General Linguistics, University of Helsinki, 1994.

[Waltz, 1975] D. Waltz. Understanding line drawings of scenes with shadows. In P. Winston, editor, *Psychology of Computer Vision*. MIT Press, 1975.

[Weischedel *et al.*, 1992] Ralph Weischedel, Damaris Ayuso, Sean Boisen, Heidi Fox, and Robert Ingria. A New Approach to Text Understanding. In *Proceedings, Speech and Natural Language Workshop*, San Mateo, CA, 1992.

[Weischedel *et al.*, 1993] Ralph Weischedel, Richard Schwartz, Jeff Palmucci, Marie Meteer, and Lance Ramshaw. Comping with ambiguity and unknown words through probabilistic models. *Computational Linguistics*, 19.2:359–382, June 1993.

[XTAG-Group, 1995] The XTAG-Group. A Lexicalized Tree Adjoining Grammar for English. Technical Report IRCS 95-03, University of Pennsylvania, 1995.

[Zue *et al.*, 1991] Victor Zue, James Glass, David Godine, Hong Leung, Michael Phillips, Joseph Polifroni, and Stephanie Seneff. Integration of Speech and Natural Language Processing in MIT Voyager System. In *Proceedings, IEEE ICASSP*, 1991.