From Linguistic Theory to Syntactic Analysis: Corpus-Oriented Grammar Development and Feature Forest Model
言語理論から統語解析へ：コーパス指向の文法開発と素性森モデル

Yusuke Miyao
宮尾祐介

Department of Computer Science
The University of Tokyo

Abstract

The goal of this thesis is to establish a system for the automatic syntactic analysis of real-world text. Syntactic analysis in this thesis denotes computation of in-depth syntactic structures that are grounded in syntactic theories like Head-Driven Phrase Structure Grammar (HPSG). Since syntactic structures provide essential components for computing meanings of natural language sentences, the establishment of syntactic analyzers is a starting point for intelligent natural language processing. Syntactic analyzers are strongly demanded in natural language processing applications, including question answering, dialog systems, and text mining. To date, however, few syntactic analyzers can process naturally occurring sentences such as newswire texts.

This task involves two significant obstacles. One is the scalability of a grammar to analyze real-world texts. Grammar theories that successfully worked in a toy system could not be applied to the analysis of real-world sentences. Despite intensive research on syntactic analysis, development of wide-coverage grammars is almost impractical. This is due to the inherent difficulty in scaling up a grammar; as a grammar becomes larger, the maintenance of the consistency of the grammar is more difficult. Modern syntactic theories, which are called *lexicalized grammars*, explain diverse syntactic structures with various combinations of *lexical entries* to express word-specific constraints and *linguistic principles* to represent generic syntactic regularities. However, grammar writers cannot simulate in their mind all possible combinations of lexical entries and linguistic principles. Notably, a number of lexical entries are required to treat real-world sentences, and the consistent expansion of lexical entries creates a bottleneck in the scaling up of lexicalized grammars. The problem is further deteriorated by the complicated data structures required in linguistic theories to express in-depth syntactic regularity.

The first proposal of this thesis is a new methodology for the development of lexicalized grammars. The method is *corpus-oriented*, in the sense that the objective of the grammar development is the construction of an annotated corpus, i.e., a treebank, rather than a lexicon. This methodology supports an inexpensive development of lexicalized grammars owing to the systematic control of grammar inconsistencies and the reuse of existing linguistic resources. First, grammar developers define linguistic principles that conform to a target syntactic theory, i.e., HPSG in our case. Next, existing linguistic resources, such as Penn Treebank, are converted into an HPSG treebank. The major work of grammar developers is to maintain the conversion process with the help of consistency checking by principles. That is, because conflicts in a grammar are automatically detected as violations of principle applications to a treebank, grammar writers can easily identify sources of inconsistencies. When we have a sufficient treebank of HPSG, a lexicon is collected from terminal nodes of HPSG syntactic structures in the treebank. Lexicon collection is completely deterministic; that is, treebank construction theoretically subsumes lexicon development.

The other obstacle is the modeling of preference of natural language syntax. Since linguistic research on syntax has focused on structural regularity, modeling of preference was not respected. However, it is indispensable for automatic syntactic analysis because applications usually require disambiguated or ranked parse results. Since probabilistic models attained great success in CFG

parsing, they should be a plausible choice also for lexicalized grammars. Because corpus-oriented development gives us a treebank, it can be used for the estimation of probabilistic models.

Probabilistic modeling of lexicalized grammars is difficult because these grammars exploit complicated data structures, such as typed feature structures. This prevents us from applying common methods of probabilistic modeling in which a complete structure is divided into sub-structures under the assumption of statistical independence among sub-structures; for example, part-of-speech tagging to a sentence is decomposed into tagging to each word, and CFG parsing is split into applications of CFG rules. These methods have relied on the structure of the target problem, namely, lattices or trees, and cannot be applied to graph structures like typed feature structures.

The second proposal of this thesis is *the feature forest model* as a solution to the problem of probabilistic modeling of complex data structures including typed feature structures. The feature forest model provides a method of probabilistic modeling without independence assumption when probabilistic events are represented with feature forests. Feature forests are generic data structures to represent ambiguous trees in a packed forest structure. Feature forest models are maximum entropy models defined over feature forests. A dynamic programming algorithm is proposed for maximum entropy estimation without unpacking feature forests. Thus probabilistic modeling of any data structures is possible when they are represented by feature forests. The thesis also proposes methods of representing HPSG syntactic structures and predicate argument structures with feature forests. Hence, we describe a complete strategy for developing probabilistic models of HPSG parsing.

Finally, experimental results demonstrate that the proposals of this thesis essentially break through the impracticality of automatic syntactic analysis of real-world texts. With corpus-oriented development, an English HPSG grammar was developed using Penn Treebank as a starting treebank. Penn Treebank Section 02-21 (39,832 sentences) was converted into an HPSG treebank, and from it a lexicon of 34,765 words was acquired. Evaluation of grammar coverage against Penn Treebank Section 23 (2,416 sentences) showed that the grammar included correct lexical entries of 99.09% of words, and 84.1% of sentences were fully covered, i.e., the grammar included all lexical entries in a sentence. The results attested to a high coverage that has not yet been attained by existing grammars. The manual investigation of the grammar also proved that an HPSG grammar was developed with high accuracy. The HPSG treebank was also exploited as training data of probabilistic models for disambiguation. Feature forest models were applied for the tractable estimation of probabilistic models of HPSG parsing. Evaluation of parsing accuracy showed that the syntactic analyzer could produce at least one syntactic structure for 99.7% sentences, and the accuracy of predicate-argument relations was 87.69%/87.16% (precision/recall). The results reveal evidence of the practicality of the HPSG parser in the task of parsing real sentences.

To our knowledge, this work provides the first results of the extensive evaluation of HPSG parsing of naturally occurring sentences. Empirical experiment results are beneficial for further improvements of HPSG parsing and other work on syntactic analysis. Practically, the success of the development of a syntactic analyzer opens up a new possibility of natural language processing. Most importantly, this thesis constitutes fundamental methodologies for the syntactic analysis of real-world text.

Acknowledgments

I would like to convey my profound gratitude to, and respect for, Professor Jun'ichi Tsujii. He invited me into the exciting research field of computational linguistics, and also gave me much valuable advice and ongoing encouragement.

I also deeply appreciate the support of Dr. Takashi Ninomiya. He had a great effect on me in my research and daily life. Most of the work in this thesis was derived from extensive discussions with him. His unique philosophical way of thinking sometimes annoyed me, but such experiences with him have been and will be irreplaceable.

I would like to acknowledge Professor Kentaro Torisawa. He taught me everything about research, since I came to the Tsujii Laboratory. My style of research and writing was substantially influenced by him. I would also express my gratitude to Dr. Yuka Tateisi. She was the first researcher with whom I had a joint project on grammar development. Without my work with her, the concepts in this thesis could not have appeared. I would also like to thank Dr. Takaki Makino. My software owed much to his software, LiLFeS, and could never have been developed without his help. I also learned a lot about programming skills by watching his programs.

I am also grateful to Dr. Yoshimasa Tsuruoka for all of his help. He continuously gave me support on both technical and theoretical problems. Without his help, this work could never have been completed. I am really obliged to Dr. Jin-Dong Kim and Dr. Tomoko Ohta for their generous support and encouragement. I am also appreciative of Dr. Jun'ichi Kazama, Dr. Naoki Yoshinaga, and Ms. Akane Yakushiji for our precious and frank discussions about each other's research. The discussions greatly affected my research direction and interest.

I am really grateful to Mr. Yosuke Sakao, Mr. Tadayoshi Hara, Mr. Kazuhiro Yoshida, Ms. Hiroko Nakanishi, Mr. Manabu Sato, and Mr. Daiki Kojima. They were co-developers and extensive users of Enju. Despite many bugs in the system, they patiently used the software and reported various problems. I would also like to convey my deepest appreciation to all members of Tsujii Laboratory. They have continued to be the best colleagues and rivals, and enabled me to have a pleasant research life.

I would also like to thank Mr. Hiroshi Kanayama. He was a key person for me in coming into computational linguistics research. I am also grateful to fellows at the graduate school, Dr. Minoru Yoshida, Mr. Katsutoshi Ibushi, Mr. Hisao Imai. I have also very much appreciated the help of secretaries, Ms. Minako Ito and Ms. Mika Tarukawa. They not only lent their hands for my various duties but also gave me healing time.

Lastly, I am deeply grateful to my mother, father, Sachie, Kohei, and May for seeing and supporting me always. I also thank my grandparents and relatives for treating me with gentle patience.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Syntactic analysis, or *parsing*, has been a central research theme in the history of computational linguistics. The objective of syntactic analysis is to compute *syntactic structures*, which express correspondence among natural language sentences and their meanings. Since sentence-to-meaning correspondence is complicated and nontrivial, such research is challenging and necessary for understanding natural language. Syntactic analysis has also been demanded in natural language processing applications, such as question answering, dialog systems, and machine translation. These applications require computation of meanings of natural language sentences. Since syntactic structures provide essential components for computing meanings of sentences, the establishment of automatic syntactic analyzers is a starting point for intelligent natural language processing.

Unfortunately, however, syntactic analysis has not been able to fulfill the need of the processing of unrestricted natural text. To date, syntactic analyzers could process only short sentences in a limited domain, or were toy systems that did not aim at being used in practical applications. This is a problem of the *scalability* of syntactic analyzers; theories and methodologies that worked successfully in toy syntactic analyzers could not be applied to the processing of real-world sentences. Scalability for analyzing a variety of sentences is definitely indispensable in natural language processing because the nature of natural language is to transfer a variety of information.

This thesis is dedicated to the establishment of a syntactic analyzer that can compute syntactic structures of real-world text. The subject requires solutions to breaking through the difficulty of scaling up syntactic analysis. To begin with, we explore sources of the failure of syntactic analysis, which will lead us to the proposals of this thesis.

1.1 Discrepancy between Syntactic Theory and Syntactic Analysis

The difficulty of syntactic analysis of real-world sentences has been due to the difference in motivations of syntactic theory and syntactic analysis. Traditionally, the development of syntactic analyzers has been synonymous with putting syntactic theories into computer programs. However, when we target processing of real-world sentences, syntactic theories are insufficient.

While the syntax of language was a subject of much longer research in literature and linguistics, computer science introduced a new insight into syntax: syntactic structure is defined as a result of computation (Chomsky, 1957). This concept assumes that we humans have a meaning representation in our mind, and it emerges as a sentence through a certain computational process. The process is complicated and correspondence among sentences and their meanings is not necessarily one-to-one mapping. For example, while preserving the meaning, a subject-verb-object relation may be expressed in a declarative form, in a passive form (passivization), or in a relative clause (relativization).

To perceive sentence-to-meaning correspondence, we theorize a representation of the computational process; that is, syntactic structure.

From this concept, research on syntactic theories has been equivalent to describing *a grammar*, which is a set of rules that deduce syntactic structures. While infinite syntactic structures may exist, syntactic theories aim at explaining the regularity of syntactic structures by defining a finite grammar. To some extent, modern syntactic theories succeeded in defining a system of rules that explain complicated correspondence among sentences and meaning representations. Examples include Lexicalized Tree Adjoining Grammar (LTAG) (Schabes, 1992; Schabes et al., 1988), Lexical Functional Grammar (LFG) (Bresnan, 1982), Combinatory Categorial Grammar (CCG) (Steedman, 2000), and Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994; Sag and Wasow, 1999).

The objective of syntactic analysis involves two crucial issues, which are indispensable for the scalability of syntactic analyzers but have been neglected by syntactic theories. One is that syntactic analyzers must process any input sentence. This means that we need *a comprehensive grammar*, i.e., exhaustive rules so that it accepts any words and syntax. Syntactic theories discussed representative examples of words/syntax, and were less interested in the exhaustiveness of rules. The other is that syntactic analyzers must provide plausibility of syntactic structures; that is, *disambiguation* is necessary. Although syntactic theories intended to enumerate all grammatical syntactic structures, applications usually require only one or a few syntactic structures that are probable for a given sentence in a given context.

In late nineties, corpus-driven methods opened up the possibility of the processing of real-world sentences. The fundamental idea was to extract grammars from *treebanks* automatically (Charniak, 1997, 2000; Collins, 1996, 1997). Treebanks are linguistic resources in which syntactic structures are manually annotated to example sentences. That is, they are collections of real instances of syntactic structures. While corpus-driven methods succeeded in the analysis of real-world sentences, they were insufficient for our purpose because they did not provide full syntactic structures that were assumed in syntactic theories. Since development of treebanks is costly, existing treebanks, such as Penn Treebank (Marcus et al., 1994), provided only *shallow* syntactic structures like phrase structures, i.e., trees of phrase symbols, or CFG-style parse trees. Hence, grammars extracted from them lack necessary information for computing sentence-to-meaning correspondence, such as passivization and relativization.

Although corpus-driven methods did not fulfill the need of syntactic analysis, their success indicates clues to breaking through the problems with comprehensive grammars and disambiguation. In the following sections, we explicate the problems by contrasting the philosophy starting from syntactic theories against the methodology of corpus-driven approaches.

1.2 Theory vs. Data

The heart of the theory-oriented methodology was manual management of a grammar. That is, implementation of syntactic analyzers was regarded as writing a grammar. This was because the development of syntactic analyzers followed a philosophy of syntactic theories; that is, to define a grammar to deduce all syntactic structures of grammatical sentences. Grammarians first conceive grammar rules so as to deduce grammatical syntactic structures by consulting their linguistic intuition. Next, they implement grammar rules as computer programs. Subsequently, they debug the programs by parsing example sentences and observing the results of the parsing. Traditional syntactic analyzers were developed with this strategy: LinGO English Resource Grammar (Copestake and Flickinger, 2000; Flickinger, 2002), JACY (Siegel, 2000; Siegel and Bender, 2002), XTAG grammars (XTAG Research Group, 2001), and ParGram grammars (Butt et al., 2002). For ex-

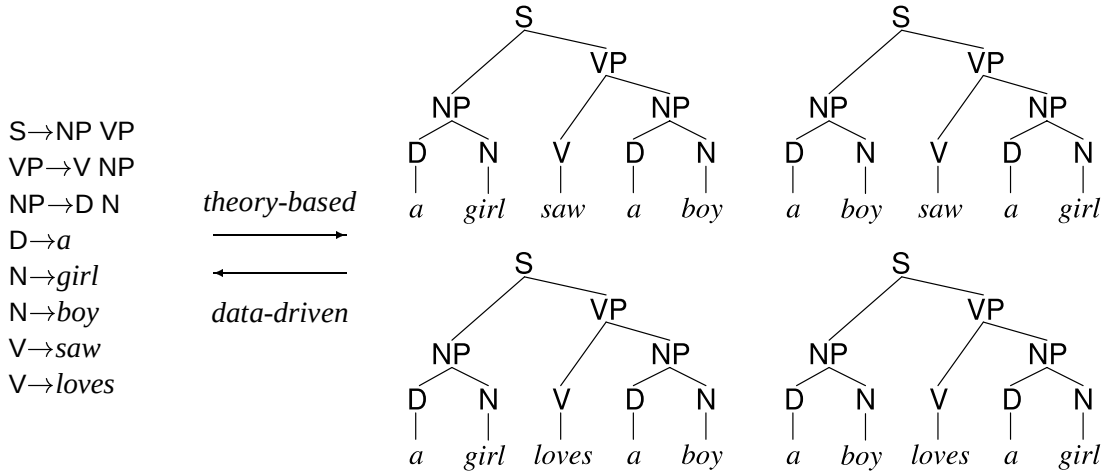


Figure 1.1: A context-free grammar and a treebank

ample, in Figure 1.1, the goal of the development is to write CFG rules (the left-hand side) in order to deduce grammatical phrase trees (the right-hand side).

An approach taken by corpus-driven methods was essentially different; human effort was directed to data. By consulting linguistic intuition, developers first write a treebank, i.e., example sentences annotated with syntactic structures. Next, grammar rules are automatically extracted from a treebank. CFG parsers widely used now were developed with this strategy (Charniak, 1997, 2000; Collins, 1996, 1997). In Figure 1.1, we first develop a treebank (the right-hand side), and extract a grammar (the left-hand side) by fragmenting each branching of phrase trees in the treebank.

An essential difference of the two methodologies is that the target of development was a grammar, or a treebank. In other words, the difference is the destination of human intervention. In this respect, an advantage of the latter methodology is that human intervention is targeted toward real instances. In the former methodology, however, human effort is directed to abstract rules, and instances of syntactic structures only exist in the mind of grammar developers.

When scaling up syntactic analyzers to assess real-world sentences, the methodology of theory-oriented methods involves an inherent difficulty in the management of human errors, i.e., consistency of grammar rules. When grammar developers add or modify a rule to treat some words/constructions, the modification often incorporates new conflicts of grammar rules for other words/constructions. This is because human developers cannot simulate in their mind all interactions of rules.

For example, let us suppose that we are adding new rules to a grammar shown in Figure 1.1 to treat new constructions. First, we add rules to handle pronouns, "*it*", "*this*", and "*that*."

- NP→*it*
- NP→*this*
- NP→*that*

This modification allows sentences like "*a boy saw it*." Next, we add rules to treat prepositional phrases.

- PP→*in* NP
- PP→*of* NP

Prepositional phrases do not modify pronouns: for example, "*it in a school*" is ungrammatical. Hence, we add rules of noun-modifying prepositional phrases.

- N→N PP

By adding more rules for nouns, say "N→*school*," a new grammar can generate sentences like "*a boy loves a girl in a school*." However, the grammar cannot generate sentences like "*a boy saw that in a school*." This is because we added a rule for prepositional phrases to modify N, not NP. Since "*that*" is derived from NP, prepositional phrases cannot modify it. With this observation, we decide to revise a rule for "*that*," where "*that*" is generated from N.

- N→*that*

However, this revision introduces a new conflict, and "*that*" cannot be used in most grammatical situations. For example, the grammar cannot generate "*a boy saw that*," because "*that*" has no determiners while "*that*" as N must have a determiner. Even in this very simple example of extending a CFG, the difficulty of consistency maintenance is obvious.

The source of the problem is that humans are poor at comprehending abstract concepts strictly. In the above example, the concept of *pronoun* when we say "*that*" is a *pronoun* is slightly different from that when we say prepositional phrases do not modify *pronouns*. However, grammar developers do not realize a subtle difference of concepts when they tweak a grammar. On the contrary, computers are very strict with the consistency of rules. This problem is terribly serious in modern syntactic theories in which syntactic structures express detailed linguistic constraints using complex data structures. The maintenance of the consistency of a grammar becomes more difficult as the grammar grows larger. The possibility of conflicts rapidly increases because of the explosion of rule combinations, and the revision of conflicts frequently brings in other conflicts.

If grammar developers are given real instances, they can realize their misconception. For example, if a treebank includes a phrase like "*that of a house*," it is easy to see a wrong conception of the rule "prepositional phrases do not modify pronouns." Grammar developers therefore require contact with instances of syntactic structures.

Data-driven methods overcame the problem of consistency maintenance. They started from real examples of syntactic structures, and a grammar was extracted from the examples. Treebank construction is more comprehensible for human developers than rule development. This is because the target of the development is instances of syntactic structures of concrete sentences rather than abstract rules. Developers are not hampered by having to simulate interactions of rules. It should also be noted that this approach assures that an extracted grammar deduces at least syntactic structures given in a treebank. That is, undergeneration is avoided with regard to treebank sentences, although overgeneration is not controlled.

This discussion reveals that human intervention in real instances of syntactic structures is necessary and effective, when we write comprehensive grammars as computer programs. A solution will be presented in Chapter 3.

1.3 Structure vs. Statistics

Disambiguation is necessary for syntactic analysis. While grammars are designed to deduce all grammatical syntactic structures, applications often require only the most plausible syntactic structures. However, research on syntactic theories was not interested in the problem of disambiguation.

Traditionally, syntactic theories focused on structural constraints of syntactic regularity. They discussed the acceptability of sentences and constraints necessary to eliminate ungrammatical sentences. Linguists first show rules to deduce a syntactic structure of a sentence, and then demonstrate

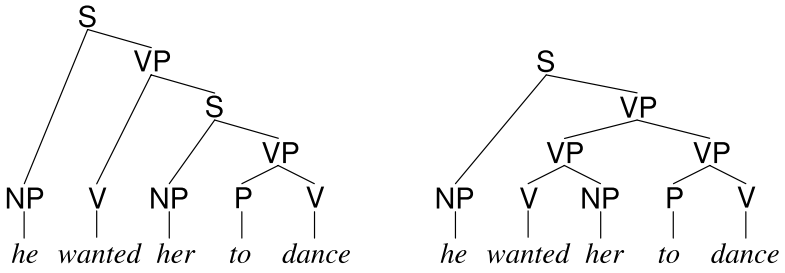


Figure 1.2: Ambiguity of syntactic structures

that the rules reject syntactic structures of similar but ungrammatical sentences. To be more accurate, they presuppose that each syntactic structure represents a distinct meaning. That is, if a sentence has multiple readings, a grammar must deduce syntactic structures, each of which corresponds to an individual meaning of the sentence. To discriminate syntactic structures of distinct meanings, they employed structural constraints expressed with symbolic representations such as feature structures (Carpenter, 1992; Rounds and Kasper, 1986).

Meanwhile, the success of corpus-driven methods has owed to statistical and machine learning methods. One reason for the rapid growth of statistical methods was that computer power increased sufficiently to process huge amounts of data, and large linguistic resources such as Penn Treebank (Marcus et al., 1994) and WordNet (Fellbaum, 1998; Miller, 1990) became available. As described below, however, a more significant reason was that disambiguation required statistical modeling of syntactic regularity.

Disambiguation of syntactic structures requires statistical preference. For example, Figure 1.2 shows two phrase structures assigned to sentence "*He wanted her to dance*." The left tree corresponds to an object-control construction, while the right represents that "*to dance*" is a purpose infinitive. The left tree seems correct to us, but it is very difficult to eliminate the right tree only by structural constraints. Intuitively, the acceptability seems to be determined by *preference*. For example, for the similar sentence, "*He wanted shoes to dance*," a phrase structure similar to the right seems plausible since "*shoes*" is unlikely to be a subject of "*dance*." This means that modeling of "preference" of syntactic regularity is necessary for disambiguation.

Corpus-driven methods adopted statistical methods for the problem of disambiguation. Since they extracted rules from treebanks as described in Section 1.2, they additionally learned statistical preferences of rules from treebanks. Disambiguation models of state-of-the-art CFG parsers were probabilistic models such as probabilistic context-free grammars (PCFGs) (Charniak, 1997, 2000; Collins, 1999, 1996, 1997) and maximum entropy models (Ratnaparkhi, 1997; Uchimoto et al., 2000, 1999). Other studies employed statistical machine learning methods such as decision trees (Magerman, 1995) and support vector machines (Kudo and Matsumoto, 2000, 2002).

Unfortunately, we cannot simply follow the above methods. A problem is that statistical methods require training data; in our case, treebanks. Development of sufficient treebanks is almost infeasible since syntactic structures must be written manually for thousands of sentences. Even shallow syntactic structures, such as Penn Treebank, required considerable human effort.

Another problem is that their methods were strongly committed to the structure of outputs, i.e., tree structures. They presupposed that outputs of syntactic analyzers were represented with sequences or tree structures. For example, when we compute probabilities of the phrase structures in Figure 1.2, a probability of a tree is first decomposed into probabilities of rewriting rules. Figure 1.3 shows probabilities of rewriting rules estimated from Figure 1.2. Rule probabilities are defined as relative

$p(\mathsf{S} \rightarrow \mathsf{NP\ VP}|\mathsf{S}) = 1.0$ $p(\mathsf{NP} \rightarrow \textit{he}|\mathsf{NP}) = 0.5$
$p(\mathsf{VP} \rightarrow \mathsf{V\ S}|\mathsf{VP}) = 0.2$ $p(\mathsf{NP} \rightarrow \textit{her}|\mathsf{NP}) = 0.5$
$p(\mathsf{VP} \rightarrow \mathsf{P\ V}|\mathsf{VP}) = 0.4$ $p(\mathsf{V} \rightarrow \textit{wanted}|\mathsf{V}) = 0.5$
$p(\mathsf{VP} \rightarrow \mathsf{VP\ VP}|\mathsf{VP}) = 0.2$ $p(\mathsf{V} \rightarrow \textit{dance}|\mathsf{V}) = 0.5$
$p(\mathsf{VP} \rightarrow \mathsf{V\ NP}|\mathsf{VP}) = 0.2$ $p(\mathsf{P} \rightarrow \textit{to}|\mathsf{P}) = 1.0$

Figure 1.3: Probabilities of rewriting rules

$\mathsf{S} \rightarrow \mathsf{NP}_{\boxed{1}}\ \mathsf{VP}_{\boxed{1}}$ $\mathsf{VP}_{\textit{no\_3sg}} \rightarrow \textit{dance}$
$\mathsf{NP}_{\textit{no\_3sg}} \rightarrow \textit{I}$ $\mathsf{VP}_{\textit{3sg}} \rightarrow \textit{dances}$
$\mathsf{NP}_{\textit{3sg}} \rightarrow \textit{she}$ $\mathsf{VP}_{\textit{no\_3sgor3sg}} \rightarrow \textit{danced}$

Figure 1.4: Rewriting rules with structure sharings

frequencies of rules applied in the treebank. For example, "$\mathsf{VP} \rightarrow \mathsf{V\ S}$" is used once in the left tree, and the frequency of rules of rewriting VP is five. Hence, the probability is estimated by $1/5 = 0.2$. Assuming that applications of rewriting rules are independent of the other applications, the probability of the whole structure is defined as a product of all probabilities of rewriting rules applied to construct the tree. For example, the probability of the left tree is computed as follows.

$$
\begin{aligned}
& p(\mathsf{S} \rightarrow \mathsf{NP\ VP}|\mathsf{S})p(\mathsf{VP} \rightarrow \mathsf{V\ S}|\mathsf{VP})p(\mathsf{S} \rightarrow \mathsf{NP\ VP}|\mathsf{S})p(\mathsf{VP} \rightarrow \mathsf{P\ V}|\mathsf{VP}) \\
& p(\mathsf{NP} \rightarrow \textit{he}|\mathsf{NP})p(\mathsf{V} \rightarrow \textit{wanted}|\mathsf{V})p(\mathsf{NP} \rightarrow \textit{her}|\mathsf{NP})p(\mathsf{P} \rightarrow \textit{to}|\mathsf{P})p(\mathsf{V} \rightarrow \textit{dance}|\mathsf{V}) \\
= \ & 1.0 \times 0.2 \times 1.0 \times 0.4 \times 0.5 \times 0.5 \times 0.5 \times 1.0 \times 0.5 \\
= \ & 0.005
\end{aligned}
$$

The decomposition of probabilities is indispensable for the tractable estimation of model parameters, i.e., rule probabilities, and decoding while avoiding an exponential explosion of ambiguities of syntactic structures. In general, ambiguity of syntactic structures increases exponentially. For example, when we apply rewriting rule "$\mathsf{S} \rightarrow \mathsf{NP\ VP}$", and the left NP and the right VP respectively have n and m ambiguous subtrees, the result of the rule application generate $n \times m$ trees. This is an obstacle to parameter estimation and the searching of the most probable syntactic structure, because enumeration of all structures is intractable. Dynamic programming algorithms, including the forward/backward and the Baum-Welch algorithms for hidden Markov models (Baum and Eagon, 1967), the inside/outside algorithm for probabilistic context-free grammars (Baker, 1979; Lari and Young, 1990), and the Viterbi decoding (Viterbi, 1967), provide polynomial-time solutions to the problems, with the assumption that the probability of a complete sequence/tree is defined as a product of rule probabilities.

Complex data structures, such as feature structures, prevent us from decomposing the probability of a whole structure into independent rule probabilities. Abney (1997) demonstrated that decomposition into rule probabilities causes distortion of a probabilistic distribution. While this problem will be discussed in detail in Chapter 4, the source of the problem is *structure sharing*, or *reentrant structures*, which are exploited for representing linguistic constraints in modern syntactic theories. For example, Figure 1.4 shows rewriting rules with constraints on agreement. Suffixes denote types of agreement, and $\boxed{1}$ is a variable that constrains that suffixes with the same variable name must have the same value. Constraints introduced by variables are called structure-sharing or reentrant structure.

The constraint by 1 inhibits ungrammatical sentences such as "*she dance*", because "*she*" has *3sg* but "*dance*" has *no_3sg*, and the constraint by 1 inhibits the NP and the VP have different suffixes. That is, rewriting rules are no more "context-free." Because the decomposition of probabilities relied on the property of "context-freeness," the same method cannot be applied to syntactic structures with structure sharings.

To summarize, syntactic analyzers require statistical methods for disambiguation, but traditional statistical models cannot be directly applied to grammars described with complex data structures such as feature structures.

1.4 Contributions of This Thesis

The focus of this thesis is on the development of a syntactic analyzer that computes syntactic structures of real-world sentences. To be more specific, this thesis is organized with the aim of the development of a practical English parser based on HPSG. The proposals will actually be implemented for the development of an HPSG parser for English, and the parser will be evaluated extensively through experiments on the parsing of real-world sentences.

The above discussion reveals that the following problems must be solved.

1. Developing a comprehensive grammar while maintaining its consistency
2. Developing a treebank for the training of disambiguation models
3. Probabilistic modeling of structured data

The following chapters of this thesis provide solutions to the problems listed above under two assumptions. One is that a grammar is based on *the lexicalized grammar framework*. The framework of lexicalized grammars is a de facto standard in modern syntactic theories, and is well studied from linguistic and computational points of view. Lexicalized grammars include LTAG (Schabes, 1992; Schabes et al., 1988), LFG (Bresnan, 1982), CCG (Steedman, 2000), and HPSG (Pollard and Sag, 1994; Sag and Wasow, 1999). Chapter 2 introduces a concept of lexicalized grammars and the details of Head-Driven Phrase Structure Grammar (HPSG), which is one of the lexicalized grammars.

The other assumption is that disambiguation models for lexicalized grammars are *maximum entropy models* (Berger et al., 1996), which are widely accepted in the probabilistic modeling of various tasks of natural language processing including parsing. Most studies on probabilistic models for parsing with lexicalized grammars also adopted maximum entropy models (Baldridge and Osborne, 2003; Clark and Curran, 2003, 2004b; Geman and Johnson, 2002; Johnson et al., 1999; Kaplan et al., 2004; Malouf and van Noord, 2004; Riezler et al., 2002, 2000; Toutanova and Manning, 2002). Chapter 2 describes maximum entropy models in detail.

To evaluate the effectiveness of our proposals, an HPSG parser for English was implemented. The performance of the parser will be evaluated extensively in the task of parsing Penn Treebank (Marcus et al., 1994), which is a treebank of newswire texts from Wall Street Journal and is often used for the evaluation of CFG parsers. However, the methods proposed in this thesis do not assume any characteristics peculiar to the theory of HPSG. This indicates that the proposals are generally applicable to the development of syntactic analyzers based on lexicalized grammars.

Contributions of the dissertation will be described in detail in four chapters. A brief introduction to each chapter is described below.

Corpus-Oriented Development of Lexicalized Grammars Chapter 3 proposes a novel methodology of grammar engineering for the solution of Problems 1 and 2, i.e., the development of a comprehensive grammar and a treebank. As discussed in Section 1.2, it is hard to develop exhaustive rules

because of the difficulty of consistency maintenance, as well as the high cost of the construction of large treebanks.

The idea is to separate grammar rules into *linguistic principles* and *lexical entries*, and to develop linguistic principles and a treebank. Linguistic principles and lexical entries are principal components of lexicalized grammars including HPSG. Linguistic principles represent general syntactic constraints of language, while lexical entries represent word-specific syntactic characteristics. In the traditional methodology, grammar developers manually wrote both linguistic principles and lexical entries. The difficulty of developing comprehensive lexicalized grammars was mainly due to lexical entries, because detailed lexical characteristics must be enumerated for thousands of words.

This chapter demonstrates a method for the development of an English grammar of HPSG. The development process of our method is *corpus-oriented*, in the sense that the target of the development is an HPSG treebank instead of lexical entries. Given a treebank and linguistic principles, lexical entries are automatically collected from terminal nodes of syntactic structures in the treebank.

In addition to the merits of developing a treebank discussed in Section1.2, this methodology has several advantages. One is that human intervention is directed to both linguistic principles and treebanks. That is, grammar developers can control the theoretical formulation of the treebank through the design of linguistic principles. In other words, since a treebank is developed so as to satisfy constraints offered by linguistic principles, grammarians can reflect their linguistic intuition into a treebank through linguistic principles. As a result, lexical entries collected from the treebank also conform to the theoretical formulation regulated by linguistic principles. Another advantage is that linguistic principles greatly help the maintenance of consistency of detailed syntactic constraints. This is because conflicts of syntactic constraints in a treebank are automatically detected as violations of applications of linguistic principles. Furthermore, existing treebank resources are reused. For example, when we develop an HPSG treebank, we start from Penn Treebank, which is a CFG-style treebank in English, and convert it into HPSG-style syntactic structures. This approach greatly reduces the cost of treebank development.

With this approach, an HPSG grammar for English was implemented using Penn Treebank as a source treebank. We implemented ID schemas, principles, and lexical rules of HPSG following the definition of Pollard and Sag (1994), and converted Penn Treebank into an HPSG treebank. In Chapter 3, details of the treebank development are described. The evaluation of the HPSG parser will be presented in Chapter 6.

Feature Forest Model Chapter 4 provides a generic solution to Problem 3, i.e., probabilistic modeling of structured data. We first define *feature forest*, which is a packed forest structure similar to a packed chart of CFG. We prove that if a structured data is represented as a feature forest, estimation of parameters of maximum entropy models is tractable.

The algorithm allows for the probabilistic modeling of complete structures, such as transition sequences in Markov models and parse trees, without dividing them into independent sub-events. In general, complete structures have an exponential number of ambiguities. This causes an exponential explosion of the cost of estimating parameters of maximum entropy models. The proposed algorithm avoids exponential explosion by representing events with feature forests, which are packed representations of trees. If complete structures are represented by feature forests of a tractable size, parameters are efficiently estimated by dynamic programming similar to the algorithm of computing inside/outside probabilities of PCFGs without unpacking the feature forests.

The algorithm provides a flexible modeling scheme for various NLP tasks because we may incorporate various overlapping features of complete structures, not only of sub-events, into the model. Moreover, it can be applied to probabilistic models of events that are difficult to divide into independent sub-events, such as probabilistic models of feature structures.

Probabilistic Modeling of Lexicalized Grammars Chapter 5 discusses the application of feature forest models to HPSG parsing. That is, the solution in Chapter 4 is discussed in a practical application context.

Since syntactic structures express various linguistic relations, such as non-local dependencies of words, their probabilistic models are expected to attain high accuracy of parsing. Existing models of probabilistic parsing (Charniak, 1997; Chiang, 2000; Clark et al., 2002; Collins, 1999, 1996, 1997; Hockenmaier and Steedman, 2002b; Magerman, 1995) decompose the probability of a parse result into probabilities of atomic events (e.g. rule probabilities) by assuming the independence of the probabilities. Such models, however, cannot be applied to parsing with lexicalized grammars, because it violates the independence assumption and the resulting probabilistic model will be inconsistent (Abney, 1997).

This chapter provides a theoretical framework for the probabilistic modeling of parsing with lexicalized grammars. We show that syntactic structures of HPSG and predicate argument structures are represented with feature forests. Their probabilistic models are therefore estimated in a tractable cost by the algorithm of feature forest models.

Experiments on HPSG parsing The methods proposed in this dissertation were implemented for the development of an HPSG parser for English. Chapter 6 reports empirical evaluation of the HPSG parser in the task of parsing of Penn Treebank, which is a collection of Wall Street Journal texts and was a standard test set for the evaluation of CFG parsers.

To our knowledge, this work provides the first results of extensive experiments of parsing of Penn Treebank with a probabilistic HPSG. The results from the Wall Street Journal are significant because the complexity of sentences differs from that of short sentences. For instance, short sentences rarely include complex constructions, such as long-distance dependencies, coordinations, and insertions. Experiments on the parsing of real-world sentences can properly evaluate the effectiveness and possibility of parsing models for HPSG.

Chapter 2

Background

This chapter introduces two concepts, *lexicalized grammar formalism* and *maximum entropy model*. They are essential conceptions not only for this thesis, but also for research on syntactic theories and statistical natural language processing.

Lexicalized grammar formalism is a modern framework for explaining syntactic structures of natural language sentences. The main concept is that structures and rules to derive syntactic structures are mainly introduced by *lexical entries*, which are structures assigned to words. Intuitively, terminal nodes of parse trees incorporate most structures, and parse trees are constructed by checking the consistency of the lexical structures. Modern syntactic theories fall into lexicalized grammars; examples include Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1987, 1994; Sag and Wasow, 1999), Lexicalized Tree Adjoining Grammar (LTAG) (Schabes, 1992; Schabes et al., 1988), Lexical Functional Grammar (LFG) (Bresnan, 1982), and Combinatory Categorial Grammar (CCG) (Steedman, 2000). In the field of linguistics, lexicalized grammars have provided a mathematical framework for describing linguistic structures, and were extensively studied in order to explicate the structure of syntax in a mathematically sound manner. They also attracted researchers working in the field of computer science, because they were defined with a strict mathematical formulation and could be implemented as computer programs. In fact, several systems are now being implemented with the aim of being used in real applications.

In this chapter, we describe details of the theory of Head-driven Phrase Structure Grammar (HPSG), which is a syntactic theory extensively studied from both linguistic and computational points of view. The main focus of this thesis is on HPSG, and the description of proposals will be presented with examples of HPSG analyses. However, the methods proposed in this thesis do not assume any structures peculiar to the theory of HPSG, and will be extended to other syntactic theories based on the framework of lexicalized grammars.

The maximum entropy model is a probabilistic model widely used in natural language processing. In maximum entropy models, probabilistic events are represented with *features*, and probabilities are defined as a function of features and their weights. A preferable characteristic of maximum entropy models is that features are not required to be statistically independent of each other. While traditional probabilistic models often decompose a probability into sub-probabilities under the assumption of statistical independence, such an assumption is not required in maximum entropy modeling. Hence, model developers can incorporate various kinds of information into a model without considering the independence among features. Even with statistically dependent features, parameters of maximum entropy models are determined so as to maximize the likelihood of training data. This characteristic is indispensable for probabilistic modeling of lexicalized grammars as discussed in detail in Chapter4.

Section 2.1 overviews the idea of lexicalized grammars, and surveys existing systems based on lexicalized grammars. Section 2.2 explains details of HPSG. Section 2.3 describes algorithms of

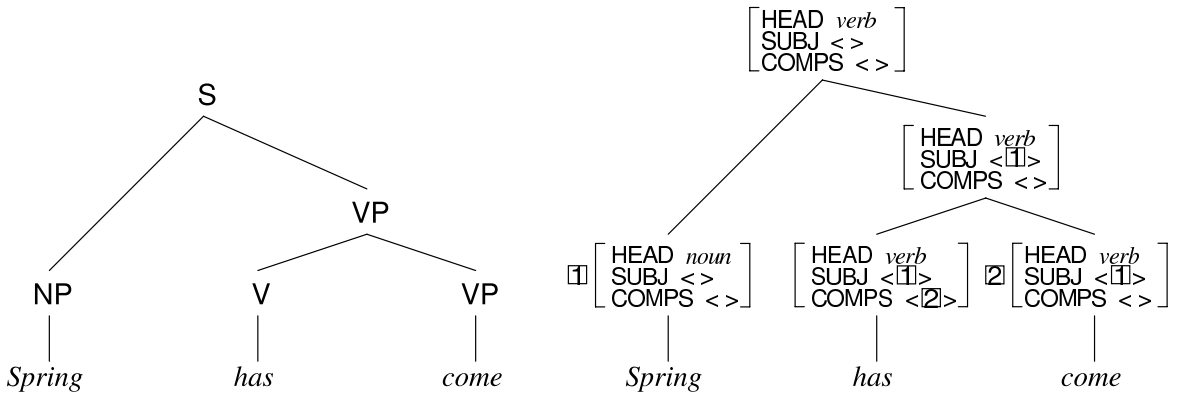


Figure 2.1: Parse trees in CFG (left) and HPSG (right)

parsing with lexicalized grammars. Section 2.4 introduces the idea of maximum entropy model and its formulation.

2.1 Lexicalized Grammars

Lexicalized grammars are frameworks for the description of syntax in modern linguistics. They are defined with *lexical entries* and *grammar rules*, which deduce syntactic structures, i.e., parse trees. The concept of lexicalized grammars is summarized in the following two points.

- A large number of *lexical entries* represent detailed word-specific linguistic constraints.
- A small number of *grammar rules* describe general linguistic constraints.

Lexical entries are terminal nodes, which correspond to words, in parse trees. Hence, the above concept indicates that the structure of parse trees is explained mostly by word-specific constraints.

This concept, which is called a "lexicalist" framework, is very different from context-free grammars (CFGs), which owe much to grammar rules to explain syntactic structures. In CFGs, terminal nodes generally represent poor information. For example, in Penn Treebank (Marcus et al., 1994), which is a corpus annotated with CFG-style parse trees, terminal nodes are annotated with part-of-speech tags such as VBD (past-tense verbs) and NNS (plural nouns).

Figure 2.1 compares CFG and HPSG-style parse trees for the same sentence. In the CFG-style tree, nodes represent only syntactic categories such as NP (noun phrases), VP (verb phrases), and S (sentences). The structure of the parse tree is mainly determined by grammar rules such as VP $\rightarrow$ V VP and S $\rightarrow$ NP VP. In the HPSG-style tree, however, lexical entries not only have syntactic categories (represented by HEAD) but also other syntactic constraints (see Section2.2.2 for details). For example, SUBJ and COMPS describe *a subcategorization frame*, which represents constraints on arguments that are taken by the verb. SUBJ represents constraints of what the verb takes as a subject, and COMPS has constraints of an object. Boxed numbers represent *structure-sharings* (will be explained in detail in the next section). That is, structures denoted with the same numbers must be identical. In the example, the verb "*come*" has an element in SUBJ, which has the same boxed number as the lexical entry for "*spring*." This means that "*come*" must have a noun subject, and this constraint is satisfied by the lexical entry for "*spring*." A parse tree is built by verifying the consistency of constraints specified in lexical entries.

$$\begin{bmatrix} \text{HEAD } \textit{verb} \\ \text{SUBJ } \langle [\text{HEAD } \textit{noun}] \rangle \\ \text{COMPS } \langle [\text{HEAD } \textit{noun}] \rangle \end{bmatrix} \quad \begin{bmatrix} \text{HEAD } \textit{verb} \\ \text{SUBJ } \langle [\text{HEAD } \textit{noun}] \rangle \\ \text{COMPS } \langle \rangle \end{bmatrix}$$

Figure 2.2: Lexical entries for "*loves*" (left) and "*dances*" (right)

The background of this concept is that the grammaticality of sentences depends largely on characteristics specific to words. Let us consider the grammaticality of the following sentences[1].

1. He loves a girl.
2. *He loves.
3. *He dances a girl.
4. He dances.

The grammaticality of the above sentences is completely determined by the characteristics of main verbs ("*loves*" or "*dances*"). To be more specific, the grammaticality depends on subcategorization frames of the main verbs, i.e., transitive or intransitive. This indicates that grammaticality, or the syntactic structures of sentences, is determined by the characteristics of words in sentences. Hence, linguistic constraints, including subcategorization frames and syntactic categories, must be specified in lexical entries.

Figure 2.2 shows lexical entries for the main verbs in the above sentences. The difference is in the value of COMPS. The left lexical entry, which is for "*loves*", has an element in COMPS. This means that the verb takes a noun object. However, the right lexical entry, which is for "*dances*", has no element in COMPS. As shown in this example, lexicalized grammars express differences of subcategorization frames in lexical entries. Various constraints not limited to subcategorization frames may be specified in lexical entries. For example, in HPSG, agreement features, verbal forms, and semantics are described in lexical entries.

Suppose that set $\mathcal{W}$ of words and set Γ of *categories* (e.g., nonterminal symbols in CFG and feature structures in HPSG) are given. A grammar is then defined formally as follows.

Definition 2.1 (Grammar) *A grammar is a tuple, $G = \langle L, R \rangle$, where*

- *$L = \{\langle w, \gamma \rangle | w \in \mathcal{W}, \gamma \in \Gamma\}$ is a lexicon, i.e., a set of lexical entries, and*
- *R is a set of grammar rules, i.e., $r \in R$ is a partial function: $\Gamma \times \Gamma \to \Gamma$.*[2]

Although this definition does not include the formulation of the concept of the lexicalist framework, it is sufficient for discussing computational aspects of lexicalized grammars. Informally, in the lexicalist framework, the size of R is relatively small, while Γ is large and complex in general.

Modern syntactic theories, including LTAG, LFG, CCG, and HPSG, conform to the lexicalist framework. They differ in the degree of the separation of lexical entries and grammar rules, and in the treatment of certain linguistic phenomena. However, they share the concept of lexicalized grammars described above.

Several groups are developing grammars and parsers based on lexicalized grammar formalisms. Studies listed below are aiming at applying parsers to practical applications that require in-depth syntactic structures and semantic representations.

[1] Asterisks denote ungrammatical sentences.

[2] For simplicity, we assume that grammar rules are binary.

LinGO English Resource Grammar (ERG) (Copestake and Flickinger, 2000; Flickinger, 2002) is the largest HPSG grammar for English and has developed at Stanford University and in other groups. Following the theory of HPSG, the developers have been concentrating on the description of precision linguistic constraints, as well as the expanding of a grammar to be used in practical applications. Since the development of HPSG grammars is a labor-intensive task, they have been developing several tools to support grammar development: LKB (Copestake, 1992) for providing an interactive environment for grammar development, `[incr tsdb()]` (Oepen and Carroll, 2000; Oepen et al., 2004) for grammar profiling and treebanking, and PET (Callmeier, 2000) for efficient run-time parsing.

Although the LinGO ERG is an English grammar, they are also interested in explicating language-independent structures of natural language grammars. The Grammar Matrix (Bender and Flickinger, 2005; Bender et al., 2002) is a framework for the development of wide-coverage HPSG grammars for various languages. They provide a starter kit to create an initial language-dependent grammar given some specifications of a language, such as a word order and complement categories. Their purpose is to assist the rapid development of grammars in other languages, as well as to share experiences in the development of various grammars. MRS (Copestake et al., 1995, 1999) is a semantic representation they assume in their grammars, and is incorporated into the Matrix. With the help of the Matrix, grammar writers can develop a grammar from scratch in which feature structure definitions and semantic representations are common to other grammars developed with the Matrix.

JACY (Siegel, 2000; Siegel and Bender, 2002) is a Japanese HPSG grammar which has been being developed at DFKI for around ten years, following the Matrix and MRS. It has a large lexicon of more than 35,000 words. A German grammar has also been developed using the same methodology (Müller and Kasper, 2000). This grammar has a lexicon of more than 50,000 words. NorSource, a Norwegian grammar, has a lexicon of 84,240 words (Hellan and Haugereid, 2003), owing to the conversion of existing lexicon resources. With the same tools and methodology, other groups are developing grammars of Modern Greek (Kordoni and Neu, 2003), Spanish, Korean, and Italian, although they are smaller than the others.

The LinGO ERG has a large lexicon as it has been being developed for more than ten years. The grammar includes a manually maintained lexicon of more than ten thousand words. However, the coverage of the grammar against real-world texts is still unsatisfactory. Baldwin et al. (2004a,b) reported the coverage of the ERG against the British National Corpus (BNC) (Burnard, 2000). They found that for 32% of sentences the ERG had full lexical span (i.e., lexical entries were assigned to all words in a sentence) without unknown word handling except for number expressions and proper names. Of the sentences with full lexical span, the ERG could generate at least one parse tree for 57%, and 83% of them included a correct parse. This means that the proportion of correctly parsed sentences is around 15%. The other grammars are at a similar level.

The above projects are also interested in the development of treebanks for the training of disambiguation models. Their idea is that when we already have a grammar, a treebank is rapidly developed by parsing a corpus with the grammar. With the treebanking tool, `[incr tsdb()]`, grammar developers can construct a treebank by only selecting a correct parse tree for each sentence. Until now, LinGO Redwoods (Oepen et al., 2002b) for English and Hinoki (Bond et al., 2004) for Japanese have been developed. Research on disambiguation models is also ongoing (Baldridge and Osborne, 2003; Malouf and van Noord, 2004; Oepen et al., 2002b; Toutanova et al., 2004; Toutanova and Manning, 2002), although experiments have been small due to limitations of the size of the treebanks and the coverage of the grammars.

SLUNG is another Japanese HPSG grammar developed at the University of Tokyo from scratch (Mitsuishi, 1998; Mitsuishi et al., 1998). This grammar achieved impressively high coverage. The grammar could parse 98.4% sentences of the EDR corpus (EDR, 1996), which is a collection of

newswire sentences (Kanayama et al., 2000). The high coverage owed much to the concept of *underspecification*; a limited number of functional words, e.g. postpositions and auxiliary verbs, were assigned word-specific lexical entries, while open class words, e.g. nouns and verbs, were assigned lexical entries specific to each part-of-speech. That is, the grammar did not distinguish word-specific constraints of open class words, such as subcategorization frames. This means that the grammar provided only coarse syntactic structures. In addition, the grammar lacked any semantic representations. It is therefore insufficient for computing sentence-to-meaning correspondences.

The Alpino grammar is a wide-coverage HPSG grammar for Dutch (Bouma et al., 2001). The grammar includes a lexicon of around 100,000 words and components to handle unknown words, number expressions, etc. The grammar has wide-coverage and high accuracy against the Alpino treebank, which consists of newspaper sentences (Malouf and van Noord, 2004).

Another interesting approach to developing HPSG grammars is grammar conversion from LTAG into HPSG (Tateisi et al., 1998; Yoshinaga and Miyao, 2001, 2002; Yoshinaga et al., 2001, 2003, 2002). With this method, we can reuse existing LTAG grammars. However, the performance of the grammar is bound by that of original grammars. As described below, the coverage of existing LTAG grammars is still insufficient because the development of LTAG grammars is no less difficult than HPSG grammars.

The biggest group in the LTAG community is the XTAG Research Group (XTAG Research Group, 2001). Their largest grammar is an English grammar, which includes a large lexicon of more than 31,000 word stems, although the coverage of the grammar against real-world texts is still limited. A Korean grammar is also under development (Yoon et al., 2000). Recently, algorithms were proposed for the automatic induction of LTAG from treebanks (Chen and Vijay-Shanker, 2000; Chiang, 2000; Xia, 1999). They extracted elementary trees (lexical entries in LTAG) by splitting parse trees of Penn Treebank. Different from hand-crafted grammars, treebank grammars attained impressively high coverage against real-world texts. Probabilistic models were also developed using treebanks. However, automatically extracted grammars were not theoretically formulated in conformity with linguistic theories. Few linguistic formalizations are exploited in grammar development, and the grammars lack explanation of grammaticality and linguistically motivated formulation. Details and problems of automatic grammar induction will be discussed in Section 3.5.

The LFG research community has many groups involved in the development of grammars including English, French, German, Japanese, Norwegian, and Urdu (Butt et al., 2002). Similar to the HPSG community, they have common development tools developed in the Parallel Grammar (ParGram) project. The XLE system, an English LFG parser, succeeded in the parsing of real-world texts (Riezler et al., 2002), owing to extensive human labor that lasted for over a decade and various techniques for robust parsing. The techniques include methods of unknown word handling, creating fragment parses, and giving up processing when the parsing of a subtree exceeded some threshold. Recently, they reported that the XLE system performed better than the state-of-the-art CFG parser (Collins, 1997) in terms of labeled dependency relations (Kaplan et al., 2004). Besides, automatic induction of LFG grammars from Penn Treebank has been proposed (Burke et al., 2004; Cahill et al., 2003, 2002; Frank et al., 2003b). Their aim was to automatically annotate parse trees (corresponding to c-structures in the theory of LFG) with *functional schemata*, which are unification-based construction rules in LFG. That is, they automated the development of LFG-style construction rules. The induced grammars achieved high coverage against Penn Treebank, as did the work on LTAG.

CCG is also extensively studied in linguistics and computer science. Hockenmaier and Steedman (2002a) proposed an algorithm of automatically extracting a CCG from Penn Treebank. Their idea was basically following LTAG's; Penn Treebank was automatically converted into CCG derivations, and lexical categories (lexical entries in CCG) and construction rules were extracted from the derivations. The extracted grammar attained high coverage against Penn Treebank.

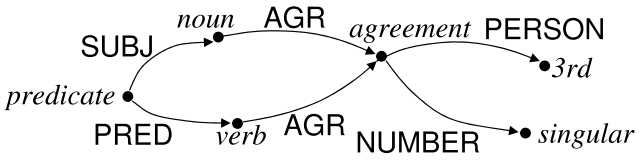


Figure 2.3: Feature structure

2.2 Head-Driven Phrase Structure Grammar

Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1987, 1994; Sag and Wasow, 1999) is a syntactic theory that follows the lexicalist framework. HPSG describes linguistic structures with *typed feature structures* (Carpenter, 1992). That is, lexical entries and grammar rules are both described with typed feature structures. In the remainder of this section, we first provide a mathematical definition of typed feature structures, which follows Carpenter (1992). We then describe lexical entries and grammar rules in HPSG.

2.2.1 Typed Feature Structures

Typed feature structure is a mathematical device for expressing symbolic and structured constraints. The definition of feature structures in this thesis follows that of Carpenter (1992). Typed feature structures are rooted directed labeled graphs, whose nodes are labeled with *types*, and whose edges are labeled with *features*. Figure 2.3 shows an example feature structure that represents a predicate argument structure and the agreement in it. Sans-serif uppercase labels are assigned to edges and express features, while lowercase labels are to vertices and denote types. Intuitively, features represent attributes of a certain entity, and types express their values. For example, NUMBER is one of the attributes of *agreement*, and its value is *singular*. With two features, PERSON and NUMBER, the information of *agreement* is expressed. It should be noted that the value of SUBJ and PRED is expressed as the same node in the graph. This is called *reentrancy* or *structure-sharing*, and expresses that two attributes denote identical information.

As shown in the example, typed feature structures express structured information with a bundle of features. They are widely used for modeling various linguistic phenomena in computational linguistics. Especially, syntactic theories are extensively studied, and their fruits are known as unification-based grammars such as HPSG.

Let us suppose that $\mathcal{T}$ is a finite set of types and $\sqsubseteq$ is a partial order defined on $\mathcal{T}$. For types $\sigma, \tau \in \mathcal{T}$, σ *subsumes* or *is more general than* or *is a supertype of* τ, when $\sigma \sqsubseteq \tau$. Conversely, τ *is subsumed by* or *is more specific than* or *is a subtype of* σ. Given a partial order $\sqsubseteq$, we define *a least upper bound* of two types. It is the most general common subtype of two types.

Definition 2.2 (Least Upper Bound) *For types $\tau_1, \tau_2 \in \mathcal{T}$, τ_1 and τ_2 are* consistent *if they have* a common subtype *or* an upper bound σ'*, such that $\tau_1 \sqsubseteq \sigma' \wedge \tau_2 \sqsubseteq \sigma'$.*

A least upper bound σ *of τ_1 and τ_2 is then defined as follows:*

- *σ is an upper bound of τ_1 and τ_2.*
- *For any upper bound σ', $\sigma \sqsubseteq \sigma'$.*

Let $lub(\tau_1, \tau_2)$ denote a set of least upper bounds.

Given the above definition, a type hierarchy is defined as having a unique least upper bound for every consistent types.

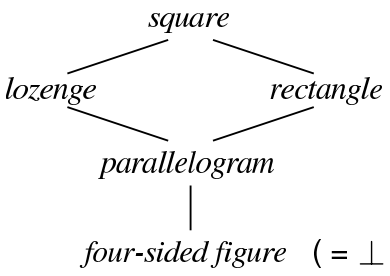


Figure 2.4: Type hierarchy representing a class of four-sided figures

Definition 2.3 (Type Hierarchy) *Type hierarchy $\mathcal{H}$ is a pair $\langle \mathcal{T}, \sqsubseteq \rangle$ that satisfies the following condition: $\forall \tau_1, \tau_2 \in \mathcal{T}.(lub(\tau_1, \tau_2) = \emptyset \vee \exists \sigma \in \mathcal{T}.lub(\tau_1, \tau_2) = \{\sigma\})$.*

By following Carpenter (1992), we denote the most general type in a type hierarchy by $\bot$ (bottom).

By this condition, we can define an operation to compute a least upper bound of two types as a partial function. The operation is called *type unification*.

Definition 2.4 (Type Unification) Unification *of types $\tau_1, \tau_2 \in \mathcal{T}$ is an operation to compute the least upper bound of τ_1 and τ_2. Type unification of τ_1 and τ_2 is denoted as $\tau_1 \sqcup \tau_2$. That is,*

$$\tau_1 \sqcup \tau_2 = \begin{cases} \sigma & \text{if } lub(\tau_1, \tau_2) = \{\sigma\} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

As described above, $\sqcup$ is a partial function: $\mathcal{T} \times \mathcal{T} \rightarrow \mathcal{T}$. This characteristic is beneficial for efficient processing, because the result of unification is deterministically computed. In fact, by precomputing the results of unification of all pairs of types, we can obtain type unification by looking up a two-dimensional or a hash table.

For example, Figure 2.4 shows a type hierarchy of four-sided figures, where *square*, *rectangle* and so on denote types. Lines denote IS-A relations (or direct subtype-supertype relations) where the upper side of a line is more specific than the lower side. In Figure 2.4, $\sqsubseteq$ is formed by reflexive transitive closures of IS-A relations and $\bot$ is *four-sided figure*. Type unification is an operation to find the lowest common subtype. For example, type unification of *lozenge* and *rectangle* results in *square*. Note that *square* has all of the information of both *lozenge* and *rectangle*.

A type hierarchy can express various structured data: lists, conjunction/disjunction of a finite set, and IS-A relations like thesauri. The effectiveness of type hierarchies is discussed in detail in other literature (Carpenter, 1992; Flickinger, 2002).

Given a set of features, $\mathcal{F}$, and a set of types, $\mathcal{T}$, typed feature structures are defined as follows.

Definition 2.5 (Typed Feature Structure) *A typed feature structure is a tuple, $F = \langle Q, \bar{q}, \theta, \delta \rangle$, where:*

- *Q: a finite set of nodes*
- *$\bar{q} \in Q$: the root node*
- *$\theta : Q \rightarrow \mathcal{T}$: a total function for typing*
- *$\delta : \mathcal{F} \times Q \rightarrow Q$: a partial function for feature values*

Let $\mathcal{FS}$ denote a collection of typed feature structures.

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$
$\bar{q} = q_0$
$\theta = \{q_0 \rightarrow \textit{predicate}, q_1 \rightarrow \textit{noun}, q_2 \rightarrow \textit{verb}, q_3 \rightarrow \textit{agreement}, q_4 \rightarrow \textit{3rd}, q_5 \rightarrow \textit{singular}\}$
$\delta = \{(\mathsf{SUBJ}, q_0) \rightarrow q_1, (\mathsf{PRED}, q_0) \rightarrow q_2, (\mathsf{AGR}, q_1) \rightarrow q_3, (\mathsf{AGR}, q_2) \rightarrow q_3,$
$(\mathsf{PERSON}, q_3) \rightarrow q_4, (\mathsf{NUMBER}, q_3) \rightarrow q_5\}$

Figure 2.5: Definition of a typed feature structure

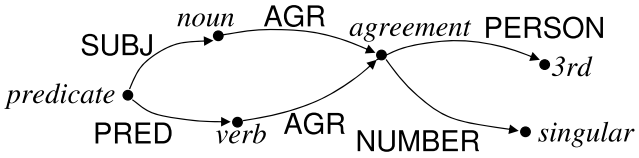


Figure 2.6: Graph representation of a typed feature structure (same as Figure2.3)

$$\begin{bmatrix} \textit{predicate} \\ \mathsf{SUBJ} \begin{bmatrix} \textit{noun} \\ \mathsf{AGR}\ \boxed{1} \begin{bmatrix} \textit{agreement} \\ \mathsf{PERSON}\ \textit{3rd} \\ \mathsf{NUMBER}\ \textit{singular} \end{bmatrix} \end{bmatrix} \\ \mathsf{PRED} \begin{bmatrix} \textit{verb} \\ \mathsf{AGR}\ \boxed{1} \end{bmatrix} \end{bmatrix}$$

Figure 2.7: AVM representation of a typed feature structure

Two notations are often used to describe feature structures. One is the most intuitive way, a graph notation. A feature structure is described by a rooted directed labeled graph, where Q is a set of nodes and θ determines labels on the nodes. δ is represented by arcs from q to $\dot{q}$, which is labeled by f if $\delta(f, q) = q'$. For instance, Figure 2.5 shows a definition of a feature structure, and Figure 2.6 shows its graph notation (the same figure as Figure 2.3). The other notation is *an attribute-value matrix (AVM)*. In this notation, the values of nodes are drawn following feature names. A complex value is described with a bracketed entry, and an atomic value is written in an italic string. Types are denoted at the upper left corner of each entry, while they are sometimes omitted when obvious. Figure2.7 shows the AVM notation of the feature structure defined in Figure 2.5. In the AVM notation, reentrancy is denoted as a boxed number, such as $\boxed{1}$.

Next, *subsumption relation* is defined over typed feature structures. The fundamental idea is essentially the same as that of types. That is, both represent a relation that a feature structure (or a type) is more informative or specific than the one it is subsumed by.

Definition 2.6 (Subsumption) *For typed feature structures $F = \langle Q, \bar{q}, \theta, \delta \rangle$ and $F' = \langle Q', \bar{q}', \theta', \delta' \rangle$, F* subsumes *F', if there is a total function $h\colon Q \rightarrow Q'$ such that:*

- $h(\bar{q}) = \bar{q}'$
- $\theta(q) \sqsubseteq \theta'(h(q))$ *for* $\forall q \in Q$
- $h(\delta(f, q)) = \delta'(f, h(q))$ *for* $\forall q \in Q, \forall f \in \mathcal{F}$ *such that* $\delta(f, q)$ *is defined.*

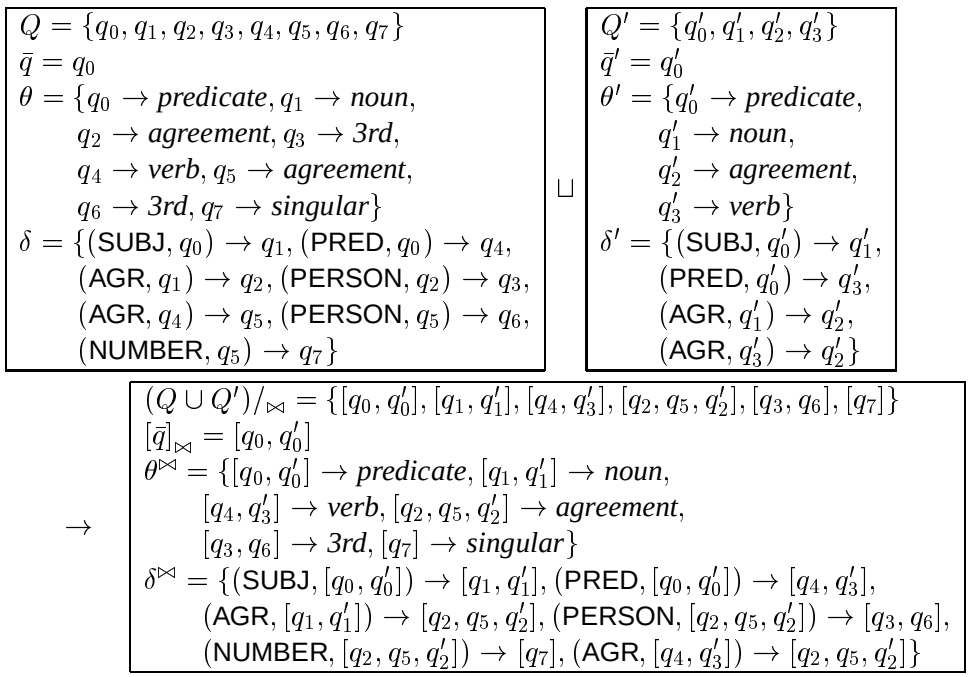


Figure 2.8: Unification of typed feature structures

According to this definition, the subsumption ordering of typed feature structures forms a partial order (Carpenter, 1992).

Unification is an operation defined on pairs of typed feature structures. The fundamental idea is essentially the same as that of types. That is, the goal of both operations is to find a typed feature structure (or a type) containing all of the information of input feature structures (or types).

Definition 2.7 (Unification) *Suppose $F, F' \in \mathcal{FS}$ are feature structures such that $F = \langle Q, \bar{q}, \theta, \delta \rangle$ and $F' = \langle Q', \bar{q}', \theta', \delta' \rangle$, and $Q \cap Q' = \phi$. We first define an equivalence relation $\bowtie$ on $Q \cup Q'$ as the least equivalence relation such that:*

- $\bar{q} \bowtie \bar{q}'$
- $\delta(f, q) \bowtie \delta(f, q')$ *if both are defined and $q \bowtie q'$*

The unification *of F and F' is then defined as:*

$$F \sqcup F' = \langle (Q \cup Q')/_{\bowtie}, [\bar{q}]_{\bowtie}, \theta^{\bowtie}, \delta^{\bowtie} \rangle$$

where:

$$\theta^{\bowtie}([q]_{\bowtie}) = \sqcup\{(\theta \cup \theta')(q') | q' \bowtie q\}$$

$$\delta^{\bowtie}(f, [q]_{\bowtie}) = \begin{cases} [(\delta \cup \delta')(f, q)]_{\bowtie} & \text{if } (\delta \cup \delta')(f, q) \text{ is defined} \\ undefined & otherwise \end{cases}$$

if all of the type unifications in the definition of $\theta^{\bowtie}$ exist. $F \sqcup F'$ is undefined otherwise.

Definition 2.7 tells that the root node of the result of a unification is the joint of the roots of input feature structures. Nodes in the inputs are then traversed from the root nodes. If current nodes are

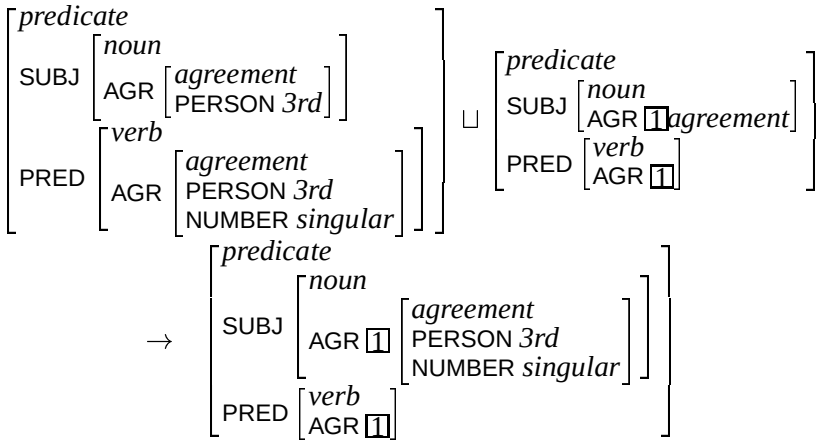


Figure 2.9: Unification of typed feature structures (AVM notation)

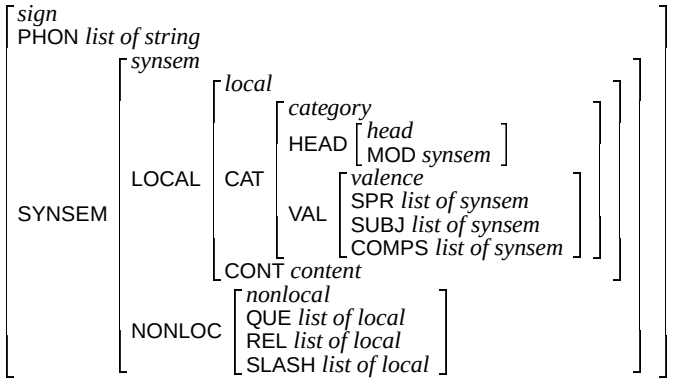


Figure 2.10: Typed feature structure of an HPSG sign

already unified and have features with the same label, the nodes followed by the features are also unified.

Figure 2.8 shows an example of unification, and Figure 2.9 represents the same operation in an AVM notation. Since both root nodes have the type *predicate*, they are successfully unified. Next, two features, SUBJ and PRED, are traversed. Since the second input tells that SUBJ and PRED are shared, they are also shared in the output. Furthermore, their values are also unified recursively. That is, since the first feature structure has features, PERSON, in SUBJ and PRED, their values are also unified. They have the same value, *3rd*. Hence, the unification of the whole structures of the inputs succeeds as in the figures. It should be noted that the definition of the output feature structure in Figure 2.8 is equivalent to the one in Figure 2.5.

2.2.2 Signs and Lexical Entries

In HPSG, linguistic entities, such as words and phrases, are represented by *signs*. Signs are formal representations of combinations of phonological forms, syntactic structures, and semantic structures, and express which phonological form signifies which syntactic/semantic structure. Signs in HPSG are denoted with typed feature structures. Figure 2.10 shows the definition of a typed feature structure to represent a sign as defined in Pollard and Sag (1994). As discussed in Section 2.1, in modern

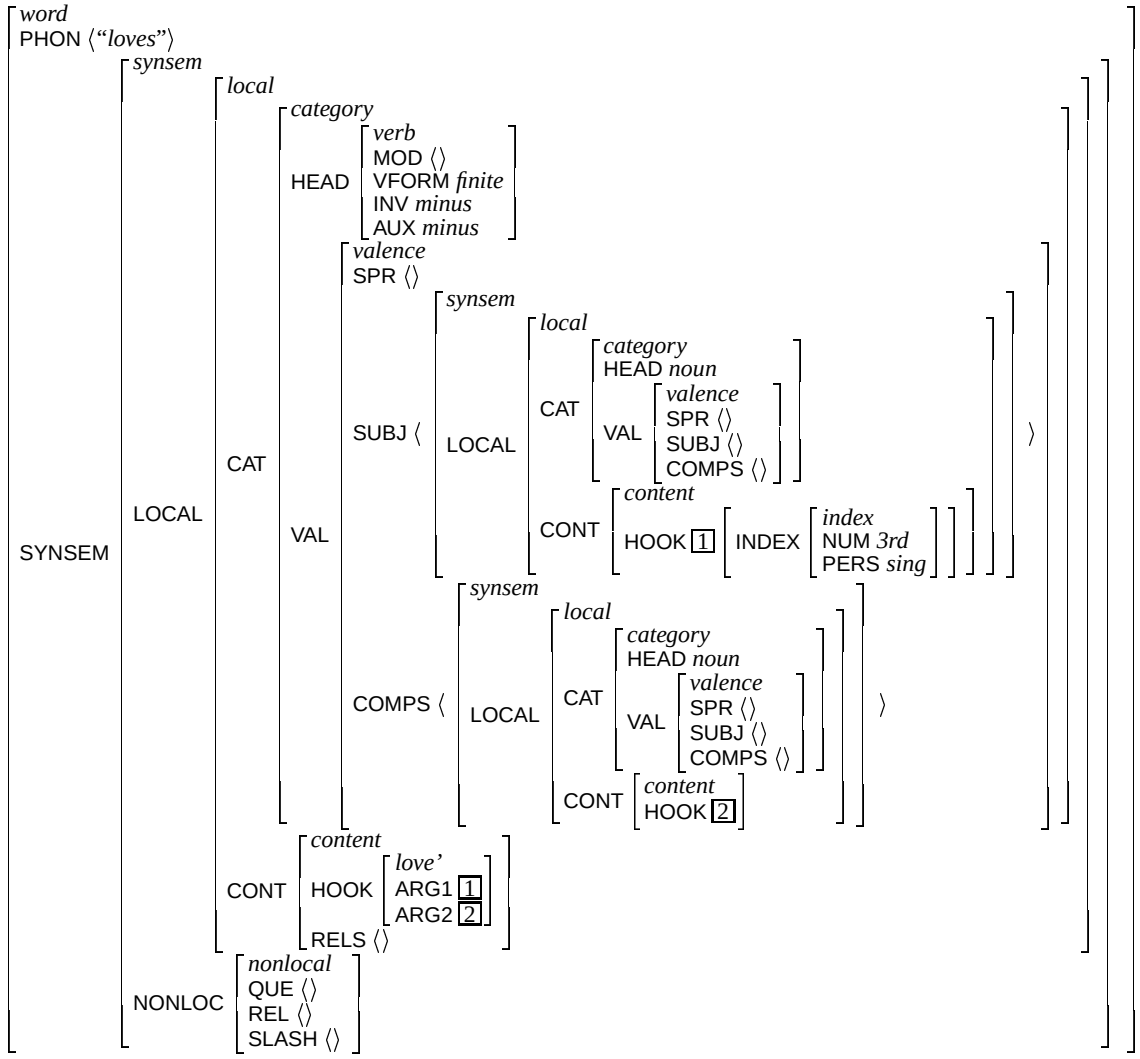


Figure 2.11: Lexical entry for transitive verb "*loves*"

syntactic theories including HPSG, a grammar is *lexicalized*. That is, lexical entries, i.e., signs of words, involve most of the grammatical constraints.

Let us examine the structure of a lexical entry of HPSG. Figure 2.11 shows a lexical entry for "*loves*" in HPSG. The intuitive roles of the features are given below.

PHON represents phonological information, or is just a list of strings appearing in a sentence.

SYNSEM represents syntactic and semantic information. The value of this feature has all of the information of syntactic/semantic constraints on words or phrases.

LOCAL represents all information of constraints to restrict local dependencies. Local dependencies are syntactic relations regulated by subcategorization and modification. See the description of NONLOC to contrast non-local dependencies.

NONLOC is used in order to restrict unbound dependencies, which are the relations beyond the arbitrary number of clauses. For example, in the following sentence, underlined constituents with the same index i denote the same semantic entity.

Who$_i$ do you think he likes ____$_i$?

In this example, the object of "*likes*" is usually put in the position just after "*likes*" in declarative sentences. However, in *wh*-questions, it is moved to the beginning of a sentence. The constraints of a moved entity are exported through this feature to remote places in a sentence.

CAT represents a syntactic category, such as NP and VP, of words or phrases.

CONT represents context-independent meanings directly related to semantic interpretation. Various theories were proposed to represent the semantics of a sentence. In the HPSG community, Minimal Recursion Semantics (MRS) (Copestake et al., 1995, 1999) has been the de facto standard. MRS is a variant of a higher-order predicate logic, and its advantage is that it can represent scope ambiguities underspecified. In this thesis, we represent semantics with predicate argument structures, which will be described in Section 2.2.4. Predicate argument structures are one of the simplest semantics, and represent the semantic relations of words in a sentence as a labeled graph structure. In Figure 2.11, HOOK denotes a structure of a main predicate, RELS is a list of other predicate argument structures modifying the main predicate, and ARGX represents arguments of a predicate[3].

HEAD represents information percolated from *a head daughter*, which takes a principal role to determine the structure of a phrase. For example, a part of speech and a verbal form are specified in this feature.

VAL describes valence constraints by three sub features, SPR, SUBJ, and COMPS. Each has a list as its value, which expresses a subcategorization frame. They constrain the structure of possible words or phrases as specifiers, subjects, and complements. One of the list elements is unified with another sign's SYNSEM to construct a larger phrase. This feature is called *a selecting feature* because it determines which feature structure should be selected in order to construct a larger phrase.

MOD is another selecting feature that represents a sign to be modified by this phrase. For example, adjectives have a noun in this feature since they modify noun phrases. Since this feature is put in HEAD, it is percolated from a head daughter.

VFORM, INV, and AUX appear only for verbs. VFORM describes a verbal form. INV expresses whether the phrase is inverted or not. AUX represents whether it is an auxiliary verb or not. Other features are defined for other parts of speech.

Considering these roles of features, we can now interpret the information represented in Figure 2.11 as follows.

A word "*loves*" is a finite verb. A subject and an object are not inverted. It is not an auxiliary verb. It takes a noun phrase as a subject, another noun phrase as a complement, and no specifiers.

As you can see in figure 2.11, signs in HPSG are very large to be shown in a text. For simplicity, in this thesis, we will present signs in a reduced form as shown in Figure 2.12 when it is not confusing. We will only show features that are relevant to an explanation, expecting that readers can fill in the values of suppressed features. For example, in Figure 2.12, we suppose that SPR and NONLOC features have empty lists. Although this representation is not mathematically nor linguistically strict, we prioritize readability when feature values are trivially guessable from contexts.

[3] We borrow a notation of MRS rather than that of Pollard and Sag (1994), although semantics is represented by predicate argument structures.

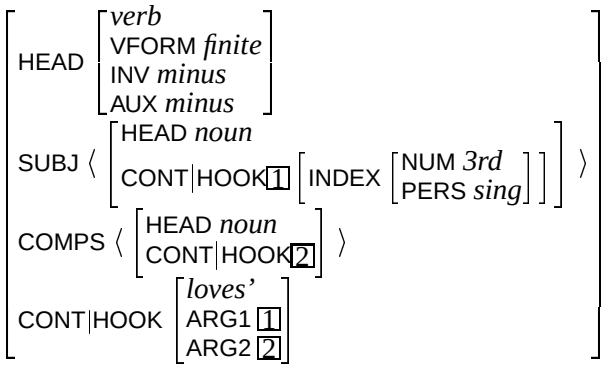


Figure 2.12: Simplified representation of the lexical entry in Figure 2.11

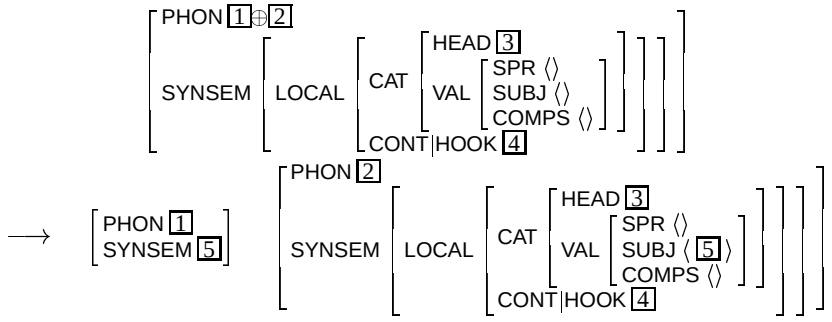


Figure 2.13: Head Subject Schema

2.2.3 Grammar Rules and Linguistic Principles

Grammar rules describe conditions that hold between the structure of a mother and those of her daughters. In CFGs, grammar rules are construction rules; that is, each grammar rule determines a mother symbol and a combination of daughter symbols. Grammar rules are applied when constructing a larger phrase from daughter phrases.

In HPSG, grammar rules are classified into *schemas* and *principles*. Schemas are construction rules, and correspond to grammar rules in CFGs. Figure 2.13 shows a schema in an HPSG grammar, which is called *the Head Subject Schema*. This schema is applied to subject-head constructions, such as for a verb taking its subject. In this rule, candidate daughters are unified to the left and the right signs of the right-hand side of the rule, and the mother is then constructed as in the left-hand side.

The constraints represented in this schema are then summarized as follows.

- PHON of the mother is a concatenation of PHONs of the daughters.
- SUBJ of the right daughter is a list of one-element, and the element is unified with the value of SYNSEM of the left daughter.
- COMPS and SPR of the right daughter are empty lists.
- SUBJ, COMPS, and SPR of the mother are empty lists.
- HEAD and CONT of the right daughter are percolated to the mother.

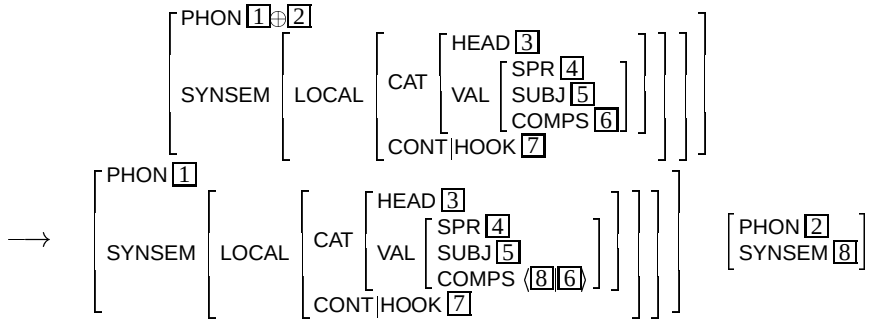


Figure 2.14: Head Complement Schema

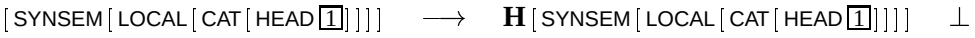


Figure 2.15: Head Feature Principle

Figure 2.14 shows the definition of the Head Complement Schema. This schema is defined for head-complement constructions, such as a verb or a preposition taking an object. The essential difference from the Head Subject Schema is that an element of COMPS of the left daughter selects the SYNSEM of the right daughter.

By comparing Figure 2.13 and 2.14, we observe some common constraints. For example:

- The value of PHON is a concatenation of PHONs of the daughters.
- HEAD and CONT are percolated from one of the daughters.
- The values of SPR, SUBJ, and COMPS are percolated from one of the daughters when an element of a list is not used for selecting the other daughter.

Such constraints common to schemas are formalized as *principles*. That is, principles are the formulation of general regularities of syntactic structures. For example, Figure 2.15 shows the most important principle, *the Head Feature Principle* (HFP). This principle regulates that HEAD of the mother must be percolated from one of the daughters. The prefix, **H**, denotes the head daughter, which is a key concept of the HPSG theory. HPSG supposes that in every construction one of the daughters takes a principal role in determining syntactic structures. We therefore denote a head daughter as **H** in the definition of schemas and principles. When **H** appears in a principle, the order of daughters is irrelevant. A daughter with **H** is a head daughter, and the other is non-head. When it appears in schemas, it determines which daughter is a head.

Figure 2.16 lists other principles defined in Pollard and Sag (1994)[4]. The Phonology Principle simply concatenates PHONs of daughters. The Semantic Principle percolates a semantics of the head daughter to the mother. The Valence Principle percolates VAL features to the mother. Slashes represent *default* values, which may be overwritten by constraints of other schemas/principles[5]. For

[4] Some of the definitions are slightly different from the original ones in Pollard and Sag (1994) because we eliminated constraints irrelevant to the theme of this thesis and modified the definitions by following the improvements of the HPSG theory.

[5] Computational/theoretical formulation of defaults in typed feature structures is a difficult problem and has been studied by many researchers. Details are not described here and readers should refer to other literature (Bouma, 1990; Carpenter, 1993; Copestake, 1993; Lascarides et al., 1996; Lascarides and Copestake, 1999; Ninomiya et al., 2002; Russell et al., 1991). The definition by Lascarides et al. (1996) is adopted in the current theory of HPSG.

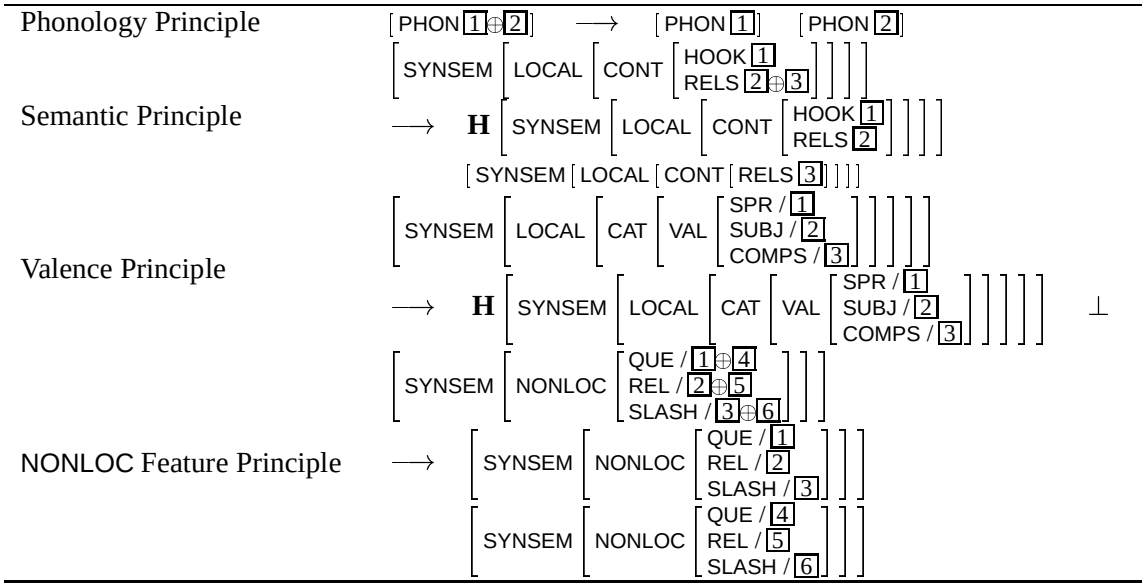


Figure 2.16: Principles

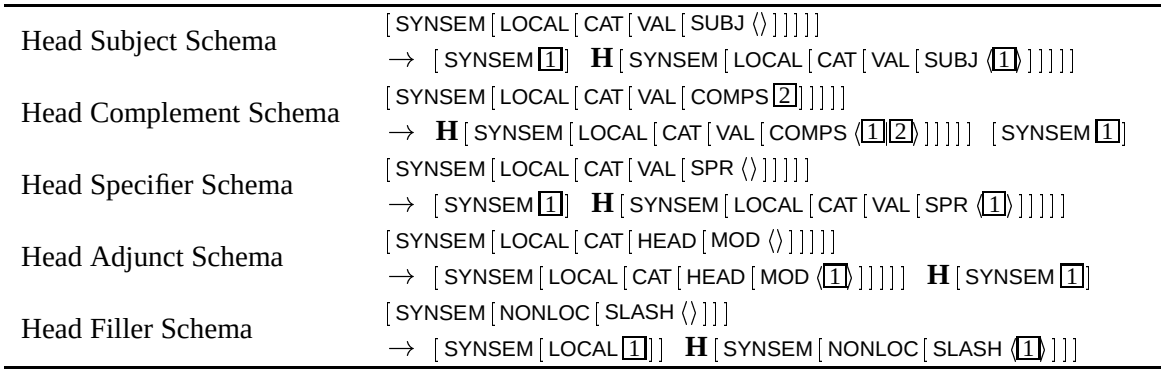


Figure 2.17: Schemas

example, the Head Subject Schema overwrites the values of SUBJ. The NONLOC Feature Principle determines the percolation of NONLOC features.

With principles, schemas are re-defined simply because construction-independent constraints can be omitted. Figure 2.17 lists all schemas defined in Pollard and Sag (1994)[6]. The schemas listed in the figure are explained below.

Head Specifier Schema explains the relation between a specifier (determiner) and its head; e.g., "*a boy*". Common nouns have non-empty list in SPR, and it must be consumed by a determiner.

Head Adjunct Schema explains modification constructions. Different from head subject/complement constructions, a non-head daughter selects the other in this construction. A non-head daughter has a non-empty list in its MOD, and the constraint of MOD is consumed by a head.

Head Filler Schema explains non-local dependencies. For example, in the following sentence,

[6]Head Marker Schema is omitted because it is not used in the current theory.

Who$_i$ do you think he likes ____$_i$?

we should regard that "*who*" is moved from the object position of "*likes*". In order to explain the dependency between "*likes*" and "*who*", the object of "*likes*" is represented in SLASH of the lexical entry of "*likes*". The value of SLASH is percolated to the mother by the NONLOC Feature Principle until it meets an antecedent. Finally, Head Filler Schema checks the constraint of SLASH with the sign of "*who*", and SLASH is consumed.

In the schemas listed in Figure 2.17, constraints offered by principles are omitted. A schema is applied to daughters to construct a larger phrase, and all of the principles are also applied to them. To be precise, in the theoretical formulation of HPSG, the application of schemas is also controlled by another important principle, *the ID Principle*. Although this principle is not formalized by typed feature structures, it says that every construction must satisfy one of the schemas. That is, the HPSG theory describes syntactic structures only with principles, and construction rules are just derived from the principles.

HPSG captures generality across construction rules using principles, while CFGs cannot. Since the heart of the HPSG theory is to explicate generic constraints to explain syntactic structures of natural languages, the arrangement of principles is essential in HPSG. Hence, grammar rules in the HPSG theory are no more construction rules but principles that explain generic syntactic regularities.

In what follows, we denote principled rules of syntactic structures as *linguistic principles* and distinguish linguistic principles from construction rules. Obviously, linguistic principles include HPSG principles, and may also involve other rules such as lexical rules and generic constraints on templates of lexical entries. We need to focus on linguistic principles when we discuss linguistic aspects of HPSG analysis. In particular, in this thesis, linguistic principles will take an important role in the development of grammars.

However, we may focus only on the constructional aspects of rules when we discuss parsing algorithms. We therefore denote construction rules as *grammar rules* in this thesis. That is, we consider grammar rules in HPSG as an extension of CFG rewriting rules. In theory, grammar rules are derivatives of linguistic principles, but we will ignore underlying linguistic principles. In this thesis, when we discuss probabilistic modeling and parsing algorithms of HPSG, we will focus only on grammar rules.

2.2.4 Predicate Argument Structures

Predicate argument structures are semantic representations of natural language sentences. It expresses a semantics of a sentence by the semantic dependencies of words in the sentence. Figure2.18 shows an example of a predicate argument structure of a sentence "*She ignored the fact that I wanted to dispute*." Figure 2.19 is a feature structure representation of the same semantics. Each node in a graph represents a word in a sentence, and an edge describes a semantic dependency of two words. ARG1 and ARG2 represent a semantic subject and object, respectively. Note that predicate argument structures can express non-local dependencies such as a control expression ("*I*" and "*dispute*") and an unbounded dependency ("*fact*" and "*dispute*").

Predicate argument structures are formally defined as follows.

Definition 2.8 (Predicate argument structure) *A predicate argument structure of sentence* $\mathbf{w}$ *is defined as a labeled graph structure,* $A = \{\langle w_h, w_n, \pi, \rho\rangle\}$*, where*

- $w_h, w_n \in \mathbf{w}$ *are a predicate and an argument word of the dependency,*
- π *is a predicate type (e.g., transitive_verb), and*

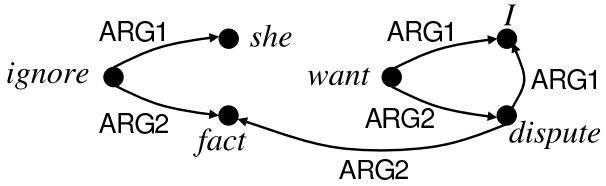


Figure 2.18: Predicate argument structure

$$\begin{bmatrix} \textit{ignore} \\ \text{ARG1}\ \textit{she} \\ \text{ARG2}\ \boxed{1}\ \textit{fact} \end{bmatrix} \qquad \begin{bmatrix} \textit{want} \\ \text{ARG1}\ \boxed{2}\ \textit{I} \\ \text{ARG2} \begin{bmatrix} \textit{dispute} \\ \text{ARG1}\ \boxed{2} \\ \text{ARG2}\ \boxed{1} \end{bmatrix} \end{bmatrix}$$

Figure 2.19: Feature structure representation of the predicate argument structure in Figure2.18

$$\begin{bmatrix} \textit{mrs} \\ \text{TOP}\ \textit{h2} \\ \text{RELS}\ \{ \begin{bmatrix} \textit{she} \\ \text{LBL}\ \textit{h1} \\ \text{ARG0}\ \textit{x1} \end{bmatrix}, \begin{bmatrix} \textit{ignore} \\ \text{LBL}\ \textit{h2} \\ \text{ARG0}\ \textit{e1} \\ \text{ARG1}\ \textit{x1} \\ \text{ARG2}\ \textit{x2} \end{bmatrix}, \begin{bmatrix} \textit{the} \\ \text{LBL}\ \textit{h3} \\ \text{ARG0}\ \textit{x2} \\ \text{RSTR}\ \textit{h4} \\ \text{BODY}\ \textit{h5} \end{bmatrix}, \begin{bmatrix} \textit{fact} \\ \text{LBL}\ \textit{h4} \\ \text{ARG0}\ \textit{x2} \end{bmatrix}, \begin{bmatrix} \textit{I} \\ \text{LBL}\ \textit{h6} \\ \text{ARG0}\ \textit{x3} \end{bmatrix}, \begin{bmatrix} \textit{want} \\ \text{LBL}\ \textit{h7} \\ \text{ARG0}\ \textit{e2} \\ \text{ARG1}\ \textit{x3} \\ \text{ARG2}\ \textit{h8} \end{bmatrix}, \begin{bmatrix} \textit{dispute} \\ \text{LBL}\ \textit{h8} \\ \text{ARG0}\ \textit{e3} \\ \text{ARG1}\ \textit{x3} \\ \text{ARG2}\ \textit{x2} \end{bmatrix} \} \end{bmatrix}$$

Figure 2.20: Minimal Recursion Semantics

- *ρ is a label of the dependency of w_h and w_n.*

Predicate argument structures are one of the simplest representations of semantics of natural languages. Various semantic representations have been studied, not limited to the field of lexicalized grammars.

In the community of HPSG, Minimal Recursion Semantics (MRS) (Copestake et al., 1995, 1999) is becoming a de facto standard. In MRS, the semantics shown in Figure 2.18 is represented as in Figure 2.20. Each element in the set RELS denotes *a relation* introduced by a word. A superficial difference from predicate argument structures is that MRS represents connections among *relations* in a Davidsonian format. The value of ARGX is a variable, and relations are connected through having the same variable. For example, ARG0 of *I* is *x3*, and *want* and *dispute* have the same variable in ARG1. This means that the first argument of "*want*" and "*dispute*" is "*I*." While this is necessary for a flat representation of semantics, predicate argument structures can be converted into this format automatically.

An essential difference is that MRS represents scopes of relations. Scope information is necessary in some situations or applications. For example, the difference in meaning between the following sentences requires scope information.

- *Sometimes everybody dances.*
- *Everybody sometimes dances.*

Predicate argument structures of the above sentences are equivalent because the subject of "*dances*" is "*everybody*" and "*sometimes*" modifies "*dances*". The difference of the meanings is expressed as the difference of the scopes of "*sometimes*" and "*everybody*." That is, the difference is whether the scope of "*sometimes*" includes the meaning of "*everybody*," or not.

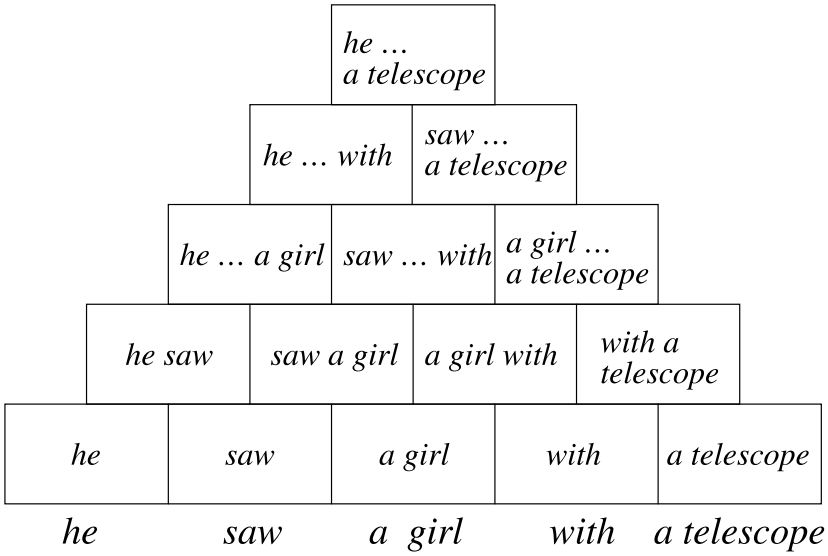


Figure 2.21: CKY chart

In Figure 2.20, LBL is assigned to each relation to distinguish the level of a scope. Relations having the same LBL values are in the same scope. TOP represents the top-most scope. For example, in Figure 2.20, TOP is equivalent to LBL of *ignore*, and this means that the top-most scope of the sentence is that of *ignore*. The value of ARG2 of *want* is equivalent to LBL of *dispute*, and this means that the scope of "*dispute*" is inside of that of "*want*." A scope of generalized quantifiers (such as "*the*") is represented with RSTR and BODY, which denote what is restricted by the quantifier and the scope of the quantifier, respectively. In the example, "*the*" restricts "*fact*," and the scope of "*the fact*" is the one represented by *h5*. It should be noted that *h5* does not appear in the other relations. This is because MRS can represent underspecified scope information. That is, the scope of "*the fact*" is not yet determined, and a fully scoped semantics is obtained by equating scope variables. For example, we can express that the scope of "*the fact*" is "*I want to dispute*" by equating *h5*, *h6*, and *h7*.

The research in this thesis adopted predicate argument structures because they were simpler and included sufficient information useful for practical applications. Since the only difference from MRS is scope information, the methods proposed in this thesis are essentially applicable to syntactic analyzers with MRS. Additionally, MRS can be constructed from word dependencies included in predicate argument structures (Spreyer and Frank, 2005).

2.3 Parsing Algorithms

Assuming that a grammar includes only binary and unary construction rules, we can apply a simple CKY-style algorithm (Cocke and Schwartz, 1970; Kasami, 1965; Younger, 1967) for parsing with lexicalized grammars. The CKY algorithm is a dynamic programming algorithm that fills each cell in *a chart*, as is shown in Figure 2.21, in a bottom-up manner. Each cell stores lexical/phrasal signs that cover a part of a sentence. The bottom cells store lexical signs, each of which corresponds to a word in a sentence. When computing phrasal signs to be stored in the upper part of the chart, binary construction rules are applied to lexical/phrasal signs of adjacent daughter phrases. For example, when we construct signs for "*a girl with a telescope*," two signs are fetched from cells that span adjacent phrases, i.e., "*a girl*" and "*with a telescope*," or "*a girl with*" and "*a telescope*." Binary

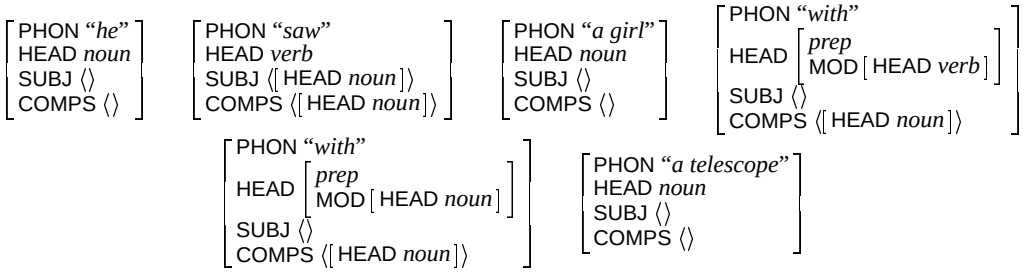


Figure 2.22: Lexical entries for "*he*", "*saw*", "*a girl*", "*with*", and "*a telescope*"

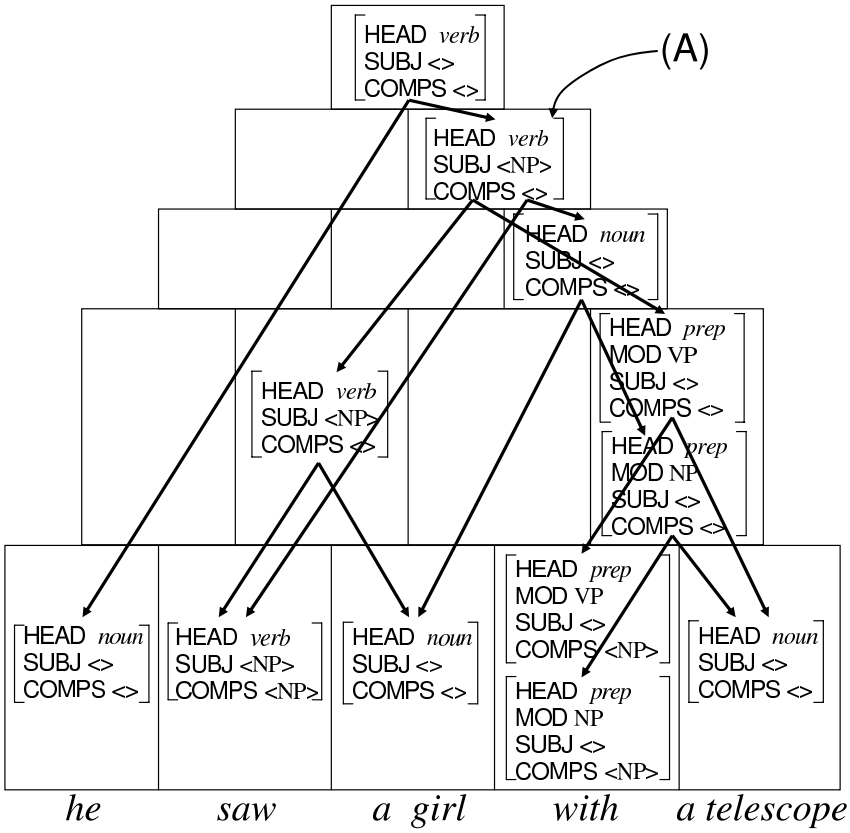


Figure 2.23: Chart for parsing "*He saw a girl with a telescope*"

construction rules are applied to every combination of daughter signs, and signs of the outputs of successful applications are stored in the cell of "*a girl with a telescope*."

Let us see the process of parsing a sentence "*He saw a girl with a telescope*" with a simple HPSG grammar. Figure 2.22 shows lexical entries required for parsing this sentence. For simplicity, we assign lexical entries directly to "*a girl*" and "*a telescope*", although they are respectively parsed as specifier-head constructions.

First, we fill the bottom cells with lexical entries. In Figure 2.23, lexical entries given in Figure 2.22 are put in the cells corresponding to each word. Next, the second row is filled. We choose one of the schemas and two daughters to be combined. Suppose that we chose lexical entries for "*saw*" and "*a girl*", and Head-Complement Schema. The schema is applied to the daughters, i.e., to

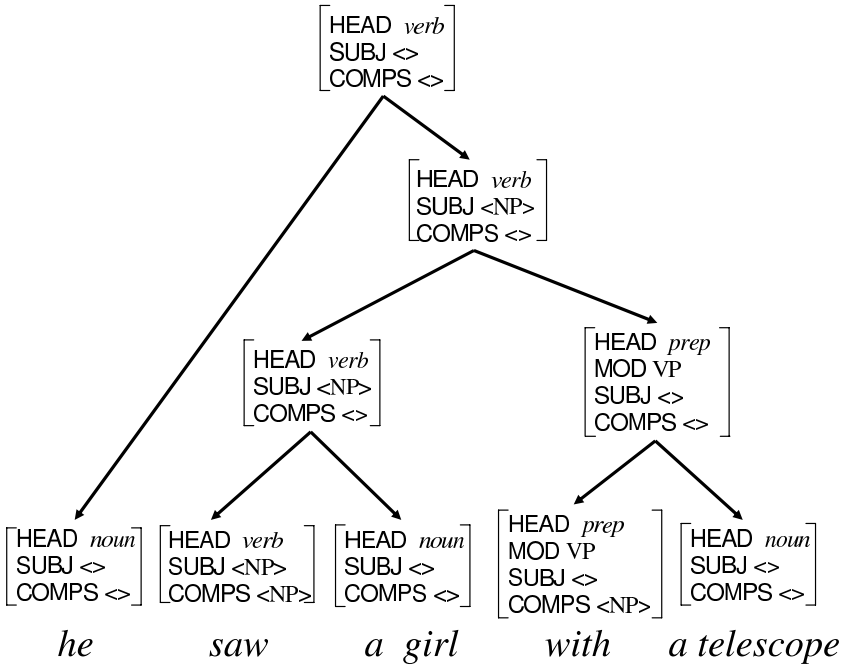


Figure 2.24: Result of parsing "He saw a girl with a telescope"

unify each daughter to the specified part of the schema. In this example, the lexical entries for "*saw*" and "*a girl*" are respectively unified with the left and the right daughters of the schema. The phrasal sign of the mother is obtained if the unification succeeds, and it is put in the chart. In the same way, each cell in the chart is filled in a bottom-up manner by combining lexical/phrasal signs. When we have got the sign of the whole sentence, parsing is finished. One of the resulting structures of "*He saw a girl with a telescope*" is shown in Figure 2.24. When we choose the alternative daughters of "*saw a girl with a telescope*", we can obtain an alternative reading.

When a new sign is put into the cell, we check whether an *equivalent* sign is already stored in the same cell, and if so, we choose only one of the signs for the cell. This mechanism is called *factoring*. Signs are equivalent when they cause no difference in combination with other signs. This means that we may ignore internal differences of the signs and only one of them may be targeted in following analyses. In particular, differences of structures of daughters may be ignored because schemas and principles do not mention them. For example, when we combine "*saw*" with "*a girl with a telescope*", the resulting sign is equivalent with the phrasal sign of the combination of "*saw a girl*" and "*with a telescope*", when we ignore daughter structures. In this case, we put only one sign in the cell (A in the figure). This mechanism is necessary for packing ambiguities in a local tree, and avoids explosion of parse trees. This is a key to polynomial-time parsability in CFG parsing, although the worst-case complexity of HPSG parsing is still exponential.

Figure 2.25 shows a pseudo code of a CKY-style algorithm of parsing with a lexicalized grammar. At beginning, lexical entries are put in the bottom cell of the chart. Next, the code includes three loops (three "**for**"s in the figure) to traverse the chart in a bottom-up fashion. In the figure, d represents the depth of the cell; the depth of the second rows is 1, and that of the top cell is $n-1$. The index i represents the left-most word that the cell covers; therefore, the span that the cell covers is represented as $\langle i, i+d \rangle$. In the inner-most code, a schema and two daughters are chosen, and the schema is applied. If it succeeds, the existence of an equivalent edge is checked, and the newly created edge is put into

```
G = ⟨L, R⟩:  a lexicalized grammar
w = ⟨w_1, ..., w_n⟩:  an input sentence
C_{i,j}:  constituents that span the i-th word to the j-th word
P_γ:  a set of pointers that γ has as daughters

# Set lexical entries
for i = 1 to n
  C_{i,i} ← {γ | ⟨w_i, γ⟩ ∈ L}
end_for
# Fill each cell in the chart in a bottom-up way
for d = 1 to n − 1
  for i = 1 to n − d
    C_{i,i+d} ← ∅
    for k = i + 1 to i + d − 1
      foreach r ∈ R, γ_1 ∈ C_{i,k}, and γ_2 ∈ C_{k,i+d}
        if γ = r(γ_1, γ_2) succeeds
          # Check whether an equivalent sign exists
          if γ ∈ C_{i,i+d}
            P_γ ← P_γ ∪ {⟨γ_1, γ_2⟩}
            C_{i,i+d} ← {γ}
          else
            C_{i,i+d} ← C_{i,i+d} ∪ {γ}
          end_if
        end_if
      end_foreach
    end_for
  end_for
end_for
```

Figure 2.25: CKY-style algorithm for lexicalized grammars

the cell.

As seen in Figure 2.25, the inner-most operation in parsing is a rule application, which requires unification of typed feature structures. Since unification is more expensive than application of CFG rules, parsing with lexicalized grammars is much heavier than CFG parsing. Hence, various techniques for improving parsing efficiency have been explored.

The most fundamental method is to accelerate the efficiency of unification of typed feature structures. Abstract machine architectures for efficient feature structure unification have been proposed (Carpenter and Qu, 1995; Makino et al., 2002; Wintner and Francez, 1999), following the idea of Prolog (Aït-Kaci, 1991; van Roy, 1990). That is, when feature structures are hard-coded in a program, the code for the unification of hard-coded feature structures can be optimized. Hard-coded feature structures are first converted into abstract machine instructions, and off-line optimizations are performed for the generated instructions. This algorithm was implemented in several systems including ALE (Carpenter and Penn, 1994), ProFIT (Erbach, 1995), PET (Callmeier, 2000), and LiLFeS (Makino et al., 1997, 1998; Miyao et al., 2000; Nishida et al., 1999). The program implemented in

this dissertation was implemented in LiLFeS.

Since another source of the inefficiency of parsing is ambiguity, ambiguity packing and efficient handling of disjunction have also been extensively studied (Blache, 1997, 1998; Carter, 1990; Dörre and Eisele, 1990; Eisele and Dörre, 1988; Flickinger, 2002; Griffith, 1995, 1996; Hasida, 1986; Kasper, 1987; Konig, 1992; Maxwell III and Kaplan, 1995; Miyao, 1999; Miyao et al., 1998; Nakano, 1991; Ramsay, 1990). In CFG parsing, ambiguities of local parse trees cause the exponential explosion of entire trees. Parsing algorithms such as CKY (Cocke and Schwartz, 1970; Kasami, 1965; Younger, 1967), Earley's algorithm (Earley, 1970), and chart parsing (Kay, 1986), avoid exponential explosion by packing ambiguities in local trees with the factoring method as mentioned above. However, this approach is not applicable to disjunctions internal in feature structures. Feature structures have reentrant structures as described above, and reentrancy complicates the necessary and sufficient conditions to express disjunctions in a packed structure. Previous studies listed above proposed packed representations of disjunctive feature structures, and efficient algorithms of unifying packed feature structures. Since none of them are employed in this thesis, see references listed above for details.

Following the observation that unification of feature structures during parsing rarely succeeds, several methods were proposed to detect unification failures as quickly as possible (Kiefer et al., 1999; Malouf et al., 2000). Sources of unification failures are limited to a small number of feature paths. For example, in the above parsing example, unification checks the consistency of selecting features such as SUBJ and COMPS. Hence, we should check consistency of these features first. The quick check algorithm first checks consistency of prespecified paths, and only when it succeeds, unification of feature structures is attempted. This means that the method filters out unnecessary unification quickly. This algorithm was implemented in our system (Ninomiya et al., 2005).

Another approach to filtering out unnecessary unification is a method of CFG filtering (Goldstein, 1988; Kiefer and Krieger, 2000; Torisawa et al., 2000; Torisawa and Tsujii, 1996). The idea is that we first compile a grammar into a CFG that can produce all of the parse trees that the original grammar generates. A compiled CFG may overgenerate parse trees, and the validness of parse trees generated by the CFG is verified in the second phase. Previous studies reported significant improvements of parsing efficiency (Torisawa et al., 2000). With a similar motivation, the compilation of HPSG to TAG was also proposed (Kasper et al., 1995). A theoretical bound of TAG parsing is $O(n^6)$ where the length of a sentence is n, while that of HPSG parsing is $O(2^n)$. If an HPSG grammar is compiled to a TAG grammar, we can expect an improvement in efficiency. However, other work (Yoshinaga et al., 2003) demonstrated that HPSG parsing was empirically faster than TAG parsing.

A recently developed technique is to exploit outputs of shallow analyzers for restricting the search space of parsing (Crysmann et al., 2002; Daum et al., 2003; Frank, 2004; Frank et al., 2003a). Shallow parsers, including named entity recognizers, chunk parsers, and CFG parsers, are generally more efficient than deep parsers including HPSG parsers. Hence, we first analyze sentences with shallow analyzers, and their outputs are then converted to constraints for deep parsers. For example, when we have a noun phrase chunker, it provides noun phrase boundaries to a deep parser. The deep parser does not need to compute constituents that violate given boundaries, and search space is reduced. Although the methods exploit shallow parsers, they are essentially different from CFG filtering. They exploit existing shallow analyzers that give outputs disambiguated by their own methods. That is, shallow analyzers do not have any theoretical connection to deep parsers. We should therefore regard the methods as practically oriented techniques. On the contrary, CFG parsers used in the methods of CFG filtering were automatically compiled from original lexicalized grammars. Since the CFG parsers are theoretically assured to produce all parse candidates output by the deep parser, they are theoretically sound.

Recent advances of probabilistic models of lexicalized grammars opened up new possibilities

for accelerating parsing efficiency. In probabilistic CFG parsing, various methods have been proposed for efficient searching guided by probabilities; examples include the beam search, the best-first search, and the A* search (Caraballo and Charniak, 1998; Charniak, 2000; Chitrao and Grishman, 1990; Collins, 1999; Goodman, 1997; Klein and Manning, 2003; Ratnaparkhi, 1999; Roark, 2001; Tsuruoka et al., 2004). Some of the algorithms were successfully applied to probabilistic HPSG parsing (Malouf and van Noord, 2004; Ninomiya et al., 2005).

2.4 Maximum Entropy Model

Following the recent advances in efficiency of parsing with lexicalized grammars, construction of disambiguation models has been a central topic for the practical application of lexicalized grammars. Because of the success of probabilistic CFG parsers (Charniak, 1997, 2000; Collins, 1997, 2003), probabilistic modeling seemed to be a promising solution to this goal. However, this was not that simple. One problem was that construction of training data, i.e., treebanks, was very expensive. While construction of CFG-style treebanks was expensive, treebanks of lexicalized grammars would require much more human effort because lexicalized grammars involved detailed description of syntactic constraints represented by complex data structures such as typed feature structures. We will tackle this difficulty through corpus-oriented development of lexicalized grammars in Chapter3.

Another difficulty is that complex data structures in lexicalized grammars prevent us from applying common methods of probabilistic modeling of CFG parsers. State-of-the-art CFG parsers assume that each application of a CFG rule (construction rule) is probabilistically independent of other rule applications, and regard a rule application as an atomic probabilistic event. Probabilities of atomic events are then estimated by maximum likelihood estimation, i.e., relative frequencies of events in training data. Owing to the independence assumption, the probability of the whole parse tree was defined as the product of probabilities of rule applications. However, it is difficult to divide complex data structures into independent probabilistic events. For feature structure grammars, Abney (1997) demonstrated that estimation by a relative frequency method did not maximize the likelihood of a training data because rule applications in feature structure grammars were not independent. This difficulty will be addressed in detail in Chapter 4.

The key of the solution in Chapter 4 is *maximum entropy models*, which are also known as *log-linear models* (Berger et al., 1996). This is because maximum entropy models provide methods of probabilistic modeling without independence assumption. In the following, we introduce the formulation of maximum entropy models, and the current achievements of this model.

2.4.1 Overview

Maximum entropy models are probabilistic models that have widely been accepted for various tasks in natural language processing because they achieve high accuracy. Examples include:

Statistical language model for English (Chelba et al., 1997; Chen and Rosenfeld, 1999b; Rosenfeld, 1997)

Part-of-speech tagging for Penn Treebank (Kazama et al., 2001; Ratnaparkhi, 1996), and for biomedical paper abstracts (Tsuruoka et al., 2005)

Named entity recognition for technical terms in biomedical papers (Borthwick, 1999; Kazama, 2004; Kazama and Tsujii, 2003, 2005)

PP attachment in English (Ratnaparkhi et al., 1994)

sentence	He	gave	books	to	his	sister .
part-of-speech	PRP	VBD	?			

feature	*target* (t_k)	*current word* (w_k)	*previous tag* (t_{k-1})
f_1	NNS	books	–
f_2	VBZ	books	–
f_3	NNS	books	VBD
f_4	VBZ	books	VBD
f_5	NNS	gave	VBD
f_6	VBZ	gave	VBD
f_7	NNS	*ends_with_"s"*	–
f_8	VBZ	*ends_with_"s"*	–

Figure 2.26: Maximum entropy bigram model for part-of-speech tagging

Subcategorization acquisition (Utsuro et al., 1997)

Shallow parsing for an English parser (Charniak and Johnson, 2005; Ratnaparkhi, 1997, 1999), for partial parsing (Skut and Brants, 1998), and for Japanese dependency analysis (Kanayama et al., 2000; Uchimoto et al., 2000, 1999)

Deep parsing for feature structure grammars in general (Abney, 1997), for LFG (Johnson et al., 1999; Kaplan et al., 2004; Riezler et al., 2002, 2000; Riezler and Vasserman, 2004), for CCG (Clark and Curran, 2003, 2004b), and for HPSG (Malouf and van Noord, 2004; Osborne, 2000; Toutanova and Manning, 2002)

Statistical machine translation for French-to-English translation (Berger et al., 1996), and for word alignment (Liu et al., 2005)

An advantage of maximum entropy models is that probabilistic events are represented with a bundle of *features*[7]. Probabilistic events in natural language processing, such as the tagging of part-of-speech to words and the assignment of nonterminal symbols to phrases, are not atomic but a combination of various characteristics. For example, when we assign a part-of-speech to "*books*", the correct part-of-speech (verb or noun) depends on the contexts in which the word occurs; if the previous word is a verb or a determiner, the word is likely to be a noun, but if the previous word is a pronoun (e.g. "*he*"), it should be a verb. The correct tag is determined according to various contexts including the surrounding words and part-of-speech tags. In maximum entropy models, contexts of events are formulated by features, and a probability of an event is computed by weights assigned to features.

Figure 2.26 illustrates an example of a simple bigram model for part-of-speech tagging[8]. Given a sentence as a sequence of words, i.e., $\mathbf{w} = \langle w_1, \ldots, w_n \rangle$, the task is to find a sequence of part-of-speech tags, i.e., $\mathbf{t} = \langle t_1, \ldots, t_n \rangle$. A bigram model defines conditional probability $p(\mathbf{t}|\mathbf{w})$ as the

[7] *Features* in maximum entropy models are a different concept from *features* in typed feature structures.

[8] Part-of-speech tags are of Penn Treebank (Marcus et al., 1994): NNS is for plural nouns, VBZ is for singular present verbs, and VBD is for past-tense verbs.

product of bigram probabilities $p(t_k|w_k, t_{k-1})$ as follows.

$$\begin{aligned} p(\mathbf{t}|\mathbf{w}) &= \prod_k p(t_k|w_1, \ldots, w_k, t_1, \ldots, t_{k-1}) \\ &= \prod_k p(t_k|w_k, t_{k-1}). \qquad \text{(by bigram assumption)} \end{aligned}$$

As in conventional tagging models, a probabilistic event is an assignment of a tag to a word, given a current word and a previous tag as a context. Our task is therefore to estimate $p(t_k|w_k, t_{k-1})$.

In maximum entropy models, a characteristic of an event is mapped into a scalar value by *a feature function* (or *a feature* for short). In Figure 2.26, $f_1, \ldots, f_8$ represent example feature functions in part-of-speech tagging. A feature function is an indicator function that returns 0 or 1 to represent the observation of a characteristic of an event. If the value of a feature is 0, the event does not have the characteristic corresponding to the feature. If it is 1, the event has the characteristic. For example, f_1, which represents whether the part of speech to be assigned is NNS and the current word is *books*, is defined as:

$$f_1(t_k, w_k, t_{k-1}) = \begin{cases} 1 & \text{if } t_k = \text{NNS and } w_k = \text{books} \\ 0 & \text{otherwise.} \end{cases}$$

Dashes in the figure mean "don't care"; for example, f_1 does not mention the value of the previous tag. When we assign NNS to "books" and its previous tag is VBD, feature functions in Figure2.26 have the following values.

$$\begin{aligned} f_1(\text{NNS,books,VBD}) &= 1 \\ f_2(\text{NNS,books,VBD}) &= 0 \\ f_3(\text{NNS,books,VBD}) &= 1 \\ f_4(\text{NNS,books,VBD}) &= 0 \\ f_5(\text{NNS,books,VBD}) &= 0 \\ f_6(\text{NNS,books,VBD}) &= 0 \\ f_7(\text{NNS,books,VBD}) &= 1 \\ f_8(\text{NNS,books,VBD}) &= 0 \end{aligned}$$

A feature function represents the existence of a certain characteristic in an event, and a set of *activated* features, i.e., $\{f_i|f_i = 1\}$, is considered to be an abstract representation of an event. In particular, maximum entropy models allow us to incorporate features which are not statistically independent. For example, f_1, f_3, and f_7 are not independent; when f_3 is 1, f_1 and f_7 are always 1 because the condition of f_3 strictly includes the conditions of f_1 and f_7. Features that have overlapping information may be incorporated into maximum entropy models. This allows for flexible modeling with various kinds of overlapping features. Model developers tweak feature functions to well describe characteristics of a target task.

A maximum entropy model is then defined as giving the probability $p(t_k|w_k, t_{k-1}; \boldsymbol{f}, E)$, where $\boldsymbol{f} = \{f_i\}$ is a set of feature functions and E is a training data.

2.4.2 Formulation

This section derives a set of formulae used in maximum entropy models, which is necessary for the descriptions in Chapter 4. In the following, a probability distribution is denoted as $p(y|x)$, where x

and y are probabilistic variables of a history and a target event, respectively. The values of x and y are taken from their domains $\mathcal{X}$ and $\mathcal{Y}$. The domain of pairs of history and target events is denoted as $\mathcal{E}$, i.e., $\mathcal{E} = \mathcal{X} \times \mathcal{Y}$. In part-of-speech tagging, $\mathcal{X}$ is a set of words and their contexts, and $\mathcal{Y}$ is a set of part-of-speech tags. In parsing, $\mathcal{X}$ is a set of sentences and $\mathcal{Y}$ is a set of parse trees. A set of target events may be restricted depending on a history event. For example, in part-of-speech tagging, target events should be restricted to part-of-speech tags that can be assigned to a word; a noun or a verb tag can be assigned to "books", but neither an adjective nor an auxiliary verb tag cannot. In parsing, target events will be parse candidates that are provided by grammar. In the following, we denote a set of all target events for history x as $Y(x)$.

A training data, $E \subset \mathcal{E}$, is given as a set of pairs of history and target events, that is,

$$E = \{\langle x_1, y_1 \rangle, \ldots, \langle x_{|E|}, y_{|E|} \rangle\}.$$

Evidently, each target event y_i paired with history event x_i must be included in $Y(x_i)$. Frequencies of history event x and a pair of target and history events $\langle x, y \rangle$ in the training data are denoted as $c(x)$ and $c(x, y)$, respectively. A relative frequency is then defined as follows:

$$\tilde{p}(x) = \frac{c(x)}{|E|},$$

$$\tilde{p}(x, y) = \frac{c(x, y)}{|E|}.$$

A feature function is a partial function from a target/history event into a real value:

$$f_i : \mathcal{E} \mapsto \mathbf{R}.$$

Although the example in Figure 2.26 included only indicator functions, i.e., binary-valued features, the range of feature functions may be a real value in general. Examples of real-valued features include *tf-idf* in text retrieval and log-probability given by an auxiliary probability model (Johnson and Riezler, 2000). In the following, we denote a set of feature functions as $\boldsymbol{f}$.

Given a set of feature functions $\boldsymbol{f} = \{f_i\}$, we collect the expectations of the values of feature functions from training data.

Definition 2.9 (Empirical expectation) *Given training data E, an empirical expectation of feature f_i is defined as follows:*

$$\tilde{\mu}_i = \sum_{x \in \mathcal{X}} \sum_{y \in Y(x)} \tilde{p}(x, y) f_i(x, y).$$

Maximum entropy models define the probability $p(y|x)$ so as to give expectations of feature values that are equal to the empirical expectations. An expectation given by a model is defined as follows.

Definition 2.10 (Model expectation) *A model expectation of feature f_i according to a probabilistic model $p(y|x)$ is defined as follows:*

$$\mu_i = \sum_{x \in \mathcal{X}} \tilde{p}(x) \sum_{y \in Y(x)} p(y|x) f_i(x, y).$$

Hence, from Definition 2.9 and 2.10, maximum entropy models offer the following constraints on the probabilistic model.

Definition 2.11 (Constraint equation) *For feature function $f_i \in \boldsymbol{f}$, constraint equation is defined as follows:*

$$\mu_i = \tilde{\mu}_i$$

$$\Longleftrightarrow \qquad \sum_{x \in \mathcal{X}} \tilde{p}(x) \sum_{y \in Y(x)} p(y|x) f_i(x,y) = \sum_{x \in \mathcal{X}} \sum_{y \in Y(x)} \tilde{p}(x,y) f_i(x,y).$$

An infinite number of probabilistic models may satisfy the above constraints if the number of feature functions is smaller than the degree of freedom. Among these, a maximum entropy approach selects the model that is most *uniform*. A measure of uniformity is the *conditional entropy*.

Definition 2.12 (Conditional entropy) *A conditional entropy of probabilistic model $p(y|x)$ is defined as follows:*

$$H(p) = -\sum_{x \in \mathcal{X}} \tilde{p}(x) \sum_{y \in Y(x)} p(y|x) \log p(y|x).$$

Theoretically, the model that maximizes the conditional entropy is equivalent to the probabilistic model that is closest to the uniform distribution in terms of the Kullback-Leibler distance (Berger et al., 1996).

A maximum entropy model is defined as the probabilistic model that maximizes the conditional entropy under the constraint equations. Hence, a maximum entropy model is the solution of the following constrained optimization problem.

Definition 2.13 (Maximum entropy model) *Maximum entropy model $p_M(y|x)$ in terms of training data E is defined as follows:*

$$p_M(y|x) = \operatorname*{argmax}_p \left\{ -\sum_{x \in \mathcal{X}} \tilde{p}(x) \sum_{y \in Y(x)} p(y|x) \log p(y|x) \right\},$$

subject to:

$$\forall f_i \in \boldsymbol{f} \qquad \sum_{x \in \mathcal{X}} \tilde{p}(x) \sum_{y \in Y(x)} p(y|x) f_i(x,y) = \sum_{x \in \mathcal{X}} \sum_{y \in Y(x)} \tilde{p}(x,y) f_i(x,y),$$

$$\forall x \in \mathcal{X} \qquad \sum_{y \in Y(x)} p(y|x) = 1.$$

The last constraint is necessary for ensuring p to be a probabilistic distribution.

The above problem can be transformed into a parametric form by the method of *Lagrange multipliers*. That is, by introducing a Lagrange multiplier for each constraint equation, the optimization problem is transformed into an unconstrained optimization. Conversion of the problem and the solution are described in detail in Appendix A.

In a parametric form, maximum entropy models are re-defined as follows.

Definition 2.14 (Maximum entropy model (parametric form)) *A maximum entropy model is defined as the solution of the following optimization problem.*

$$p_M(y|x;\boldsymbol{\lambda}) = \operatorname*{argmax}_p \left\{ -\sum_x \tilde{p}(x) \log Z(x) + \sum_i \lambda_i \tilde{\mu}_i \right\}, \tag{2.1}$$

where:

$$p(y|x;\boldsymbol{\lambda}) = \frac{1}{Z(x)} \exp\left(\sum_i \lambda_i f_i(x,y)\right), \tag{2.2}$$

$$Z(x;\boldsymbol{\lambda}) = \sum_{y \in Y(x)} \exp\left(\sum_i \lambda_i f_i(x,y)\right). \tag{2.3}$$

In fact, the optimization function in Equation 2.1 is a log-likelihood of training data.

Definition 2.15 (Log-likelihood) *Log-likelihood of probabilistic model $p(y|x)$ according to training data E is defined as follows:*

$$\begin{aligned} L(p,E) &= \log \prod_{\langle x,y\rangle \in E} p(y|x)^{\tilde{p}(x,y)} \\ &= -\sum_{x \in \mathcal{X}} \tilde{p}(x) \log Z(x) + \sum_i \lambda_i \tilde{\mu}_i. \end{aligned}$$

This means that maximum entropy models are the result of a maximum likelihood estimation when a family of probability distributions is given in the log-linear form of Equation2.2.

Note that the above model assumes nothing about feature functions. That is, it yields a maximum likelihood estimation even when features are not independent, which is unattainable when using relative frequency methods of traditional probabilistic models. This advantage allows for flexible modeling with various kinds of overlapping feature functions, as described in Figure2.26. As will be discussed in Chapter 4, it is indispensable for the probabilistic modeling of parsing with lexicalized grammars, because complex data structures, such as feature structures, are difficult to decompose into independent events (Abney, 1997).

The optimization problem in Definition 2.14 does not have a closed form solution. Numerical algorithms have been proposed for finding model parameters (i.e., $\boldsymbol{\lambda}$) that maximize the log-likelihood of the training data, i.e., $L(p,E) = L(\boldsymbol{\lambda})$. Improved Iterative Scaling (IIS) (Della Pietra et al., 1997) and Generalized Iterative Scaling (GIS) (Darroch and Ratcliff, 1972) are parameter estimation algorithms specialized for maximum entropy models. General-purpose gradient-based methods, such as Conjugate Gradient (CG) (Fletcher and Reeves, 1964) and limited-memory BFGS methods (Nocedal and Wright, 1999; Nocedal, 1980), are also applicable because the objective function and its gradient are computable. The above algorithms are explained in detail in ChapterB.

All of the algorithms mentioned above require the computation of model expectations (Definition 2.10) of feature values, i.e., μ_i, (or factored model expectations $\mu_{\langle i,f\#\rangle}$ in IIS) in each iteration. For example, gradient-based algorithms require optimization function $L(\boldsymbol{\lambda})$ and its gradient $\boldsymbol{g} = \nabla L(\boldsymbol{\lambda})$.

$$L(\boldsymbol{\lambda}) = -\sum_{x \in \mathcal{X}} \tilde{p}(x) \log Z(x) + \sum_i \lambda_i \tilde{\mu}_i,$$

$$\begin{aligned} \nabla L(\boldsymbol{\lambda}) &= \langle \frac{\partial L(\boldsymbol{\lambda})}{\partial \lambda_1}, \ldots, \frac{\partial L(\boldsymbol{\lambda})}{\partial \lambda_n} \rangle \\ &= \langle \tilde{\mu}_1 - \mu_1, \ldots, \tilde{\mu}_n - \mu_n \rangle. \end{aligned}$$

Computation of model expectations dominates the cost of overall computation because it requires traversal of all training data. Computational complexity of each iteration is $O(|\tilde{Y}||\tilde{F}||E|)$, where $|\tilde{Y}|$

and $|\tilde{F}|$ are the average numbers of targets and activated features for an event, respectively, and $|E|$ is the number of events. The algorithm is sufficiently efficient for various NLP applications, such as part-of-speech tagging, because $|\tilde{Y}|$ and $|\tilde{F}|$ are usually small (more or less than 100), while $|E|$ is large, as in other statistical models. However, this method is intractable in some cases, which will be the main problem discussed in Chapter 4.

2.4.3 Extensions

Several extensions have been proposed for maximum entropy models. A substantial one is to tackle a data sparseness problem. Generally, maximum entropy models are considered to be robust against data sparseness because feature functions allow for the incorporation of overlapping information into the model and thus we expect smoothing effects by introducing generalized features together with specific features. For example, in Figure 2.26, f_7 and f_8, which check whether the current word ends with "s" or not, represent less specific contexts than the word itself. Such features are expected to avoid overfitting to the data.

To achieve higher accuracy, model developers must incorporate many feature functions into a model in order to represent various fine-grained contexts. With many feature functions and less training data, even maximum entropy models may overfit to training data. To be concrete, constraint equations in Definition 2.13 may overfit to training data because empirical expectations $\tilde{\mu}_i$ are unreliable if the frequency of f_i is too small. For example, Johnson et al. (1999) pointed out that if a model involved a feature that is always active with the correct target, the optimal weight of the feature would be infinite. This is because increase of a weight of the feature always increases the log-likelihood. However, this obviously means the model is overfitting to training data.

The simplest solution to this problem is to remove infrequent features from the model (so-called *cut-off* methods). With a threshold value, c, features whose frequency is less than c are removed from the model. The method was widely used in the early stage of maximum entropy modeling.

A more sophisticated solution to the problem is maximum a posteriori estimation (MAP estimation) with a Gaussian prior (Chen and Rosenfeld, 1999a). This is a kind of *regularization*, which penalizes features with large weights. The Gaussian MAP estimation penalizes features according to a Gaussian distribution.

We assume the following probabilistic distribution on model parameters as a prior distribution.

Definition 2.16 (Gaussian prior) *A Gaussian prior distribution on model parameter λ_i is defined as follows:*

$$p_G(\lambda_i; \sigma_i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\lambda_i^2}{2\sigma_i^2}\right),$$

where σ_i is a meta parameter of the distribution.

In this formula, σ_i is a standard deviation, and model parameter λ_i is assumed to distribute in terms of the Gaussian distribution. Intuitively, an optimal model parameter becomes smaller when σ_i is smaller.

This distribution is assumed to be a prior distribution of the likelihood. That is, a log-likelihood is re-defined as the logarithm of the product of the Gaussian prior and the original likelihood.

Definition 2.17 (Log-likelihood with a Gaussian prior) *A log-likelihood with a Gaussian prior on*

model parameters is defined as follows:

$$\begin{aligned} L_G(\boldsymbol{\lambda};\boldsymbol{\sigma}) &= L(\boldsymbol{\lambda}) + \log \prod_i p_G(\lambda_i;\sigma_i) \\ &= -\sum_x \tilde{p}(x)\log Z(x) + \sum_i \lambda_i \tilde{\mu}_i + \sum_i \left(-\log\sqrt{2\pi}\sigma_i - \frac{\lambda_i^2}{2\sigma_i^2} \right). \end{aligned}$$

By ignoring the constant term, the optimization function is a regularized log-likelihood:

$$L'_G(\boldsymbol{\lambda};\boldsymbol{\sigma}) = -\sum_x \tilde{p}(x)\log Z(x) + \sum_i \lambda_i \tilde{\mu}_i - \sum_i \frac{\lambda_i^2}{2\sigma_i^2}.$$

Hence, a maximum entropy model is redefined as follows:

Definition 2.18 (Maximum entropy model with a Gaussian prior) *A maximum entropy model with a Gaussian prior is defined as the solution of the following optimization problem.*

$$p_G(y|x;\boldsymbol{\lambda},\boldsymbol{\sigma}) = \operatorname*{argmax}_p \left\{ -\sum_x \tilde{p}(x)\log Z(x) + \sum_i \lambda_i \tilde{\mu}_i - \sum_i \frac{\lambda_i^2}{2\sigma_i^2} \right\},$$

where:

$$p(y|x;\boldsymbol{\lambda}) = \frac{1}{Z(x)} \exp\left(\sum_i \lambda_i f_i(x,y) \right),$$

$$Z(x;\boldsymbol{\lambda}) = \sum_{y \in Y(x)} \exp\left(\sum_i \lambda_i f_i(x,y) \right).$$

Chen and Rosenfeld (1999a) presented the IIS algorithm for maximum entropy models with Gaussian priors. The algorithm is almost the same as Figure B.1. Only the equation to obtain $\Delta\lambda_i$ is rewritten as follows:

$$\sum_{x,y} \tilde{p}(x,y) f_i(x,y) = \sum_{x,y} \tilde{p}(x) p(y|x) f_i(x,y) \exp\left(\Delta\lambda_i f^{\#}(x,y) \right) - \frac{\lambda_i + \Delta\lambda_i}{\sigma_i^2}.$$

Gradient-based methods can also be applied to the optimization of the log-likelihood with the Gaussian prior. The derivatives of the optimization function become as follows.

$$\frac{\partial L_G(\boldsymbol{\lambda};\boldsymbol{\sigma})}{\partial \lambda_i} = \tilde{\mu}_i - \mu_i - \frac{\lambda_i}{\sigma_i^2}.$$

These results show that the parameter estimation is not much different from ordinary maximum entropy models. This extension does not significantly increase computational complexity of parameter estimation.

Another solution is to assume an exponential distribution as a prior distribution (Goodman, 2004; Kazama, 2004; Kazama and Tsujii, 2003, 2005). This model is considered to be a model in which constraints are defined as inequations (Kazama, 2004). Constraint equations in Definition 2.11 are redefined as follows.

Definition 2.19 (Constraint inequation) *For feature function $f_i \in \boldsymbol{f}$, a constraint function is defined as follows:*

$$\begin{aligned} & -A_i \le \mu_i - \tilde{\mu}_i \le B_i \\ \iff \qquad & -A_i \le \sum_{x \in \mathcal{X}} \tilde{p}(x) \sum_{y \in Y(x)} p(y|x) f_i(x,y) - \sum_{x \in \mathcal{X}} \sum_{y \in Y(x)} \tilde{p}(x,y) f_i(x,y) \le B_i, \end{aligned}$$

where $A_i \ge 0$ and $B_i \ge 0$.

With the constraints, maximum entropy models are defined as follows.

Definition 2.20 (Maximum entropy model with inequality constraints) *Maximum entropy model $p_I(y|x)$ with inequality constraints is defined as follows:*

$$p_I(y|x) = \operatorname*{argmax}_p \left\{ -\sum_{x \in \mathcal{X}} \tilde{p}(x) \sum_{y \in Y(x)} p(y|x) \log p(y|x) \right\},$$

subject to:

$$\forall f_i \in \boldsymbol{f} \qquad \sum_{x \in \mathcal{X}} \sum_{y \in Y(x)} \tilde{p}(x,y) f_i(x,y) - \sum_{x \in \mathcal{X}} \tilde{p}(x) \sum_{y \in Y(x)} p(y|x) f_i(x,y) - A_i \leq 0,$$

$$\forall f_i \in \boldsymbol{f} \qquad \sum_{x \in \mathcal{X}} \tilde{p}(x) \sum_{y \in Y(x)} p(y|x) f_i(x,y) - \sum_{x \in \mathcal{X}} \sum_{y \in Y(x)} \tilde{p}(x,y) f_i(x,y) - B_i \leq 0,$$

$$\forall A_i, B_i \qquad A_i \geq 0, \quad B_i \geq 0.$$

Since this problem requires an optimization with bounds on variables, the algorithms described in Section 2.4.2 for parameter estimation cannot be directly applied. Instead, the problem is solved with function optimizers for problems with lower/upper bounds on variables. The most well-known algorithm is BLMVM, which is an extension of the L-BFGS algorithm to handling optimization problems with bound constraints (Benson and Moré, 2001). Kazama (2004) demonstrated that this model achieved performance comparable to the Gaussian MAP method with fewer features.

Chapter 3

Corpus-Oriented Development of Lexicalized Grammars

This chapter discusses a new methodology for the development of lexicalized grammars. The development process is *corpus-oriented*, in the sense that the target of the development is an annotated corpus, i.e., a treebank, rather than a lexicon. A lexicon is obtained as a co-product of treebank development. The treebank may also be used for training statistical models for disambiguation. This method supports the inexpensive development of lexicalized grammars because existing treebank resources are reused and the maintenance of consistency of a treebank is easier than that of a lexicon. With this approach, we developed an HPSG parser for English using Penn Treebank as a source treebank. We implemented schemas and principles of HPSG following the definition of Pollard and Sag (1994), and developed empirical annotation rules to convert Penn Treebank into an HPSG treebank. Predefined schemas and principles worked for verifying the consistency of the HPSG treebank. Lexical entries of HPSG were then collected from the HPSG treebank. Details of grammar design and empirical annotation rules are described with examples of an HPSG English grammar and Penn Treebank. Evaluation of the grammar will be presented in Chapter 6.

3.1 Methodology

Linguistically motivated and computationally oriented grammar theories take the form of *lexicalized grammar formalisms*; examples include Lexicalized Tree Adjoining Grammar (LTAG) (Schabes et al., 1988), Lexical Functional Grammar (LFG) (Bresnan, 1982), Combinatory Categorial Grammar (CCG) (Steedman, 2000), and Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994; Sag and Wasow, 1999). In terms of linguistic analysis and efficiency, they have been successful in deep syntactic analysis of natural languages (Oepen et al., 2002a). However, lexicalized grammars have not generally been considered suitable for syntactic analysis within practical NLP systems. Although a few studies could apply a lexicalized grammar to the analysis of a real-world corpus (Riezler et al., 2002), these required various techniques for robust processing and considerable human effort that lasted for over a decade.

We suspect that the impracticality of lexicalized grammars was because of the lower interest in the following essential properties of syntactic analysis.

Scalability The syntactic analysis of unrestricted text requires a comprehensive lexicon. However, the development of lexicalized grammars has been limited to small domains, because they involve complex description of metaphysical linguistic constraints. The scaling-up of a lexical-

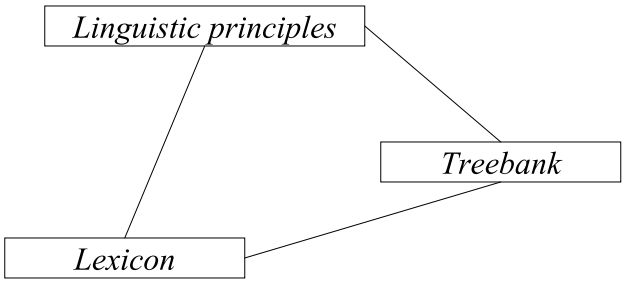


Figure 3.1: Grammar resources

ized grammar is not just a matter of increasing the size of a lexicon, but grammar developers are hampered by the maintenance of the consistency of complex constraints.

Modeling of preference Statistical preference is necessary for disambiguation, which is indispensable in practical applications. However, the main concern of the development of lexicalized grammars has been the acceptability of syntactic structures, and the modeling of statistical preference was unrespected.

The first problem requires a systematic procedure of the verification of the consistency of a lexicalized grammar. Traditionally, grammar developers verified grammar consistency by observing the results of the parsing of a test corpus. Grammar writers manually investigated the sources of inconsistency, and fixed the lexicon according to their linguistic intuition. However, this process often incorporated new inconsistencies into the lexicon; when a lexicon was adjusted to treat some words/constructions, the adjustment often negatively impacted on other words/constructions. The second problem requires statistical models of lexicalized grammars. Although several studies reported statistical disambiguation models, most are limited to small experiments because a training data, i.e., a treebank, was insufficient. For the accurate syntactic analysis in practical NLP applications, we require a sufficient size of a treebank based on the target grammar theory.

The development of a treebank can be a promising solution to the above problems. Historically, a lexicon and linguistic principles were considered sufficient to develop a system of syntactic analysis. Linguistic principles, such as ID schemas and principles in HPSG, are responsible for explaining general linguistic constraints, while a lexicon provides mappings from a word into its grammatical constraints. Theoretically, they are sufficient to explain syntactic structures of sentences supposed in the theory of a lexicalized grammar. However, as described above, a testbed of consistency maintenance and a training data for statistical modeling are necessary. A treebank can be a testbed for the verification of the consistency of a grammar. Through the development of a treebank, grammar writers can detect defects in a grammar, given concrete sentences that involve problematic constructions (Oepen et al., 2004). As long as the consistency of a sufficient treebank is maintained, the consistency of a grammar is assured to some extent. In addition, a treebank will be required anyway for the development of statistical models for disambiguation. Hence, we claim that a practical system of syntactic analysis requires three grammar resources: linguistic principles, a lexicon, and a treebank (Figure 3.1).

The new strategy outlined here is *corpus-oriented development* of lexicalized grammars. We first develop a treebank that conforms to a target grammar theory, i.e., HPSG in our case. A lexicon is then collected from the treebank. When we target the development of the three resources in conformity with a lexicalized grammar theory, we find that a lexicon is automatically obtained if we have a treebank and linguistic principles. For example, Figure 3.2 shows a derivation tree of HPSG, which

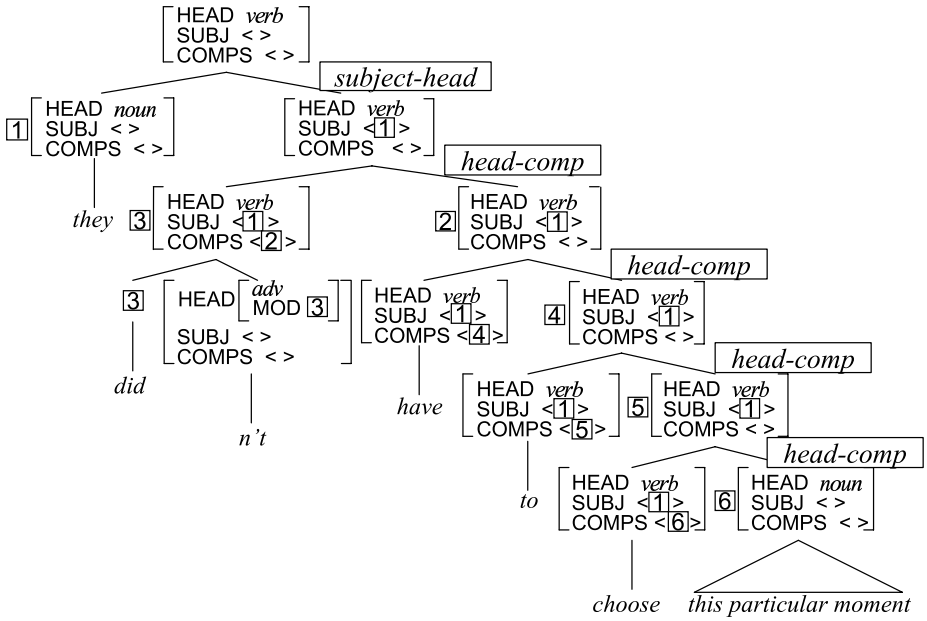


Figure 3.2: HPSG derivation tree

is a representation of a syntactic structure of HPSG. By simply collecting terminal nodes of the tree, we obtain lexical entries of the words in the sentence. This is because an HPSG treebank strictly includes the information of a lexicon, and linguistic principles determine the origin of the constraints expressed in a derivation tree.

Hence, we should explore the fastest way of establishing a treebank and linguistic principles. In this study, our target is an English grammar based on HPSG. Since we already have a large treebank of English, *Penn Treebank* (Marcus et al., 1994), we decided to exploit this resource for the development of an HPSG treebank. Our strategy is outlined as follows.

- We define a type hierarchy of HPSG signs, ID schemas, and principles that are regulated by the theory of HPSG.
- We develop empirical annotation rules to convert Penn Treebank into an HPSG treebank.

The design of a type hierarchy, ID schemas, and principles basically follows the definition by Pollard and Sag (1994). In addition to these, by investigating sentences in the Penn Treebank, we add several schemas to deal the constructions that appear in Penn Treebank though their treatments are not explicitly described in Pollard and Sag (1994) (details will be described in Section 3.2). Empirical annotation rules are pattern rules on the tree structure of Penn Treebank-style annotations. The structure of the trees is converted, and feature constraints are added to make the treebank conform to the theory of HPSG, i.e., to make the treebank satisfy linguistic principles. By applying empirical annotation rules to Penn Treebank, we obtain partially-specified derivation trees of HPSG. By applying linguistic principles to them, the consistency of the annotated constraints is automatically verified. Inconsistency of annotations and structures uncovered by linguistic principles are automatically detected, and they will require modification of empirical annotation rules and/or linguistic principles. If no inconsistency is found, we obtain fully specified derivation trees that conform to the linguistic principles. From the HPSG treebank, we collect lexical entries from terminal nodes of HPSG derivation trees. Since derivation trees are assured to conform to the linguistic principles, so do lexical entries.

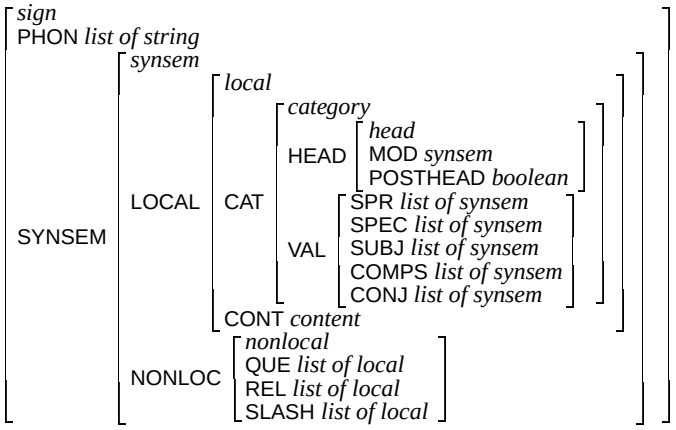


Figure 3.3: HPSG sign

The remainder of this chapter is devoted to reporting the details of our treebank and grammar. In Section 3.2, we introduce the design of a sign and the definition of ID schemas in HPSG. In Section 3.3, we describe the method of converting Penn Treebank into an HPSG treebank with empirical annotation rules. In Section 3.4, an example of lexicon extraction is presented. In Section 3.5, our methodology is compared in detail with two existing methodologies, i.e., manual development of a grammar and grammar learning from a treebank.

3.2 Grammar Design

This section introduces the design of a typed feature structure of HPSG signs and the definition of ID schemas and principles in our HPSG grammar. HPSG (Pollard and Sag, 1994) is a syntactic theory based on lexicalized grammar formalism. *A sign* is a linguistic entity that represents syntactic/semantic constraints of words/phrases. *Lexical entries* are signs assigned to words and express word-specific characteristics such as syntactic categories and subcategorization frames. *Schemas* define general construction rules for combining lexical/phrasal signs. A schema is applied to lexical/phrasal signs to make a larger constituent. A history of schema applications constitutes *an HPSG derivation tree*, which can be represented as a tree. *Principles*, such as Head Feature Principle, regulate general constraints of lexical/phrasal signs. Signs, principles, and schemas are represented with typed feature structures, and constraints represented by feature structures are checked by *unification* (Carpenter, 1992). Refer to Chapter 2 for the details of HPSG.

Figure 3.3 provides the definition of a sign in our grammar. The sign in our grammar basically follows the definition of Pollard and Sag (1994). PHON is a feature for phonological information of a word (a surface string of a word in our grammar). HEAD expresses the characteristics percolated from the head word of a constituent, such as syntactic categories. MOD, SPR, SPEC[1], SUBJ, and COMPS represent selectional constraints of a modifiee, a specifier, a specifiee, left-arguments, and right-arguments. QUE, REL, and SLASH are used to explain long-distance dependencies. CONT represents the semantics of a constituent, and in this study it expresses predicate-argument dependencies.

Several features are additionally defined in our grammar. POSTHEAD, which is cited from LinGO English Resource Grammar (ERG) (Copestake and Flickinger, 2000; Flickinger, 2002), determines whether a phrase is a post-head modifier. CONJ is specially defined in our grammar to explain

[1]We defined SPEC in VAL, while Pollard and Sag (1994) defined it in HEAD.

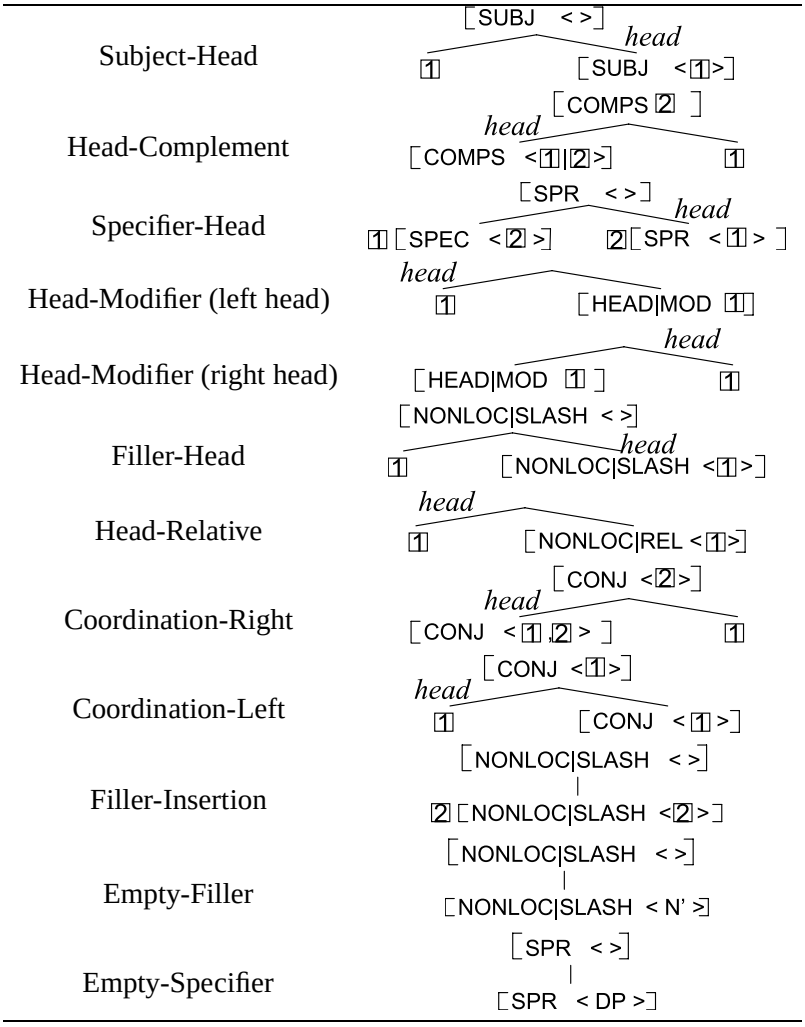


Figure 3.4: HPSG schemas

coordination constructions. Coordination conjunctions have non-empty CONJ, which expresses the constraints of left/right conjuncts.

Features are additionally defined for each syntactic category to represent fine-grained constraints although they have been omitted from the figure. As described in Pollard and Sag (1994) and in Chapter 2, VFORM, AUX, and INV are defined in HEAD of verbs. Our grammar also includes TENSE to represent tense of verbs. CASE is defined for nouns to represent cases of nouns. Prepositions include PFORM to represent types of prepositions. AGR represents agreement constraints between verbs and their subjects.

Figure 3.4 is a list of schemas defined in our grammar[2]. Following Pollard and Sag (1994), this study defines the following schemas: Subject-Head, Head-Complement, Specifier-Head, Head-Modifier (left-head and right-head), and Filler-Head Schema. In addition to these, the following schemas are defined as dealing with the constructions that appear in Penn Treebank though their treatments are not explicitly described in Pollard and Sag (1994). The *Head-Relative Schema* is defined for constructions in which a relative clause modifies its antecedent. It should be noted that

[2]For simplicity, we omit features irrelevant to the explanation in the figures in this chapter.

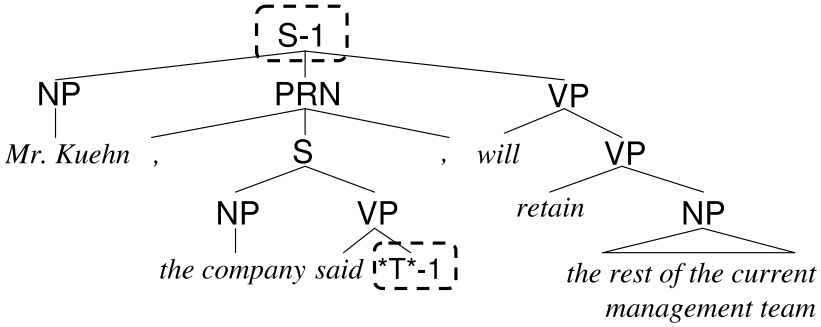


Figure 3.5: Sentence with a filler-insertion construction

Pollard and Sag (1994) introduced a null relativizer, a phonologically empty constituent, to explain relative clause constructions, and Sag (1997) proposed another explanation by exploiting default unification. Our approach is similar to the latter explanation, although we defined a new schema instead of introducing default unification. The *Coordination-Right/Left Schemas* are for coordination constructions. A coordination conjunction first takes an argument to its right by the Coordination-Right Schema, and then takes a left conjunct by the Coordination-Left Schema. The *Filler-Insertion Schema* is defined for the construction in which an inserted clause introduces a slash, which is filled by the entire sentence. For example, Figure 3.5 shows a parse tree with a filler-insertion construction in Penn Treebank. In the sentence "*Mr. Kuehn, the company said, will retain the rest of the current management team*," the complement of the inserted clause is coindexed with the entire sentence. The *Empty-Filler Schema* is a unary schema and necessary for relative clause constructions without relativizers: e.g., "*a book he bought*." The *Empty-Specifier Schema* is another unary schema to attach an empty specifier to nouns without specifiers.

Features not mentioned in the figure are percolated from daughters to their mother as is specified in *principles*. For example, the *Head Feature Principle* regulates the structure-sharing of HEAD features of a daughter and its mother. The *Valence Principle* lets VAL features be percolated to the mother by default. We implemented HPSG principles explained in Section 2.2.3.

In addition to these, our grammar includes lexical rules to explain inflectional and derivational lexical transformations (Nakanishi et al., 2004a,b). We wrote lexical rules, and conditions to determine lexical signs of base forms of verbs, nouns, and prepositions. Terminal nodes of HPSG derivation trees are converted to signs of base forms, when lexical rules can be applied inversely to them and obtained signs satisfy the conditions of base form signs. That is, a terminal node is given as an output of a lexical rule, and the input of the lexical rule is computed and stored in the lexicon. In run-time, lexical rules are applied to base form signs to derive lexical entries for inflected words. For details, see Nakanishi et al. (2004a,b).

3.3 Development of an HPSG Treebank

The aim of our grammar development is to make an HPSG treebank in conformity with the linguistic principles introduced in Section 3.2. That is, grammar writers develop a collection of sentences annotated with HPSG derivation trees. For example, Figure 3.2 shows the goal of annotation of the sentence "*They didn't have to choose this particular moment*," where only HEAD, MOD, SUBJ, and COMPS are shown in the figure. If a certain amount of sentences are annotated with HPSG derivation trees, lexical entries are collected from terminal nodes of the given derivation trees.

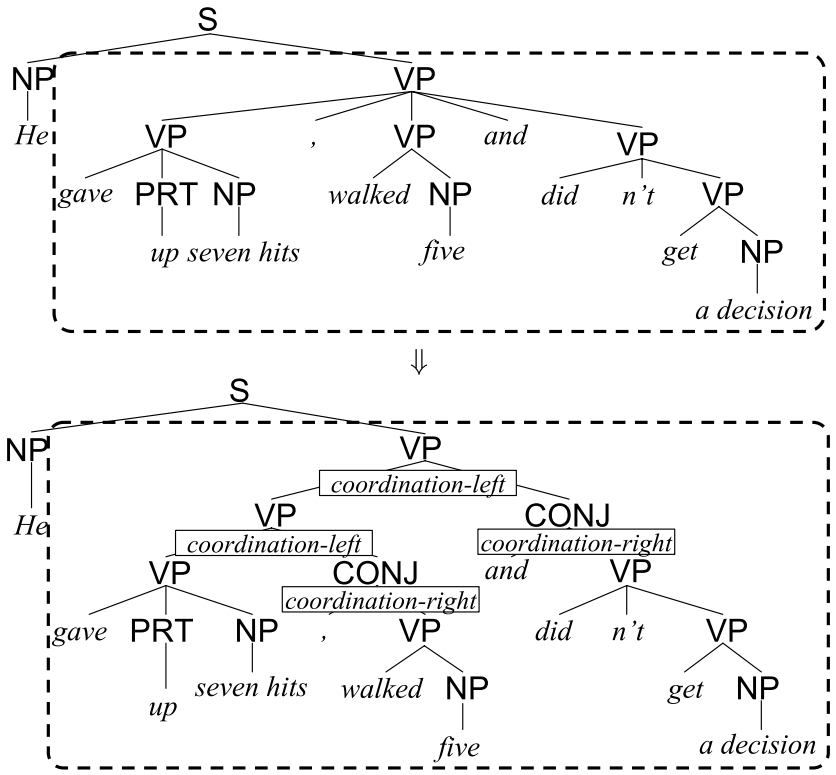


Figure 3.6: Annotation of coordination constructions

The method of making derivation trees is the main concern in this section. Obviously, manual construction from scratch is extremely expensive. In this study, an HPSG treebank was developed by converting Penn Treebank using *empirical annotation rules*, which are pattern rules on partial structures of CFG-style trees. First, empirical annotation rules are sequentially applied to each partial tree in parse trees in the Penn Treebank. Penn Treebank-style trees are then converted to *partially-specified derivation trees*, which include partial constraints of HPSG derivation trees. Next, linguistic principles are applied to partially-specified derivation trees. Partial constraints are resolved, and if no inconsistency is found, we obtain fully specified derivation trees.

In the following, we describe our current design of empirical annotation rules. The target of annotation and a simple example are explained for each group of annotation rules. The annotation rules described in this section are extensions of our previous study on grammar development (Miyao et al., 2003a, 2005).

Errors of non-/pre-terminal symbols Errors or inconsistencies in non-/pre-terminal symbols often cause failures in grammar acquisition (Hockenmaier and Steedman, 2002a). In HPSG, since the Head-Feature Principle demands that the HEAD features of a mother and its head daughter must be shared, errors of symbols often cause the violation of the principle. For example, a unary daughter of PRT (particle) is often assigned preterminal RB (adverb) (e.g. (PRT out/RB)). Since this causes the Head-Feature Principle to be violated, the preterminal of the daughter is fixed to RP (particle). We enumerated $\langle$mother symbol, daughter symbol, fixed symbol$\rangle$ tuples, e.g., $\langle$PRT, RB, RP$\rangle$, and if the mother and the daughter of a partial tree match the first two elements of a tuple, the symbol of the daughter is rewritten to the fixed symbol.

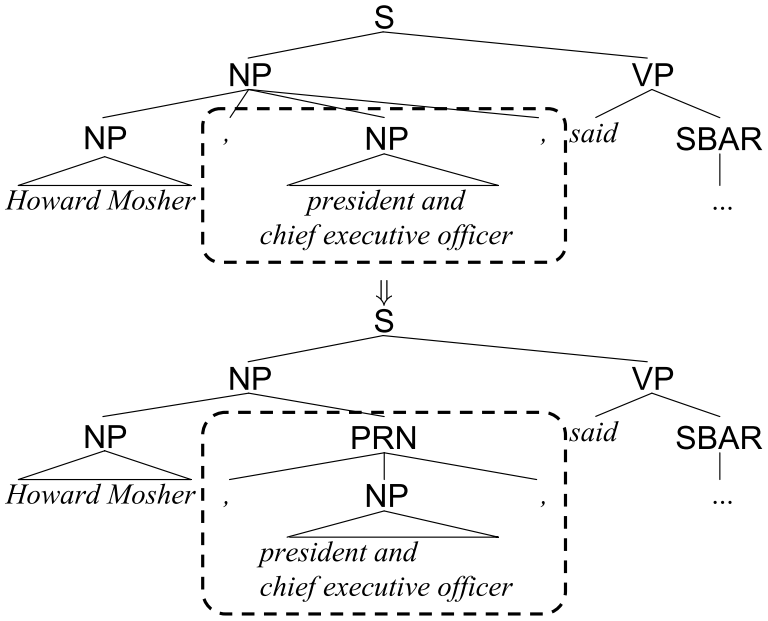


Figure 3.7: Annotation of an apposition construction

Coordination In most cases, coordination constructions are not explicitly expressed in Penn Treebank. For example, the upper side of Figure 3.6 shows a sentence with a coordination construction. Three VP conjuncts are put in a flat structure. This structure is converted into a left-branching structure by an algorithm similar to Hockenmaier and Steedman (2002a). CONJ is a nonterminal symbol assigned to the constituent of a conjunction and its right conjunct. The Coordination-Right Schema will be applied to CONJ nodes, and the Coordination-Left Schema to their mother nodes.

Insertion & apposition The Penn Treebank annotation prefers flat representations for insertion and apposition. Such structures are transformed so as to explicitly express the inserted structure. For example, in Figure 3.7, a phrase bracketed by commas is extracted and put as an inserted constituent with a nonterminal symbol PRN[3].

Disagreements in constituent structures The annotation style of Penn Treebank is different from HPSG analysis for some constructions including small clauses, "than" clauses, and quantifier phrases. Such constructions are converted to the structures conforming to the analysis in HPSG. For example, small clauses appearing in the complement of verbs are annotated as "S" in Penn Treebank, while in HPSG the subject and the predicative of a small clause are separately subcategorized by the verb. Figure 3.8 shows a sentence with a small clause. The phrase "*the two institutions face-to-face*" is annotated as S in Penn Treebank, and is converted to a flat structure as in the lower side of the figure. The subject of the small clause ("*the two institutions*") and the predicate ("*face-to-face*") are subcategorized separately by the verb "*brought*".

Ambiguous symbols Several non-/pre-terminal symbols in a certain context are re-labeled as special symbols for distinguishing their syntactic behaviors. For example, 'by' phrases representing the subject of a passive construction are labeled as 'BY'. Infinitival marker 'TO' under 'VP' is annotated

[3]In Penn Treebank, PRN is a nonterminal symbol for inserted clauses/phrases

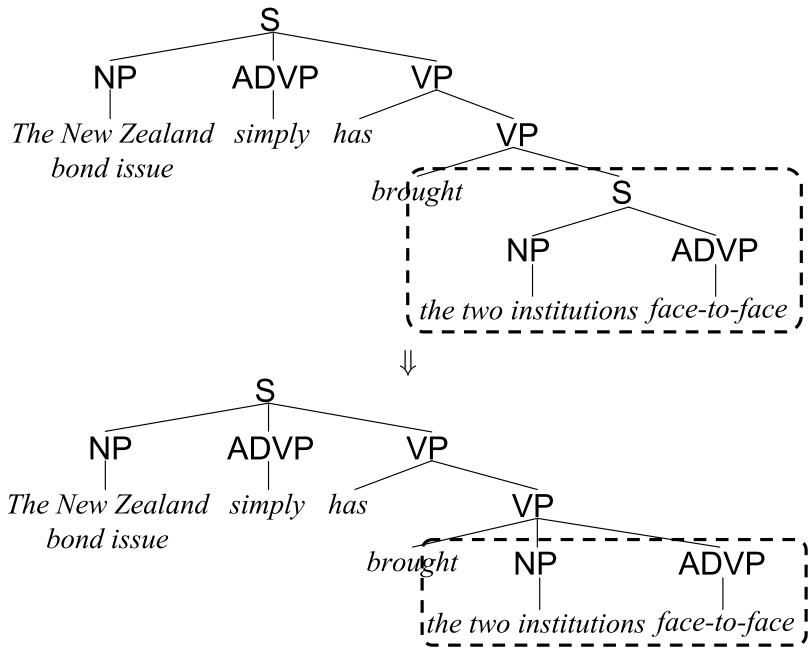


Figure 3.8: Annotation of a small clause

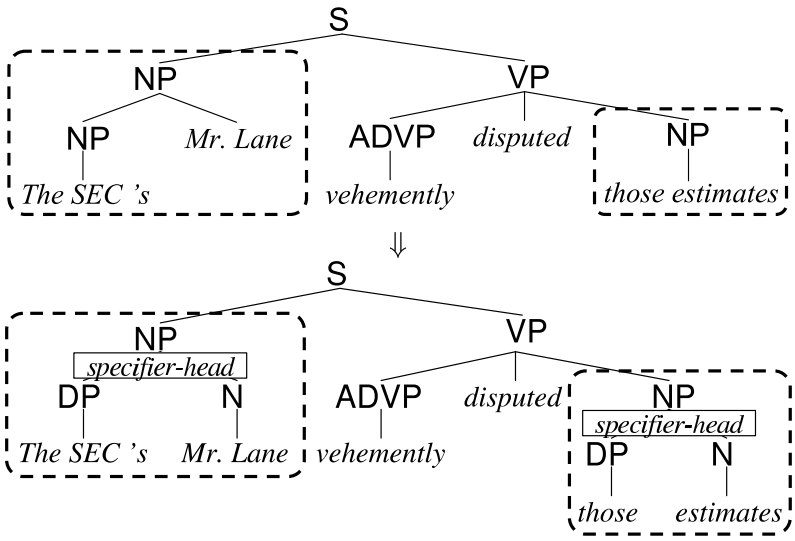


Figure 3.9: Annotation of determiners

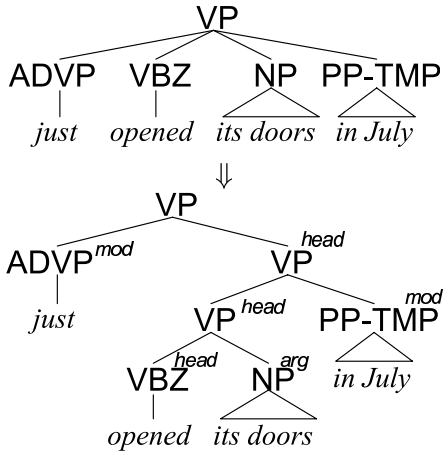


Figure 3.10: Head annotation and binarization

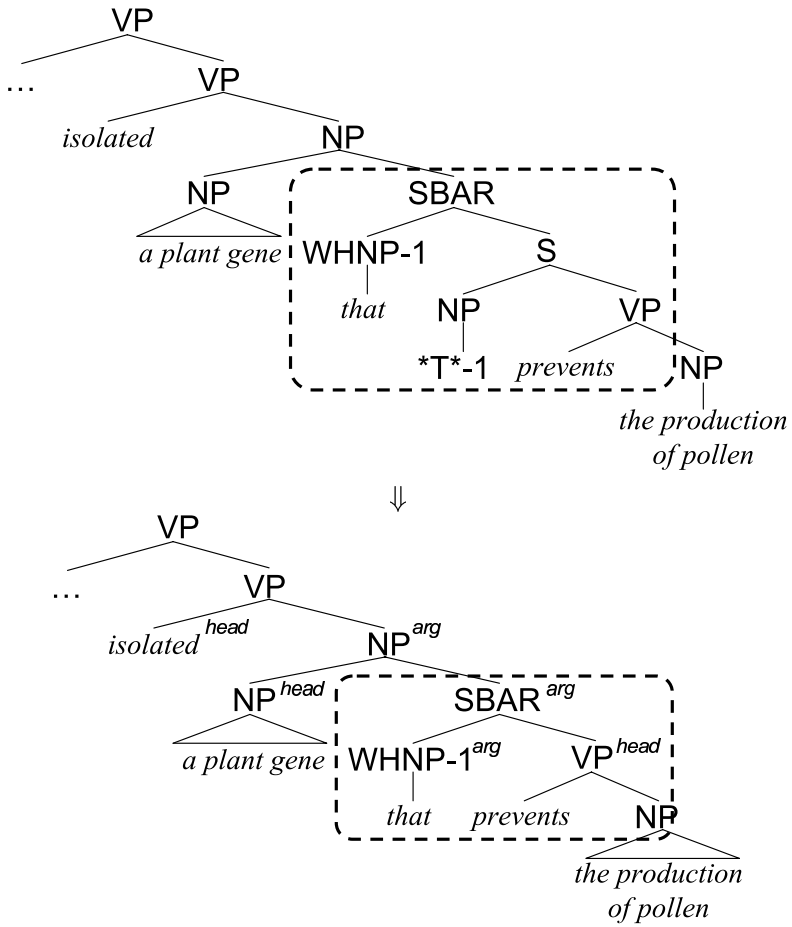


Figure 3.11: Annotation of subject extraction

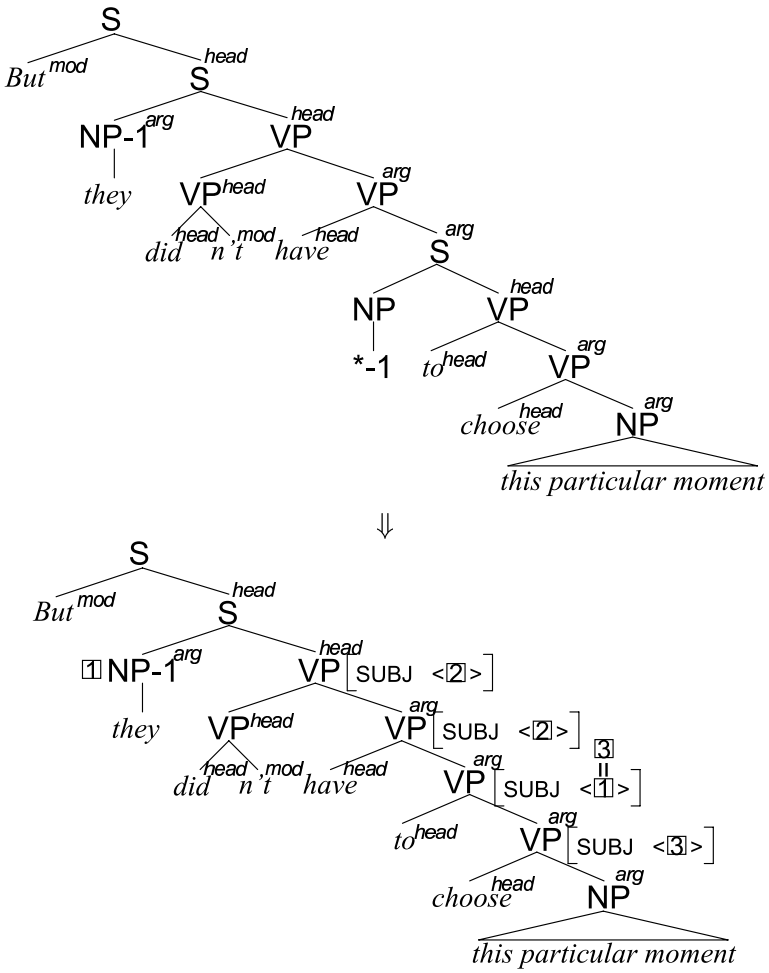


Figure 3.12: Annotation of subject-control and auxiliary verbs

as 'TOinf' for distinguishing it from prepositional "*to*". Complementizer "*that*" is assigned special symbol CM. Since subordinate conjunctions and prepositions share the same pre-terminal label IN in Penn Treebank, the labels of subordination conjunctions are relabeled as SC to distinguish them.

Determiners Noun phrases in Penn Treebank are represented as flat structures, and determiners are located at the same level as a head noun and modifiers. Determiners are raised to the distinct level and its construction is annotated as a Specifier-Head construction. Noun phrases with no determiners are labeled as an Empty-Specifier construction. Predeterminers are also processed in this step. Figure3.9 shows the annotation of determiners. First, the nonterminal symbol of a possessive phrase ("*The SEC 's*" in the figure) is converted from NP to DP (determiner phrase). Next, determiners including possessive phrases ("*The SEC 's*" and "*those*") are separated from their siblings. The noun phrases are then annotated as Specifier-Head constructions.

Head/argument/modifier annotation and binarization First, head/argument/modifier distinctions are annotated to each node in trees using *the head percolation table* (Magerman, 1995). A tree is then converted to a binary tree. Figure 3.10 shows an example of the conversion which is cited from Hockenmaier and Steedman (2002a). Readers should refer to Hockenmaier and Steedman (2002a)

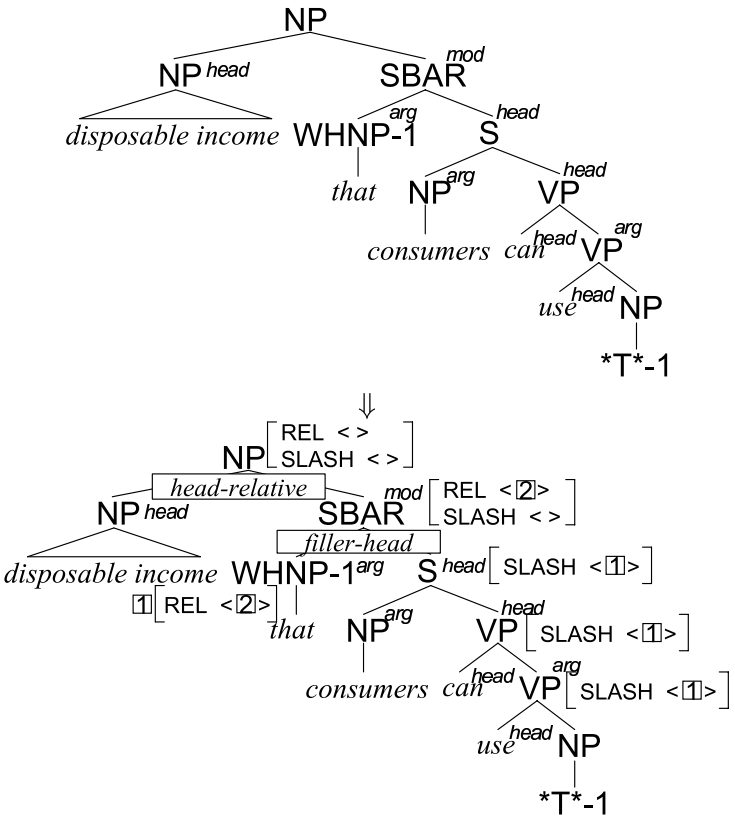


Figure 3.13: Annotation of slashes and relative clauses

for details.

Head features Pattern rules on sign or tree structures add constraints on features defined in HEAD, such as INV and CASE.

Subject extraction While the subject of a subject-extracted relative clause is annotated as a trace (*T*) in Penn Treebank, the HPSG does not introduce traces to this construction and a sub-clause takes a relative as a subject directly[4]. For example, Figure 3.11 shows an example of a subject-extracted relative clause. NP with a trace is eliminated, and the verb "*prevents*" directly takes the relative "*that*" as a subject.

Subject-control verbs Subject-control verbs such as "*try*" take VP as its complement in HPSG analysis, and the index of its subject is shared with the index of the unfilled subject of the VP. In the Penn Treebank, complements of control verbs are represented as S with the empty subject (the upper side of Figure 3.12). Such trees are annotated with the structure-sharings as shown in the lower side of Figure 3.12, where the SUBJ feature of to-infinitive is coindexed with NP-1 (represented by 1).

Auxiliary verbs HPSG regards an auxiliary verb as a head that takes VP as its complement, and the subject of the auxiliary verb is shared with the unfilled subject of the complement. This relation is

[4]Subject extraction in "*Who$_i$ did Kim claim __$_i$ left?*" is treated by another empirical annotation rule and the Subject Extraction Lexical Rule defined in Pollard and Sag (1994).

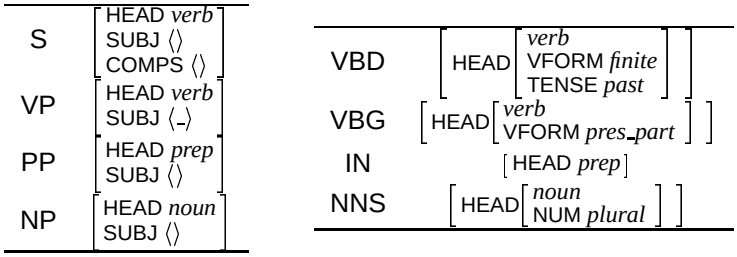


Figure 3.14: Mapping rules from Penn Treebank-style symbols into HPSG categories

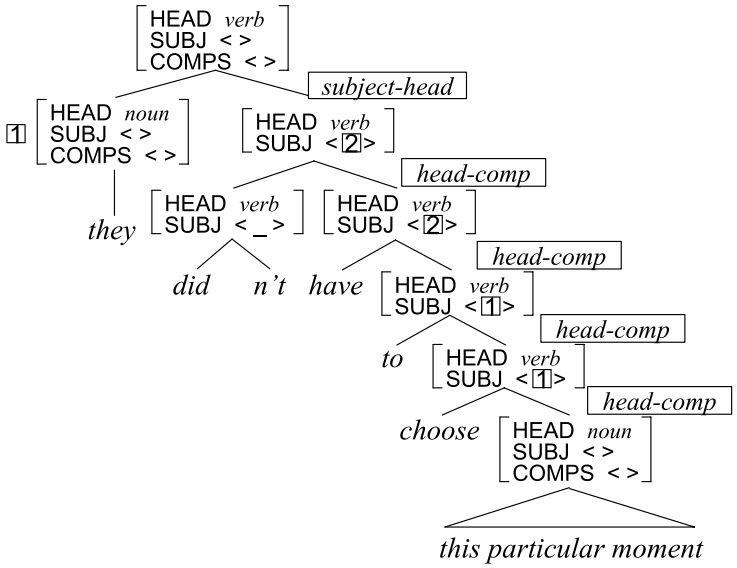


Figure 3.15: Partially-specified derivation tree corresponding to Figure 3.12

explicitly specified as shown in Figure 3.12 (represented by 2). Infinitival marker "*to*" is also treated as an auxiliary verb (represented by 3). AUX is also annotated in this step.

Slash & filler-head schema Penn Treebank-style annotation represents unbounded dependencies with trace label '*T*'. Unbounded dependencies are detected by finding the labels by the algorithm similar to the marking of *forward arguments* described by Hockenmaier and Steedman (2002a). If a trace label is found in a parse tree, its mother node has a non-empty list in SLASH (1 in Figure 3.13). SLASH is percolated to ancestors, and when a filler of the slash, i.e., the node with the same ID number, is found, the corresponding construction is annotated with the Filler-Head Schema (or the Filler-Insertion Schema), and SLASH of the mother node becomes an empty list.

Relative clauses As mentioned in Section 3.2, a special schema (Head-Relative Schema) is applied for relative clause constructions. When a relative clause is found, its construction is annotated with the Head-Relative Schema. In addition, a relative clause (SBAR) and its relativizer (WHNP) are annotated as having a non-empty list in REL (represented by 2 in Figure 3.13).

Category assignment Finally, each node is annotated with an HPSG category by mapping non-/pre-terminal symbols to HPSG categories. Figure 3.14 shows some of the mapping rules. Schema

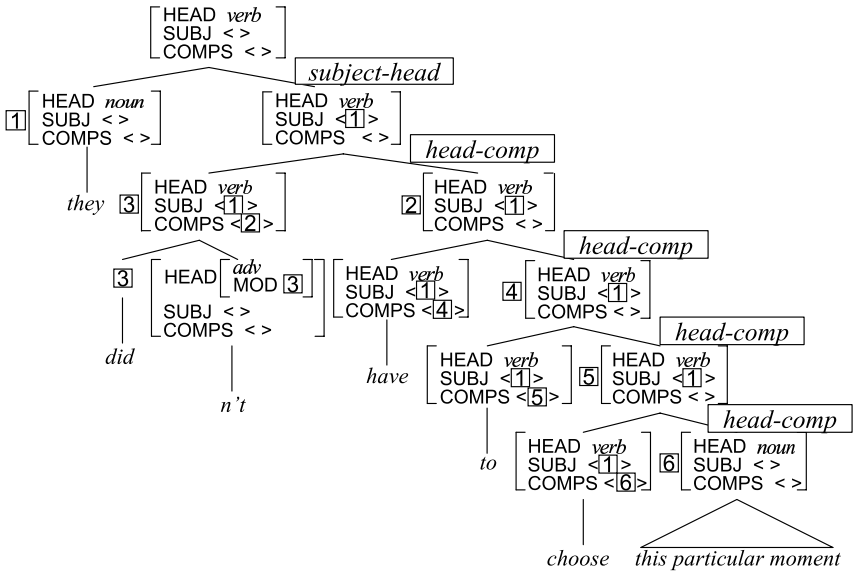


Figure 3.16: Fully-specified derivation tree corresponding to Figure 3.12

names are also assigned to all internal nodes that have not yet been assigned schema names. This is done by referring to head/argument/modifier annotations.

The above procedure converts Penn Treebank-style trees into partially-specified derivation trees of HPSG. For example, Figure 3.15 shows a partially-specified derivation tree corresponding to the Penn Treebank-style tree in Figure 3.12.

After converting Penn Treebank into partially specified derivation trees, linguistic principles, i.e, schemas and principles, are applied to each node in a tree to fill out unspecified constraints and to check the consistency of the annotated constraints. For example, given the partially-specified derivation tree in Figure 3.15, the Subject-Head Schema is applied to the root of the tree. Then, the constraints of the left daughter are percolated to SUBJ of the right daughter (represented by 1 in Figure 3.16). Subsequently, by applying the linguistic principles to the tree, we will obtain a fully specified derivation tree as shown in Figure 3.16, which is the same derivation tree as shown at the beginning of this chapter (Figure 3.2).

3.4 Extraction of Lexical Entries

Given HPSG derivation trees, we collect lexical entries from terminal nodes of the derivation trees. For example, from the derivation tree in Figure 3.16, we obtain a sign shown in Figure 3.17 for "*did*"[5].

Since signs collected from terminal nodes have too-specific constraints to be registered in a lexicon, they are generalized to *lexical entry templates* by removing some of the constraints. Lexical entry templates are lexical signs without word-specific nor context-specific constraints. We manually listed feature paths to be removed. For example, from Figure 3.17, the value of PHON is removed because it represents a surface form of the word and is obviously word-specific information. Some of the values of HEAD features, although they are not mentioned in Figure 3.17, in selectional features (MOD, SPR, SPEC, SUBJ, COMPS, CONJ, SLASH, and REL) are also removed if they are irrelevant to the syntactic constraints in English. For example, the value of AUX, which represents whether the

[5] HEAD features such as VFORM and AUX are omitted from the figure for the space limitation.

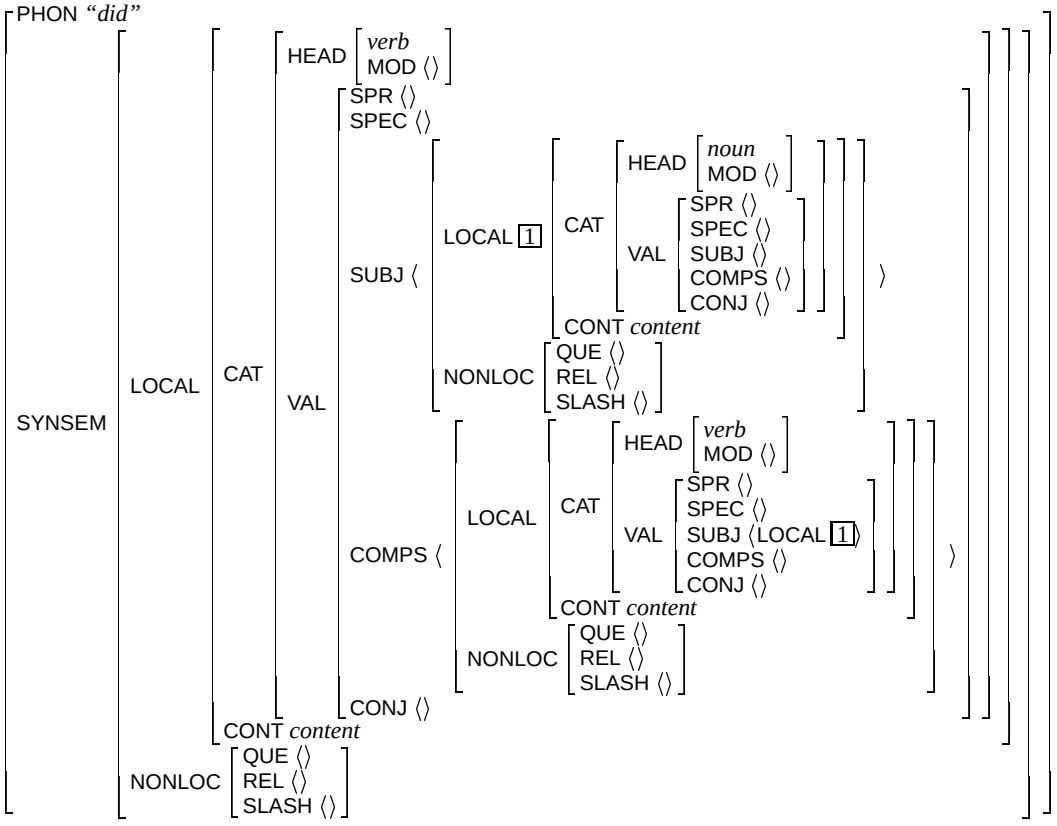


Figure 3.17: Terminal node for "*did*" extracted from the derivation tree in Figure3.16

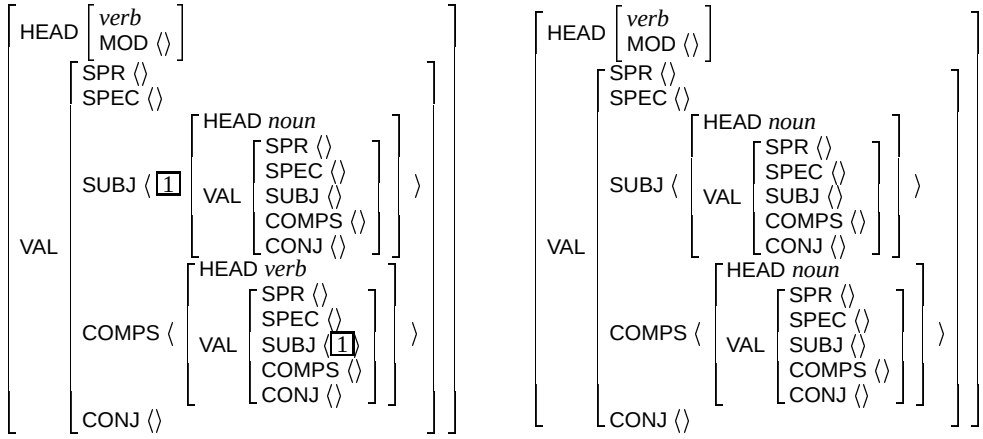


Figure 3.18: Lexical entry templates for "*did*" (left) and "*choose*"

phrase is headed by an auxiliary verb, is removed. As a result, we obtain lexical entry templates as shown in Figure 3.18 for "*did*" and for "*choose*"[6]. A lexicon stores mappings from a word into a set of lexical entry templates.

Additionally, predicate-argument dependencies are constructed using patterns on the structure of a lexical entry template. In HPSG, syntax-to-semantics mappings are represented in lexical entries.

[6]Only the values of CAT are shown in the figure because of the space limitation.

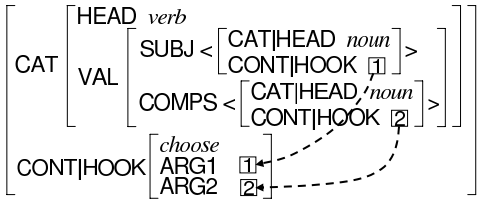


Figure 3.19: Mapping from syntactic arguments to semantic arguments

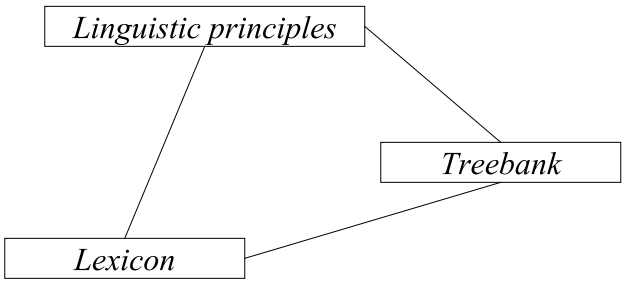


Figure 3.20: Grammar resources

For example, when we have a lexical entries for *"choose"* as shown in Figure 3.19, the lexical entry includes mappings from syntactic arguments (SUBJ and COMPS) into semantic arguments (ARG1 and ARG2).

Since lexical entry templates already have the information on syntactic arguments, what is needed is to identify the correspondence between the order of syntactic arguments and that of semantic arguments. Argument labels (ARG1, ARG2, ...) are basically defined in a left-to-right order of syntactic expressions. Additionally, syntactic alternations by long-distance dependencies, passive constructions, and dative shift are canonicalized since we have a cue for a movement in the Penn Treebank. For example, the Penn Treebank provides the annotations of the movement for non-local dependencies as shown in Figure 3.13. Referring to this annotation, we can place moved arguments in a canonical position. For passive and dative-shift constructions, since Penn Treebank distinguishes "by"-phrases to express a passive subject and "to"-phrases for dative shift constructions, they are also placed in their canonical argument position.

Note that the above procedure can canonicalize syntactic alternations, while it cannot handle lexical alternations supposed in PropBank (Kingsbury and Palmer, 2002; Kingsbury et al., 2002). For example, in the annotation scheme of the PropBank, a syntactic subject of an ergative verb is annotated as a second semantic argument. Since we have no cue for detecting this alternation given only the annotation of the Penn Treebank, it is assigned a first semantic argument in our grammar. We will be able to treat such alternations when we employ additional linguistic resources such as PropBank or other lexical databases.

3.5 Discussion and Related Work

In this section, we discuss differences among three strategies for grammar development: conventional grammar development, grammar learning from a treebank, and corpus-oriented development. As discussed in the introduction, the goals of the three strategies are the same: the development of three grammar resources, i.e., linguistic principles, a lexicon, and a treebank (Figure 3.20). Lin-

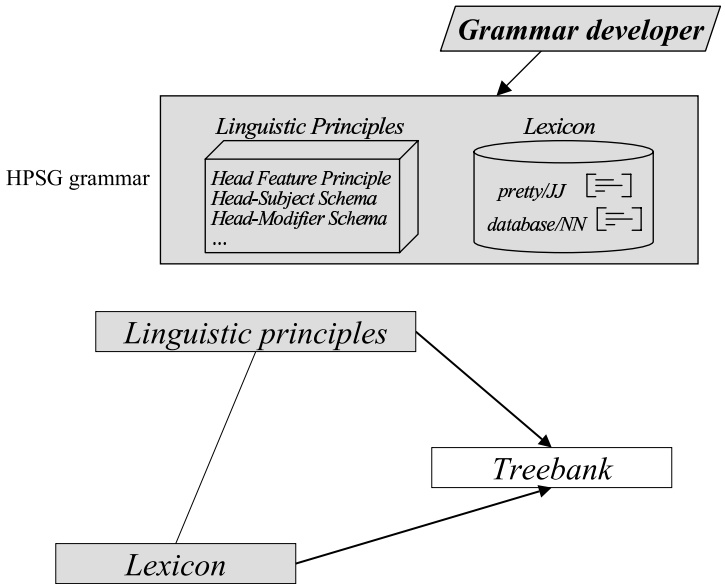


Figure 3.21: Manual development of an HPSG grammar

guistic principles are a formulation of a grammar theory, and represent generic grammatical constraints/structures. A lexicon represents mappings from a word into its lexical entry templates. A treebank is a collection of syntactic structures (in our case, derivation trees of HPSG) that are deduced by the combination of lexical entries and linguistic principles. These resources do not exist independently; if we specify two, the third is constrained by the two. Hence, we should note the order of the development, i.e., which resources should be constructed from scratch, and which should be (semi-)automatically developed.

Conventionally, grammar writers manually developed linguistic principles and a lexicon (Figure 3.21). The developed principles and the lexicon were verified by parsing test sentences and observing parse results. Most of the implemented lexicalized grammars were developed using this strategy; examples include LTAG for English (XTAG Research Group, 2001), LFG grammars (Butt et al., 2002), HPSG for English (Copestake and Flickinger, 2000; Flickinger, 2002), and HPSG for Japanese (Mitsuishi et al., 1998; Siegel, 2000; Siegel and Bender, 2002). However, almost all have not been scaled up to the level of analysis of real-world texts and have not been used in practical applications. A few systems that could analyze real-world sentences have had to rely on underspecification (Mitsuishi et al., 1998), or lots of robust parsing techniques and considerable human effort that lasted for over a decade (Riezler et al., 2002). The scaling-up of a lexicalized grammar has required the maintenance of the consistency of complex constraints in linguistic principles and lexical entries. When a linguistic principle or a lexical entry was adjusted for treating some words/constructions, the adjustment often negatively impacted on other words/constructions.

Recently, Oepen et al. (2002b) proposed a treebank-based method for the maintenance of an HPSG grammar. This method has been applied to the development and the maintenance of HPSG treebanks/grammars for English (Oepen et al., 2004, 2002b) and for Japanese (Bond et al., 2004). Their idea was that an HPSG treebank was developed by parsing sentences with a manually-tailored grammar. Through the development of the treebank, they found shortages of the grammar, and the findings were used for the refinement of the grammar. This feedback resembles our approach because they exploited a treebank for finding the shortages of a grammar. However, their approach basically followed the conventional scheme of grammar development, i.e., manual construction of a lexicon

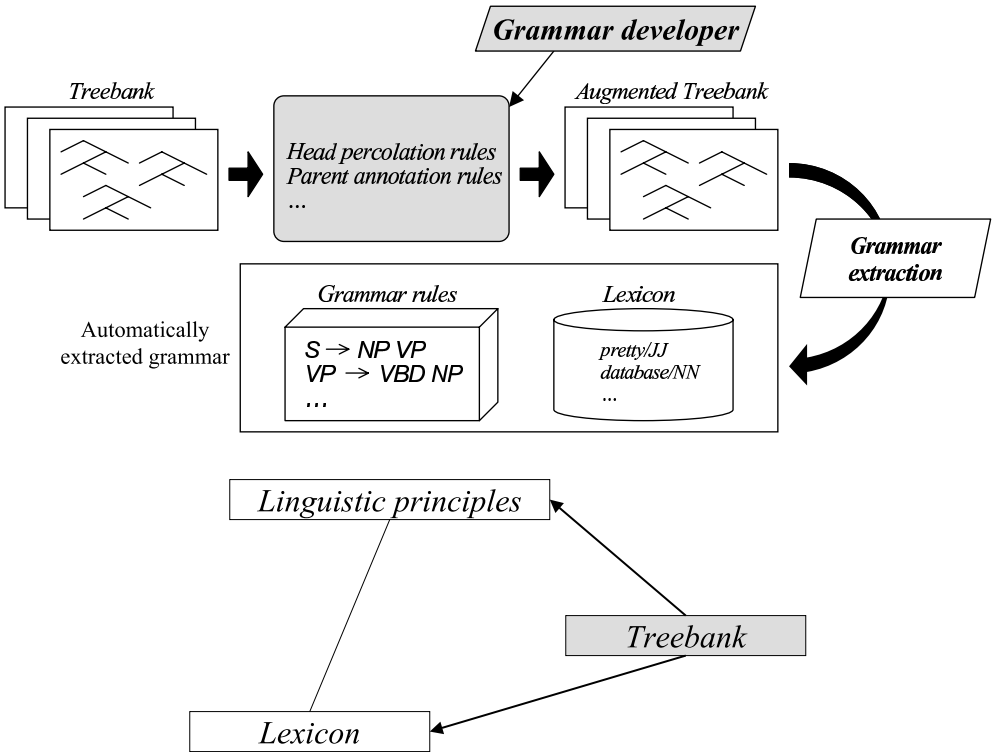


Figure 3.22: Grammar learning from a treebank

and verification by consulting a test corpus. The feedback relied on the abstraction of linguistic knowledge by grammar developers. From our perspective, their method is considered as one of the various approaches to descriptions of a treebank, but not necessarily the only nor the best way.

Corpus-based or example-based parsing has broken out from the impracticality of manually tailored grammars. The idea was that if we were given a collection of manually parsed data, i.e., a treebank, grammar rules and a lexicon would be automatically learned from the data. State-of-the-art statistical parsers (Charniak, 2000; Collins, 1997, 2003) followed this idea, although they did not extract a CFG explicitly. Interestingly, they enhanced parsing accuracy by augmenting linguistic annotations to a source treebank using various heuristics (Figure 3.22), such as head percolation rules (Magerman, 1995). In these methods, since a treebank consisted of analyses of real-world sentences, extracted grammars were sufficiently robust to analyze real-world texts. However, since the development of a treebank is expensive it usually provides only shallow surface structures such as CFG-style phrase structures. We hardly expect that we can build a treebank of a lexicalized grammar formalism such as HPSG from scratch. Another problem is that automatically extracted grammars are not formulated in conformity with a syntactic theory. The relationship between a syntactic theory and automatically extracted grammar rules is unclear. This will be an obstacle to extending a grammar while incorporating a new linguistic theory.

Recently, several studies have been proposed for the acquisition of lexicalized grammars from Penn Treebank (Cahill et al., 2002; Chen and Vijay-Shanker, 2000; Chiang, 2000; Frank et al., 2003b; Hockenmaier and Steedman, 2002a; Xia, 1999). They obtained a treebank of a lexicalized grammar by augmenting Penn Treebank using empirical annotation rules, and extracted a grammar from the obtained treebank. For example, in the case of LTAG, Xia (1999), Chen and Vijay-Shanker (2000), and Chiang (2000) developed algorithms for annotating Penn Treebank with substitution/adjunction (con-

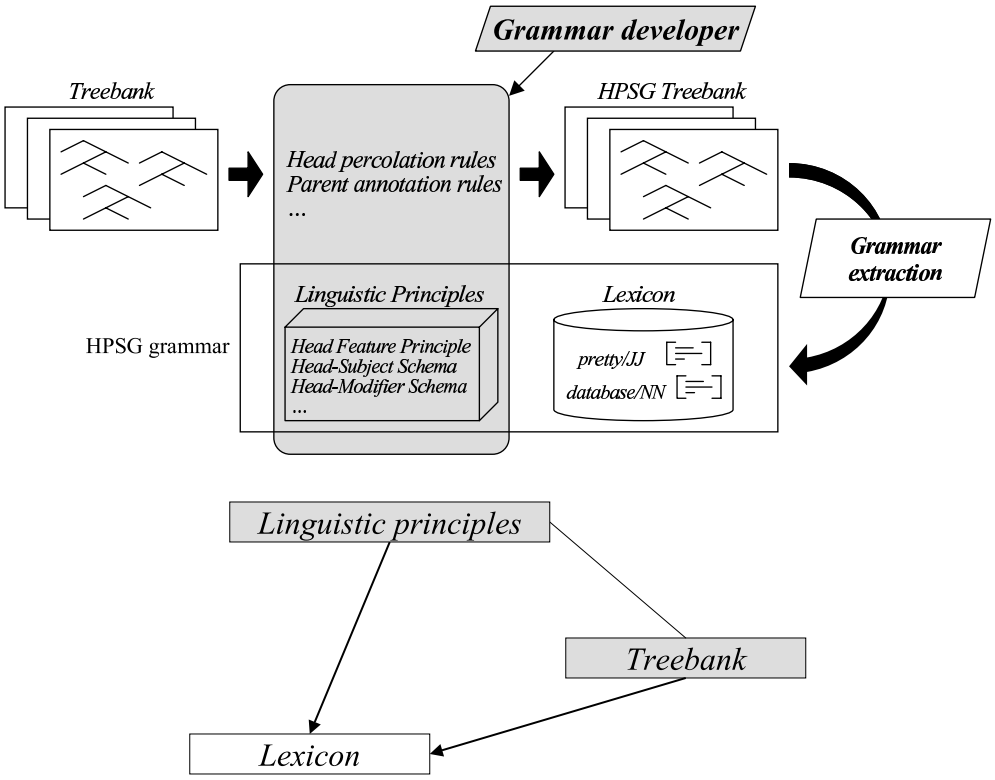


Figure 3.23: Corpus-oriented development of an HPSG grammar

struction rules in LTAG), and extracted elementary trees (lexical entries in LTAG) by splitting parse trees according to the annotations. Hockenmaier and Steedman (2002a) converted the Penn Treebank into *CCGBank*, from which lexical categories (lexical entries in CCG) and construction rules were extracted. Several methods have been proposed to extract LFGs from treebanks (Cahill et al., 2002; Frank et al., 2003b). Their aim was to automatically annotate c-structures with *functional schemata*, which are unification-based construction rules in LFG. That is, they automated the development of LFG-style construction rules. This means that both lexicon and linguistic principles are automatically extracted from a treebank.

The above studies demonstrated that existing treebanks could be reused for acquiring lexicalized grammars. However, the methods basically followed those of extracting of augmented CFGs. This means that extracted grammars were not theoretically formulated in conformity with manually-tailored linguistic principles. Hence, the conformity to a syntactic theory depended directly on the design of empirical annotation rules. Since annotation rules for treebank conversion had no theoretical formulation, we cannot be assured that extracted grammars conform to a syntactic theory.

Our study further pursues their approach. In our approach, however, model-theoretic linguistic principles and empirically designed annotation rules are clearly separated. Our method forces grammar developers to write linguistic principles, and verifies a treebank to satisfy linguistic principles. Linguistic principles are never extracted from a treebank, but are manually controlled by grammar writers. The target of annotation is partial constraints of derivation trees. Manually defined linguistic principles are exploited to verify the consistency of derivation trees and the conformity to a grammar theory (Figure 3.23). That is, the consistency and the conformity of the treebank are controlled by linguistic principles, and therefore by grammar writers. Inconsistencies of annotated constraints

are automatically detected as violations of linguistic principles. If no inconsistencies are found, partially specified constraints are organized into a lexicon. Lexical entries are systematically connected to derivation trees in conformity with linguistic principles. Verification by linguistic principles is necessary for the development of HPSG, because constraints in HPSG are more complicated and fine-grained than LTAG and CCG, and the maintenance of the consistency is far more difficult.

To summarize, our methodology has the following advantages compared to the conventional development methods.

Automatic verification of consistency Errors or inconsistencies in a treebank are automatically detected as violations of linguistic principles. That is, the process of treebank development inherently includes the verification of consistency. Hence, grammar developers are always aware of conflicts in a treebank, empirical annotation rules, and linguistic principles. The consistency of a lexicon collected from the treebank is also assured with respect to linguistic principles.

Availability for machine learning A treebank can be used as training data for the probabilistic modeling or machine learning of statistical parsing.

Concreteness of annotation Treebank development is more comprehensible than lexicon development because the target of annotation is instances of analyses of concrete sentences rather than abstract rules. The writing of analyses of concrete sentences is obviously easier than the simulation of abstract rules, because grammar developers are not hampered by simulating in their mind all usages of each word.

Low cost We can pick the fastest way for the establishment of a treebank, and may exploit various methods and existing resources. In this study, we exploit Penn Treebank, and the dominant cost in our method is in maintaining empirical annotation rules. In addition, annotation may be carried out for a closed set of a corpus because we do not intend to apply annotation rules directly to open-class sentences.

A treebank and a lexicon developed by our method do not seem to be theoretically formulated because the development depended on empirical annotation rules to convert existing language resources. However, independently of the methods of treebank development, obtained lexical entries are assured to deduce the derivation trees given in the treebank, because linguistic principles strictly regulate the relationship between derivation trees and lexical entries. Hence, lexicalized grammars developed with our methodology inherently satisfy the goal of a grammar theory, i.e., to define lexical entries and linguistic principles that deduce derivation trees. Admittedly, our methodology does not assure the generality of the grammar, i.e., how many unseen derivation trees the extracted lexical entries can deduce, and the specificity, i.e., how many illegal derivation trees the lexical entries can reject. They depend on the quality and the size of the treebank. Hence, the treebank must be maintained and improved by grammar developers through the improvement of annotation rules assisted by the consistency verification by linguistic principles.

It might be asserted that a grammar developed by our method has less constraints and is less restrictive. This is because existing corpus-based methods aimed at the automatic learning of underlying regularity that was implicitly represented in training data. As a result, they could learn statistical regularity, but failed to learn structural linguistic constraints. In fact, we start from real data and extract a grammar from it. However, our methodology does not aim at the automatic learning of underlying regularity from a treebank. Our method cannot learn any constraints that are not explicitly annotated in a treebank; grammar developers must externalize linguistic constraints manually by consulting their linguistic intuition and observation. The difference from conventional grammar development is

only that the target of writing linguistic intuition is a treebank instead of a lexicon. Since the difference is only in the target of development, there should be no difference in the restrictiveness of the grammar. A treebank is a testbed for externalizing metaphysical linguistic entities by providing concrete examples of the target of explanation, and a ground for the verification of the consistency of a grammar. Hence, the concept proposed in this thesis is not a new algorithm of grammar learning, but a novel scheme of grammar development. The methodology has abandoned the automatic learning of syntactic regularity from a treebank. The manual management of a treebank is necessary and is still a tough task for grammar developers. This is why our methodology is not a corpus-oriented grammar nor corpus-oriented parsing, but *corpus-oriented development*.

The method described in this chapter was implemented for the development of an English grammar using Penn Treebank as a source treebank. Chapter 6 reports the detail of the implementation of the grammar and the results of evaluation experiments.

Chapter 4

Feature Forest Model

Maximum entropy models are widely accepted for the probabilistic modeling of various tasks because of their high accuracy. However, studies to date have simply applied maximum entropy models to traditional models, such as n-grams and lexicalized versions of a probabilistic context-free grammar (PCFG), which rely on the *independence assumption* to divide a probabilistic event into sub-events. For example, part-of-speech tagging is divided into the tasks of assigning a tag to a word, and CFG parsing is divided into the applications of rewriting rules. The probability of a complete event, i.e., the tagging of a sentence or the construction of a parse tree, is defined as the product of the probabilities of sub-events. Maximum entropy models have been applied to the modeling of the probabilities of sub-events. However, such methods inherently restrict the flexibility offered by maximum entropy models, which do not require independence assumption.

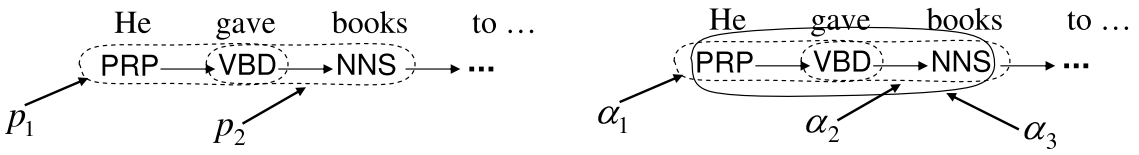
The algorithm proposed in this chapter provides a solution for the probabilistic modeling of complete structures, such as transition sequences in Markov models and parse trees, without dividing them into independent sub-events (Miyao and Tsujii, 2002). In general, complete structures have an exponential number of ambiguities. This causes exponential explosion when estimating parameters of a maximum entropy model. Our algorithm presented in this chapter avoids exponential explosion by representing an event with *a feature forest*, which is a packed representation of trees. When complete structures are represented with feature forests of a tractable size, parameters are efficiently estimated by dynamic programming similar to the algorithm of computing inside/outside probabilities in PCFG parsing without unpacking feature forests.

The proposed algorithm provides a scheme for more flexible modeling of various NLP tasks because model designers can incorporate various overlapping features of a complete structure, not only of sub-events, into the model. Moreover, it can be applied to probabilistic models of events that are difficult to divide into independent sub-events, such as a probabilistic model of feature structures (Abney, 1997). Feature forest models offer a general solution to the probabilistic modeling of complex data structures, and are indispensable for the probabilistic modeling of HPSG parsing.

Section 4.1 discusses the problem of combinational explosion with conventional probabilistic models. Section 4.2 proposes an algorithm for solving the problem. Section 4.3 shows proofs of the theories in Section 4.2. Section 4.4 introduces studies related to our algorithm.

4.1 Problem

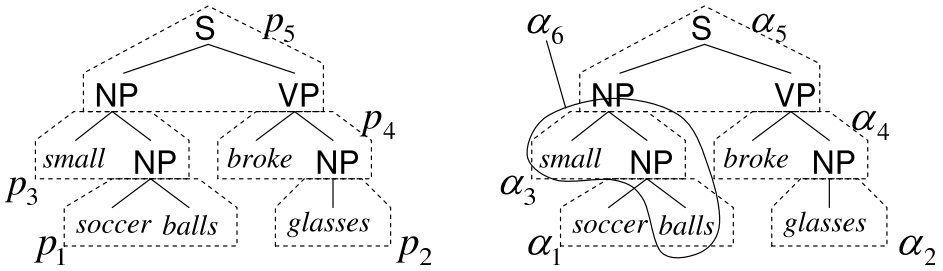
Let us consider maximum entropy modeling for tagging tasks. Conventional models divide a tagging sequence into a tagging event for each word (Kazama et al., 2001; Ratnaparkhi, 1996). That is, the models incorporate independence assumption, which is not required in maximum entropy models.



Bigram model
$p = p_1 p_2 \cdots$

Model without independence assumption
$p = \frac{1}{Z}\alpha_1 \alpha_2 \alpha_3 \cdots$

Figure 4.1: Probabilistic models for POS tagging



PCFG-like model
$p = p_1 p_2 p_3 p_4 p_5$

Model without independence assumption
$p = \frac{1}{Z}\alpha_1 \alpha_2 \alpha_3 \alpha_4 \alpha_5 \alpha_6 \cdots$

Figure 4.2: Probabilistic models for parsing

As introduced in Section 2.4, when tag sequence $\boldsymbol{t}$ is assigned to sentence $\boldsymbol{s}$, the probability $p(\boldsymbol{t}|\boldsymbol{s})$ is defined as follows:

$$\begin{aligned} p(\boldsymbol{t}|\boldsymbol{s}) &= \prod_k p(t_k|\boldsymbol{s}, t_1, \ldots, t_{k-1}) \\ &= \prod_k p(t_k|\boldsymbol{s}, t_{k-1}). \qquad \text{(by Markov assumption)} \end{aligned}$$

This definition relies on the Markov assumption, namely a tagging event depends only on the tagging of the previous word. Conventional studies on tagging applied a maximum entropy model to probabilistic modeling of $p(t_k|\boldsymbol{s}, t_{k-1})$.

Figure 4.1 shows two tagging models: a conventional model (on the left) and maximum entropy model without independence assumption (on the right). In the maximum entropy model on the right, a complete tag sequence of a sentence is considered to be a target event. The tagging to a sentence is regarded as a monolithic event without any independence assumption. The model chooses the most probable tag sequence from the set of all possible tag sequences. In this model, we can incorporate various features such as bigram/trigram/n-gram features; for example, α_3 is a trigram feature in the right-hand side of Figure 4.1. This model is an extension of the conventional model shown in the left-hand side of the figure, and is a more powerful modeling scheme.

A similar argument can be made for the maximum entropy modeling of parsing. In conventional parsing models, similar to PCFGs, each branching is assumed to be an independent event, and the probability of a complete parse tree is defined as the product of the probabilities assigned to the branchings (left-hand side of Figure 4.2) (Ratnaparkhi, 1997, 1999). On the other hand, we consider a maximum entropy model that assigns a probability to the complete parse tree without independence assumption (right-hand side of Figure 4.2). Feature functions correspond not only to the branchings but also to other characteristics such as α_6 in a parse tree. The model is formulated as a probabilistic

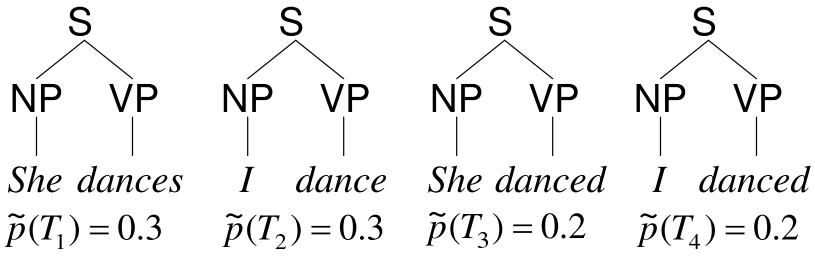


Figure 4.3: Corpus of a feature structure grammar

$p(\mathsf{S} \rightarrow \mathsf{NP\ VP}) = 1.0$
$p(\mathsf{NP} \rightarrow \textit{she}) = 0.5$
$p(\mathsf{NP} \rightarrow \textit{I}) = 0.5$
$p(\mathsf{VP} \rightarrow \textit{dances}) = 0.3$
$p(\mathsf{VP} \rightarrow \textit{dance}) = 0.3$
$p(\mathsf{VP} \rightarrow \textit{dance}) = 0.4$

Figure 4.4: PCFG extracted from Figure 4.3

model of selecting parse tree t from a parse forest for sentence $\mathbf{w}$. As described above, maximum entropy models do not require independence assumption, and various overlapping features, not limited to immediate dominance relations, can be incorporated. Obviously, this model is a natural extension of PCFG, and is a more powerful modeling scheme.

The above two examples reveal that previous studies merely applied a maximum entropy model to the estimation of the probability of sub-events such as a tagging to each word and a branching in a parse tree (Kazama et al., 2001; Ratnaparkhi, 1996, 1997, 1999).

For probabilistic modeling of feature structures grammars including HPSG, maximum entropy models are theoretically indispensable because feature constraints violate the independence assumption in PCFG-like models. As described below, in feature structure grammars, decomposition of probabilities into rule probabilities causes distortion of a probabilistic distribution (Abney, 1997). That is, a probability of a parse tree cannot be divided into independent sub-events. Conventional approaches cannot be applied to this task.

Suppose that we make a probabilistic model of CFG-style parse trees shown in Figure4.3. A corpus includes four types of parse trees and five words: "*she*", "*I*", "*dances*", "*dance*", and "*danced*". Observed relative frequencies are shown at the bottom of each parse tree. When we assume a probabilistic CFG generated these parse trees, a PCFG is estimated as shown in Figure4.4. Rule probabilities are estimated by counting a relative frequency of each branching of a parse tree. For example, the CFG rule "NP→*she*" is used in the first and the third trees, and their frequencies are 0.3 and 0.2, respectively. Hence, we estimate $p(\mathsf{NP} \rightarrow \textit{she}) = 0.5$.

This PCFG model assigns probabilities to the trees as shown in Figure4.5. The results show that the two parse trees on the left are assigned half of the observed relative frequencies. In the treebank, the frequencies of these two trees are higher than those of the two trees on the right, while the PCFG model gives them lower probabilities. This is because a CFG produces ungrammatical parse trees, and probabilities are assigned to them. Figure4.6 shows the sources of missing probabilities of the PCFG model.

Feature structures are required to eliminate ungrammatical parse trees; however, they cannot cor-

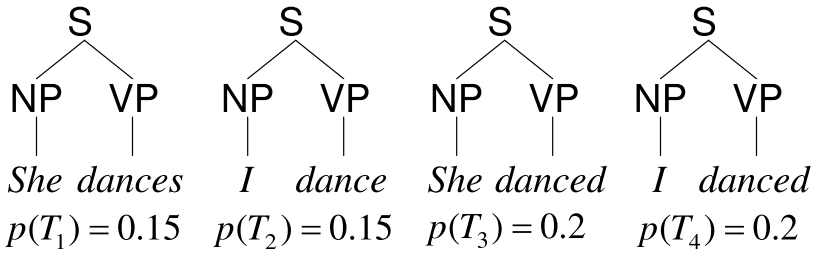


Figure 4.5: Probabilities given by the PCFG in Figure 4.4

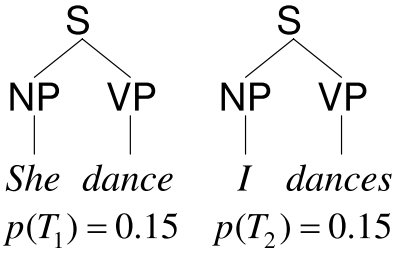


Figure 4.6: Missing probabilities

$\mathsf{S} \rightarrow \mathsf{NP}_{\boxed{1}}\ \mathsf{VP}_{\boxed{1}}$	$\mathsf{VP}_{no\_3sg} \rightarrow \textit{dance}$
$\mathsf{NP}_{no\_3sg} \rightarrow \textit{I}$	$\mathsf{VP}_{3sg} \rightarrow \textit{dances}$
$\mathsf{NP}_{3sg} \rightarrow \textit{she}$	$\mathsf{VP}_{no\_3sg or 3sg} \rightarrow \textit{danced}$

Figure 4.7: CFG rules with constraints on agreements

rect the distortion of probabilities. For example, Figure 4.7 shows a grammar augmented with constraints on agreements. Suffixes represent a type of agreement, and $\boxed{1}$ denotes a variable. Variables with the same names must have the same values. Hence, the grammar can eliminate the parse trees in Figure 4.6 because the suffixes of NP and VP are different. However, rule probabilities are not changed only by the introduction of feature structures. We cannot solve the problem of disordered probabilities because a PCFG-like estimation method yields the same probabilities as in Figure4.4.

For feature structure grammars, maximum entropy models are necessary for developing a probabilistic model that gives probabilities equivalent to relative frequencies in the treebank and for attaining the maximum likelihood of the treebank. As described in Section 2.4, maximum entropy models give a probabilistic distribution that maximizes the likelihood of training data under given feature functions. In fact, a maximum entropy model of the grammar in Figure4.7 gives probabilities equivalent to relative frequencies in the corpus in Figure 4.3. When we assign feature functions to grammar rules in Figure 4.7, maximum entropy estimation provides feature weights as in Figure4.8. Given the parameters, unnormalized weights of parse trees are computed as:

$$\begin{aligned}
\hat{p}(T_1) &= 1.0 \times 1.0 \times 1.145 = 1.145, \\
\hat{p}(T_2) &= 1.0 \times 1.0 \times 1.145 = 1.145, \\
\hat{p}(T_3) &= 1.0 \times 1.0 \times 0.763 = 0.763, \\
\hat{p}(T_4) &= 1.0 \times 1.0 \times 0.763 = 0.763.
\end{aligned}$$

$$\alpha(\mathsf{S} \rightarrow \mathsf{NP}_{\boxed{1}}\ \mathsf{VP}_{\boxed{1}}) = 1.0$$
$$\alpha(\mathsf{NP}_{\textit{no\_3sg}} \rightarrow \textit{I}) = 1.0$$
$$\alpha(\mathsf{NP}_{\textit{3sg}} \rightarrow \textit{she}) = 1.0$$
$$\alpha(\mathsf{VP}_{\textit{no\_3sg}} \rightarrow \textit{dance}) = 1.145$$
$$\alpha(\mathsf{VP}_{\textit{3sg}} \rightarrow \textit{dances}) = 1.145$$
$$\alpha(\mathsf{VP}_{\textit{no\_3sgor3sg}} \rightarrow \textit{danced}) = 0.763$$

Figure 4.8: Feature weights of a maximum entropy model for the grammar in Figure 4.7

By normalizing the above weights, we obtain probabilities given by the model, which are equivalent to the relative frequencies of the training corpus.

$$\begin{aligned}
p(T_1) &= 1.145/(1.145 + 1.145 + 0.763 + 0.763) = 0.300, \\
p(T_2) &= 1.145/(1.145 + 1.145 + 0.763 + 0.763) = 0.300, \\
p(T_3) &= 0.763/(1.145 + 1.145 + 0.763 + 0.763) = 0.200, \\
p(T_4) &= 0.763/(1.145 + 1.145 + 0.763 + 0.763) = 0.200.
\end{aligned}$$

The remaining issue is parameter estimation. The number of targets y, i.e., the number of possible tag sequences or parse trees for a sentence, is generally very large. This is because local ambiguities in a tag sequence or a parse tree result in exponential growth in the number of structures, resulting in billions of structures. For example, when we apply rewriting rule "$\mathsf{S} \rightarrow \mathsf{NP}\ \mathsf{VP}$", and the left NP and the right VP respectively have n and m ambiguous subtrees, the result of the rule application generates $n \times m$ trees.

This is problematic because the complexity of parameter estimation is proportional to the number of target events as described in Section 2.4. The complexity is $O(|\tilde{Y}||\tilde{F}||E|)$, where $|\tilde{Y}|$ and $|\tilde{F}|$ are the average numbers of targets and activated features for an event, and $|E|$ is the number of events, respectively. The cost is bound by the computation of model expectations μ_i (Definition 2.10).

$$\begin{aligned}
\mu_i &= \sum_{x \in \mathcal{X}} \tilde{p}(x) \sum_{y \in Y(x)} f_i(x, y) p(y|x) \\
&= \sum_{x \in \mathcal{X}} \tilde{p}(x) \sum_{y \in Y(x)} f_i(x, y) \frac{1}{Z(x)} \exp\left(\sum_j \lambda_j f_j(x, y)\right).
\end{aligned}$$

As shown in the above definition, the computation of model expectation requires the summation over $Y(x)$ for every x in the training data. If $Y(x)$ grows exponentially, the parameter estimation is intractable.

In PCFGs, the problem of computing probabilities of parse trees is avoided by using a dynamic programming algorithm of computing inside/outside probabilities. With the algorithm, computation of probabilities of parse trees is tractable. We can expect that the same approach would be available for maximum entropy models.

This notion yields a new algorithm for parameter estimation for maximum entropy models, as described in the next section.

4.2 Solution

Our solution to the problem is a dynamic programming algorithm of computing *inside/outside α-products*. Inside/outside α-products roughly correspond to inside/outside probabilities in probabilistic context-free grammars (PCFGs). As shown in Figure 4.2, a maximum entropy model of a parse tree resembles a PCFG-like model. A probability is defined as a normalized product of $\alpha_j^{f_j} (= \exp(\lambda_j f_j))$. Hence, similar to the algorithm of computing inside/outside probabilities, we can compute $\exp\left(\sum_j \lambda_j f_j\right)$, which we define as *$\alpha$-product*, for each node in a tree structure. If we can compute α-products at a tractable cost, model expectations,

$$\mu_i = \sum_{x \in \mathcal{X}} \tilde{p}(x) \sum_{y \in Y(x)} f_i(x,y) \frac{1}{Z(x)} \exp\left(\sum_j \lambda_j f_j(x,y)\right), \tag{4.1}$$

are also computed at a tractable cost.

We first define *feature forest*, a packed representation of a set of an exponential number of tree structures. A feature forest corresponds to a packed chart of CFG parsing. We then define *inside/outside α-products* that represent the α-products of partial structures of a feature forest. Inside α-products correspond to inside probabilities in PCFG, and represents the summation of each α-product in one of the daughter sub-trees. Outside α-products correspond to outside probabilities in PCFG, and represents the summation of each α-product in the upper part of the feature forest. Both can be computed incrementally by a dynamic programming algorithm similar to the algorithm of computing inside/outside probabilities in PCFG. Given inside/outside α-products of all nodes in a feature forest, model expectation μ_i is easily computed by multiplying them for each node.

4.2.1 Feature Forest

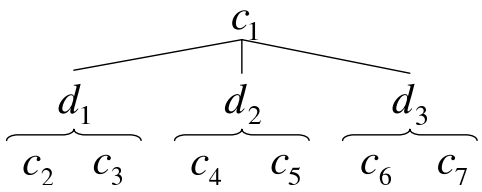
To describe the algorithm, we first define *feature forest*, the generalized representation of features in packed forest structure. Feature forests are used for enumerating possible structures of target events, that is, they correspond to $Y(x)$ in Equation 4.1.

Definition 4.1 (Feature forest) *Feature forest Φ is a tuple $\langle C, D, r, \gamma, \delta \rangle$, where:*

- *C is a set of conjunctive nodes,*
- *D is a set of disjunctive nodes,*
- *r is the root node: $r \in C$,*
- *$\gamma : D \mapsto 2^C$ is a conjunctive daughter function,*
- *$\delta : C \mapsto 2^D$ is a disjunctive daughter function.*

We denote a feature forest for history x as $\Phi(x)$. For example, $\Phi(x)$ represents a set of all possible tag sequences of given sentence x, or a set of all parse trees of x. We assume that a feature forest is an acyclic graph, while unpacked structures extracted from a feature forest are trees. We also assume that terminal nodes of feature forests are conjunctive nodes. That is, disjunctive nodes must have daughters, i.e., $\gamma(d) \neq \emptyset$ for all $d \in D$.

A feature forest represents a set of trees of conjunctive nodes in a packed structure. Conjunctive nodes correspond to linguistic entities such as states in Markov chains and nodes in CFG trees. Feature functions are assigned to conjunctive nodes and express their characteristics. Disjunctive nodes



$C = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7\}$
$D = \{d_1, d_2, d_3\}$
$r = c_1$
$\gamma(d_1) = \{c_2, c_3\},\ \gamma(d_2) = \{c_4, c_5\},\ \gamma(d_3) = \{c_6, c_7\}$
$\delta(c_1) = \{d_1, d_2, d_3\},\ \delta(c_2) = \{\},\ \delta(c_3) = \{\},\ \delta(c_4) = \{\},\ \delta(c_5) = \{\},\ \delta(c_6) = \{\},\ \delta(c_7) = \{\}$

Figure 4.9: Feature forest

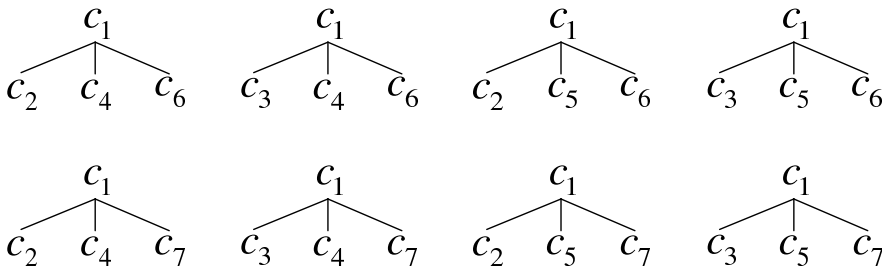


Figure 4.10: Unpacked trees

are for enumerating alternative choices. Conjunctive/disjunctive daughter functions represent immediate relations of conjunctive and disjunctive nodes. By selecting a conjunctive node as a child of each disjunctive node, we can extract a tree consisting of conjunctive nodes from a feature forest. Obviously, the models introduced in Section 4.1 can be represented with this structure.

Figure 4.9 shows an example feature forest. Each disjunctive node enumerates alternative nodes, which are conjunctive nodes. Each conjunctive node has disjunctive nodes as daughters. The feature forest in Figure 4.9 represents a set of $2 \times 2 \times 2 = 8$ unpacked trees shown in Figure4.10. For example, by selecting the left-most conjunctive node at each disjunctive node, we extract tree $(\mathrm{q}, c_2, c_4, c_6)$. Generally, a feature forest can represent an exponential number of trees with a polynomial number of nodes. Thus, a complete structure, such as tag sequences and parse trees with ambiguities, can be represented in tractable forms.

Feature functions are defined over conjunctive nodes[1].

Definition 4.2 (Feature function for feature forest) *A feature function for a feature forest is:*

$$f_i : C \mapsto \mathbf{R}.$$

Hence, together with feature functions, a feature forest represents a set of trees of features.

Feature forests may be regarded as a packed chart in CFG parsing. Conjunctive nodes correspond to active/inactive edges, and disjunctive nodes correspond to equivalence classes of edges. Although feature forests have the same structure as PCFG parse forests, each node in feature forests does not

[1]Feature functions may also be conditioned on history events. In this case, the domain of feature functions is a pair of C and x. For simplicity, we omit history events in the following discussion.

need to correspond to a node in PCFG parse forests. In fact, in Chapter 5, we will demonstrate that syntactic structures of HPSG and predicate argument structures can be represented with tractable-size feature forests. Instances of a node may thus be neglected in the following discussion, and our algorithm is applicable whenever feature forests are of a tractable size. Descriptive power of feature forests will be discussed again in Section 4.4.

As mentioned, a feature forest is a packed representation of trees of features. We first define model expectations, μ_i, on a set of unpacked trees, and then show that they can be computed without unpacking feature forests. We denote an unpacked tree as a set, $\mathbf{c} \subseteq C$, of conjunctive nodes. Our concern is only in a set of features associated with each conjunctive node, and the shape of the tree structure is irrelevant to the computation of probabilities of unpacked trees. Hence, we do not distinguish an unpacked tree from a set of conjunctive nodes.

A set of unpacked trees is defined as a multiset of unpacked trees because we allow multiple occurrences of equivalent unpacked trees in a feature forest[2]. Given sets of unpacked trees, A, B, we define a union and a product as follows.

$$\begin{aligned} A \oplus B &\equiv A \cup B \\ A \otimes B &\equiv \{\mathbf{a} \cup \mathbf{b} | \mathbf{a} \in A, \mathbf{b} \in B\} \end{aligned}$$

Intuitively, the first operation is a collection of trees, and the second lists all combinations of trees in A and B. The above operations will be used in the proof of a dynamic programming algorithm given in Section 4.3. It is trivial that they satisfy commutative, associative, and distributive laws.

$$\begin{aligned} A \oplus B &= B \oplus A \\ A \otimes B &= B \otimes A \\ A \oplus (B \oplus C) &= (A \oplus B) \oplus C \\ A \otimes (B \otimes C) &= (A \otimes B) \otimes C \\ A \otimes (B \oplus C) &= (A \otimes B) \oplus (A \otimes C) \end{aligned}$$

We denote a set of unpacked trees rooted at node $n \in C \cup D$ as $\Omega(n)$. $\Omega(n)$ is defined recursively. For terminal node $c \in C$, obviously $\Omega(c) = \{\{c\}\}$. For internal conjunctive node $c \in C$, an unpacked tree is a combination of trees each of which is selected from a disjunctive daughter. Hence, a set of all unpacked trees is represented as a product of trees from disjunctive daughters.

$$\Omega(c) = \{\{c\}\} \otimes \bigotimes_{d \in \delta(c)} \Omega(d).$$

Disjunctive node $d \in D$ represents alternatives of packed trees, and obviously a set of unpacked trees is represented as a union of daughter trees, i.e., $\Omega(d) = \bigoplus_{c \in \gamma(d)} \Omega(c)$.

To summarize, a set of unpacked trees is defined formally as follows.

Definition 4.3 (Unpacked tree) *Given feature forest $\Phi = \langle C, D, r, \gamma, \delta \rangle$, a set, $\Omega(n)$ of unpacked trees rooted at node $n \in C \cup D$ is defined recursively as follows.*

- *If $n \in C$ is a terminal, i.e., $\delta(n) = \emptyset$,*

$$\Omega(n) \equiv \{\{n\}\}.$$

[2] In fact, no feature forests include equivalent unpacked trees if every disjunctive node has no identical daughter nodes. Thus we may define a set of unpacked trees as an ordinary set, although the details are omitted here for simplicity.

$$\Omega(\Phi) = \{ \quad \{c_1, c_2, c_4, c_6\}, \quad \{c_1, c_3, c_4, c_6\}, \quad \{c_1, c_2, c_5, c_6\}, \quad \{c_1, c_3, c_5, c_6\}, \\ \{c_1, c_2, c_4, c_7\}, \quad \{c_1, c_3, c_4, c_7\}, \quad \{c_1, c_2, c_5, c_7\}, \quad \{c_1, c_3, c_5, c_7\} \quad \}$$

Figure 4.11: Unpacked trees represented as sets of conjunctive nodes

- *If* $n \in C$,

$$\Omega(n) \equiv \{\{n\}\} \otimes \bigotimes_{d \in \delta(n)} \Omega(d).$$

- *If* $n \in D$,

$$\Omega(n) \equiv \bigoplus_{c \in \gamma(n)} \Omega(c).$$

Feature forests are directed acyclic graphs and, as such, the above definition does not include a loop. Hence, $\Omega(n)$ is properly defined.

A set of all unpacked trees is then represented by $\Omega(r)$; henceforth, we denote $\Omega(r)$ as $\Omega(\Phi)$, or just Ω when it is not confusing in context. Figure 4.11 shows $\Omega(\Phi)$ of the feature forest in Figure 4.9. Following the definition, the first element of each set is the root node, c_1, and the rest are elements of the product of $\{c_2, c_3\}$, $\{c_4, c_5\}$, and $\{c_6, c_7\}$. Each set in Figure 4.11 corresponds to a tree in Figure 4.10.

Given the above formalization, a feature function for an unpacked tree is defined as follows.

Definition 4.4 (Feature function for unpacked tree) *Feature function f_i for an unpacked tree, $\mathbf{c} \in \Omega(\Phi)$ is defined as:*

$$f_i(\mathbf{c}) = \sum_{c \in \mathbf{c}} f_i(c).$$

Once a feature function for an unpacked tree is given, a model expectation is defined as in the traditional model described in Section 2.4.

Definition 4.5 (Model expectation of feature forest) *Model expectation μ_i for feature forests $\{\Phi(x)\}$ is defined as:*

$$\begin{aligned} \mu_i &= \sum_{x \in \mathcal{X}} \tilde{p}(x) \sum_{\mathbf{c} \in \Omega(\Phi(x))} f_i(\mathbf{c}) p(\mathbf{c}|x) \\ &= \sum_{x \in \mathcal{X}} \tilde{p}(x) \sum_{\mathbf{c} \in \Omega(\Phi(x))} f_i(\mathbf{c}) \frac{1}{Z(x)} \exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right), \end{aligned}$$

$$\textit{where} \qquad Z(x) = \sum_{\mathbf{c} \in \Omega(\Phi(x))} \exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right).$$

It is evident that the naive computation of model expectations requires exponential time complexity since the number of unpacked trees (i.e., $|\Omega(\Phi)|$) is exponentially related to the number of nodes in the feature forest Φ. We therefore need an algorithm for computing model expectations without unpacking a feature forest.

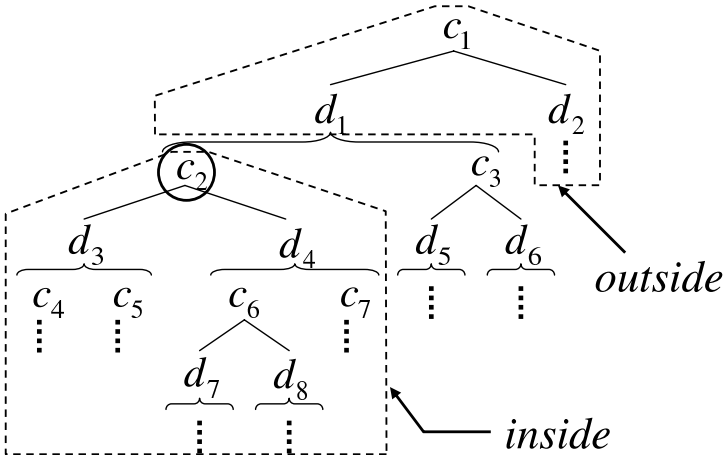


Figure 4.12: Inside/outside at node c_2 in a feature forest

4.2.2 Dynamic Programming

To efficiently compute model expectations, we incorporate an approach similar to the dynamic programming algorithm of computing inside/outside probabilities of PCFG. We first define the notion of inside/outside of a feature forest. Figure 4.12 illustrates this concept, which is similar to that of PCFG. Inside denotes a set of partial trees (sets of conjunctive nodes) derived from node c_2. Outside denotes a set of partial trees that derive node c_2. That is, outside trees are partial trees of complements of inside trees.

We denote a set of inside trees at node n as $\iota(n)$, and that of outside trees as $o(n)$. They are introduced informally here; a formal definition will be given in Section4.3.

An inside/outside α-product is then defined for each conjunctive/disjunctive node. The inside (or outside) α-products are the summation of $\exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right)$ of all inside (or outside) trees $\mathbf{c}$.

Definition 4.6 (Inside/outside α-product) *An inside α-product at conjunctive node $c \in C$ is*

$$\varphi_c = \sum_{\mathbf{c} \in \iota(c)} \exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right).$$

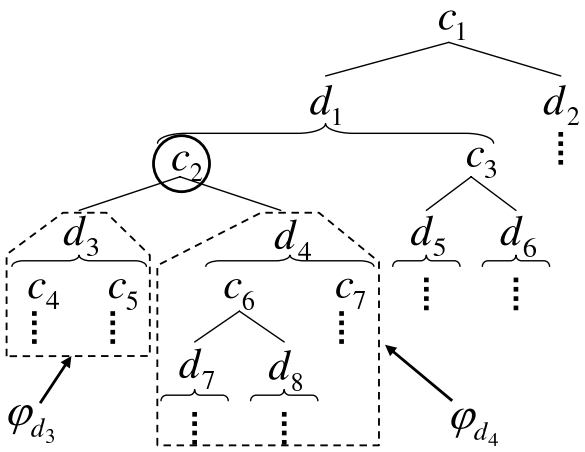
An outside α-product is

$$\psi_c = \sum_{\mathbf{c} \in o(c)} \exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right).$$

Similarly, inside/outside α-products at disjunctive node $d \in D$ are defined as follows:

$$\varphi_d = \sum_{\mathbf{c} \in \iota(d)} \exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right),$$

$$\psi_d = \sum_{\mathbf{c} \in o(d)} \exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right).$$



$$\varphi_{c_2} = \varphi_{d_3}\varphi_{d_4} \exp\left(\sum_j \lambda_j f_j(c_2)\right)$$

Figure 4.13: Incremental computation of inside α-products at conjunctive node c_2

We can derive that model expectations of a feature forest are computed as the product of the inside and outside α-products.

Theorem 4.1 (Model expectation of feature forest) *Model expectation μ_i of feature forest $\Phi(x) = \langle C_x, D_x, r_x, \gamma_x, \delta_x \rangle$ is computed as the product of inside and outside α-products as follows:*

$$\mu_i = \sum_{x \in \mathcal{X}} \tilde{p}(x) \frac{1}{Z(x)} \sum_{c \in C_x} f_i(c) \varphi_c \psi_c,$$

$$\textit{where} \qquad Z(x) = \varphi_{r_x}.$$

The proof will be given in Section 4.3.

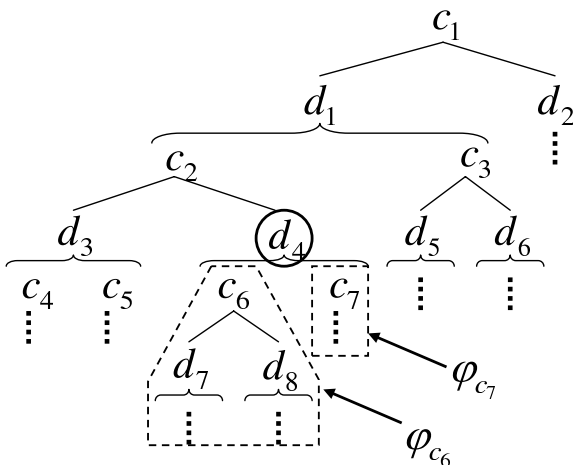
This equation shows a method for efficiently computing model expectations by traversing conjunctive nodes without unpacking the forest, if the inside/outside α-products are given. The remaining issue is how to efficiently compute inside/outside α-products.

Fortunately, inside/outside α-products can be incrementally computed by dynamic programming without unpacking feature forests. Figure 4.13 shows the process of computing the inside α-product at a conjunctive node from the inside α-products of its daughter nodes. Since the inside of a conjunctive node is a set of the combinations of all of its descendants, the α-product is computed by multiplying the α-products of the daughter trees. The following equation is derived.

$$\varphi_c = \left(\prod_{d \in \delta(c)} \varphi_d\right) \exp\left(\sum_j \lambda_j f_j(c)\right).$$

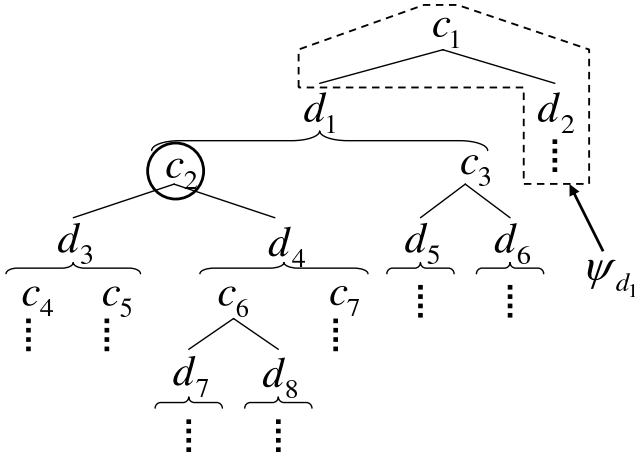
The inside of a disjunctive node is the collection of the inside trees of its daughter nodes. Hence, the inside α-product at disjunctive node $d \in D$ is computed as follows (Figure 4.14).

$$\varphi_d = \sum_{c \in \gamma(d)} \varphi_c.$$



$$\varphi_{d_4} = \varphi_{c_6} + \varphi_{c_7}$$

Figure 4.14: Incremental computation of inside α-products at disjunctive node d_4



$$\psi_{c_2} = \psi_{d_1}$$

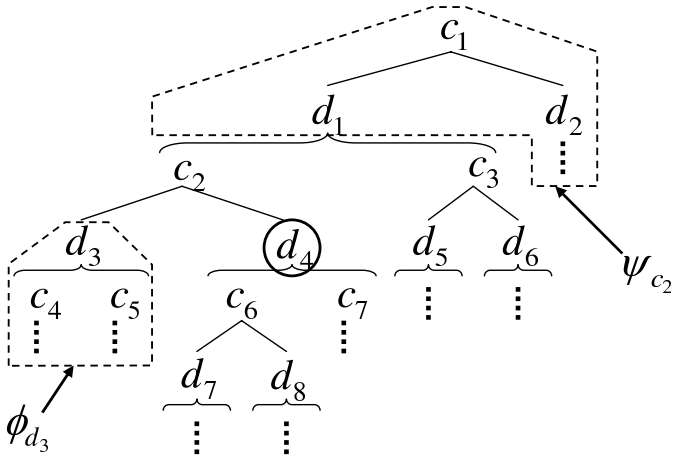
Figure 4.15: Incremental computation of outside α-products at conjunctive node c_2

Theorem 4.2 (Inside α-product) *The inside α-product φ_c at conjunctive node c is computed by the following equation if φ_d is given for all daughter disjunctive nodes $d \in \delta(c)$.*

$$\varphi_c = \left(\prod_{d \in \delta(c)} \varphi_d \right) \exp\left(\sum_j \lambda_j f_j(c) \right).$$

The inside α-product φ_d at disjunctive node d is computed by the following equation if φ_c is given for all daughter conjunctive nodes $c \in \gamma(d)$.

$$\varphi_d = \sum_{c \in \gamma(d)} \varphi_c.$$



$$\psi_{d_4} = \psi_{c_2} \varphi_{d_3} \exp\left(\sum_j \lambda_j f_j(c_2) \right)$$

Figure 4.16: Incremental computation of outside α-products at disjunctive node d_4

The outside of a disjunctive node is equivalent to the outside of its daughter nodes. Hence, the outside α-product of a disjunctive node is propagated to its daughter conjunctive nodes (Figure4.15).

$$\psi_c = \sum_{\{d|c \in \gamma(d)\}} \psi_d.$$

The computation of an outside α-product of a disjunctive node is somewhat complicated. As shown in Figure 4.16, the outside trees of a disjunctive node are all combinations of

- the outside trees of the mother nodes, and
- the inside trees of the sister nodes.

From this, we find:

$$\psi_d = \sum_{\{c|d \in \delta(c)\}} \left\{ \psi_c \exp\left(\sum_j \lambda_j f_j(c) \right) \prod_{\substack{d' \in \delta(c) \\ d' \neq d}} \varphi_{d'} \right\}.$$

We finally find the following theorem of the computation of outside α-products.

Theorem 4.3 (Outside α-product) *The outside α-product ψ_c at conjunctive node c is computed by the following equation if ψ_d is given for all mother disjunctive nodes, i.e., d such that $c \in \gamma(d)$.*

$$\psi_c = \sum_{\{d|c \in \gamma(d)\}} \psi_d.$$

The outside α-product ψ_d at disjunctive node d is computed by the following equation if ψ_c is given for all mother conjunctive nodes, i.e., c such that $d \in \delta(c)$, and $\varphi_{d'}$ for all sibling disjunctive nodes

Input: training data $E = \{\langle x_i, y_i \rangle\}$, feature forests $\{\Phi(x_i)\}$
feature functions $\boldsymbol{f} = \{f_i\}$, initial parameters $\boldsymbol{\lambda} = \{\lambda_i\}$
Output: optimal parameters $\boldsymbol{\lambda}$

foreach $\langle x, y \rangle \in E$
  $\{\varphi\} \leftarrow$ inside_product$(\Phi(x))$
  $\{\psi\} \leftarrow$ outside_product$(\Phi(x))$
  foreach $c \in C$
    foreach $f_i \in \boldsymbol{f}$ such that $f_i(c) \neq 0$
      $\mu_i \leftarrow \mu_i + f_i \varphi_c \psi_c$
    end_foreach
  end_foreach
  foreach $f_i \in \boldsymbol{f}$
    $\mu_i \leftarrow \frac{\mu_i}{\varphi_r}$
  end_foreach
end_foreach

function inside_product$(\Phi(x) = \langle C, D, r, \gamma, \delta \rangle)$
  foreach $c \in C$, $d \in D$

$$\varphi_c \leftarrow \prod_{d' \in \delta(c)} \varphi_{d'} \exp\left(\sum_j \lambda_j f_j(c)\right)$$

$$\varphi_d \leftarrow \sum_{c' \in \gamma(d)} \varphi_{c'}$$

  end_foreach
  return $\{\varphi\}$

function outside_product$(\Phi(x) = \langle C, D, r, \gamma, \delta \rangle)$
  foreach $c \in C$, $d \in D$

$$\psi_c \leftarrow \sum_{\{d' | c \in \gamma(d')\}} \psi_{d'}$$

$$\psi_d \leftarrow \sum_{\{c' | d \in \delta(c')\}} \left\{ \psi_{c'} \exp\left(\sum_j \lambda_j f_j(c')\right) \prod_{\substack{d' \in \delta(c') \\ d' \neq d}} \varphi_{d'} \right\}$$

  end_foreach
  return $\{\psi\}$

Figure 4.17: Algorithm of computing model expectations of feature forests

d'.

$$\psi_d = \sum_{\{c|d\in\delta(c)\}} \left\{ \psi_c \exp\left(\sum_j \lambda_j f_j(c)\right) \prod_{\substack{d'\in\delta(c)\\ d'\neq d}} \varphi_{d'} \right\}.$$

Figure 4.17 shows the overall algorithm for estimating the parameters, given a set of feature forests. The key point of the algorithm is to compute inside α-products φ and outside α-products ψ for each node in C, and not for all unpacked trees. The functions `inside_product` and `outside_product` compute φ and ψ efficiently by dynamic programming. Note that the order of traversing nodes is important for incremental computation, although it is not shown in Figure4.17. The computation for the daughter nodes and mother nodes must be completed before computing the inside and outside α-products, respectively. This constraint is easily solved using a topological sort.

The complexity of this algorithm is $O((|\tilde{C}| + |\tilde{D}|)|\tilde{F}||E|)$, where $|\tilde{C}|$ and $|\tilde{D}|$ are the average numbers of conjunctive and disjunctive nodes, respectively. This is tractable when $|\tilde{C}|$ and $|\tilde{D}|$ are of a reasonable size. As noted in this section, the number of nodes in a feature forest is usually polynomial even when that of the unpacked trees is exponential. Thus we can efficiently compute model expectations with polynomial computational complexity.

4.3 Proof

This section provides the proof of Theorem 4.1, 4.2, and 4.3. We first define inside/outside trees formally.

Definition 4.7 (Inside trees) *We define a set $\iota(n)$ of inside trees rooted at node $n \in C \cup D$ as a set of unpacked trees rooted at n.*

$$\iota(n) \equiv \Omega(n).$$

Definition 4.8 (Outside trees) *We define a set $o(n)$ of outside trees rooted at node $n \in C \cup D$ as follows.*

$$o(r) \equiv \{\emptyset\}$$

$$o(c) \equiv \bigoplus_{d\in\gamma^{-1}(c)} o(d)$$

$$o(d) \equiv \bigoplus_{c\in\delta^{-1}(d)} \left\{ \{\{c\}\} \otimes o(c) \otimes \bigotimes_{d'\in\delta(c), d'\neq d} \iota(d') \right\}.$$

In the definition, γ^{-1} and δ^{-1} denote mothers of conjunctive and disjunctive nodes, respectively. Formally,

$$\begin{aligned} \gamma^{-1}(c) &\equiv \{d|c \in \gamma(d)\} \\ \delta^{-1}(d) &\equiv \{c|d \in \delta(c)\} \end{aligned}$$

The following function is defined to represent a set of trees that include a given node, $n \in C \cup D$.

Definition 4.9 (Filter function) *We define function* $\Theta_c(\mathcal{C}) : 2^C \mapsto 2^C$ *as follows:*

$$\Theta_c(\mathcal{C}) \equiv \{\mathbf{c} \in \mathcal{C} | c \in \mathbf{c}\}.$$

For $d \in D$*, we define:*

$$\Theta_d(\mathcal{C}) \equiv \bigoplus_{c \in \gamma(d)} \Theta_c(\mathcal{C}).$$

Given the above definition of inside/outside trees, we derive the following lemma.

Lemma 4.1 (Inside/outside trees) *For any* $n \in C \cup D$*,*

$$\iota(n) \otimes o(n) = \Theta_n(\Omega).$$

Proof) We prove the lemma with a mathematical induction. For the root node r*,*

$$\begin{aligned} \iota(r) \otimes o(r) &= \Omega \otimes \{\emptyset\} \\ &= \Omega. \end{aligned}$$

Since $\forall \mathbf{c} \in \Omega . r \in \mathbf{c}$*,*

$$\Omega = \Theta_r(\Omega).$$

For disjunctive node $d \in D$*,*

$$\begin{aligned} & \iota(d) \otimes o(d) \\ = \; & \iota(d) \otimes \bigoplus_{c \in \delta^{-1}(d)} \left\{ \{\{c\}\} \otimes o(c) \otimes \bigotimes_{d' \in \delta(c), d' \neq d} \iota(d') \right\} \\ = \; & \bigoplus_{c \in \delta^{-1}(d)} \left\{ \iota(d) \otimes \{\{c\}\} \otimes o(c) \otimes \bigotimes_{d' \in \delta(c), d' \neq d} \iota(d') \right\} \\ = \; & \bigoplus_{c \in \delta^{-1}(d)} \left\{ \{\{c\}\} \otimes o(c) \otimes \bigotimes_{d' \in \delta(c)} \iota(d') \right\} \\ = \; & \bigoplus_{c \in \delta^{-1}(d)} \{o(c) \otimes \iota(c)\} \\ & \textit{(by Definition 4.7)} \\ = \; & \bigoplus_{c \in \delta^{-1}(d)} \Theta_c(\Omega). \\ & \textit{(by the assumption of the induction)} \end{aligned}$$

From Definition 4.7, $\forall c . c \in \delta^{-1}(d) . \forall \mathbf{c} \in \Omega . ((c) \in \Theta_c(\Omega) \rightarrow \mathbf{c} \in \Theta_d(\Omega)$*. Hence,*

$$\Theta_c(\Omega) \subseteq \Theta_d(\Omega) \Leftrightarrow \bigoplus_{c \in \delta^{-1}(d)} \Theta_c(\Omega) \subseteq \Theta_d(\Omega).$$

Meanwhile, $\forall \mathbf{c} \in \Theta_d(\Omega) . \exists c \in \delta^{-1}(d) . c \in \mathbf{c}$*. From this,*

$$\Theta_d(\Omega) \subseteq \bigoplus_{c \in \delta^{-1}(d)} \Theta_c(\Omega).$$

Namely,

$$\bigoplus_{c\in\delta^{-1}(d)} \Theta_c(\Omega) \quad = \quad \Theta_d(\Omega).$$

For conjunctive node $c \in C$,

$$\begin{aligned}
& \iota(c) \otimes o(c) \\
= \; & \iota(c) \otimes \bigoplus_{d\in\gamma^{-1}(c)} o(d) \\
= \; & \bigoplus_{d\in\gamma^{-1}(c)} \iota(c) \otimes o(d) \\
= \; & \bigoplus_{d\in\gamma^{-1}(c)} \Theta_c(\iota(d)) \otimes o(d) \\
= \; & \bigoplus_{d\in\gamma^{-1}(c)} \Theta_c(\iota(d) \otimes o(d)) \\
= \; & \bigoplus_{d\in\gamma^{-1}(c)} \Theta_c(\Theta_d(\Omega)) \\
& \textit{(by the assumption of the induction)} \\
= \; & \Theta_c(\Omega).
\end{aligned}$$

q.e.d.

The following lemma is extensively used in the proof to transform equations.

Lemma 4.2 (Distributive law) *For any function $g(\mathbf{x})$ that satisfies $g(\mathbf{x}_1)g(\mathbf{x}_2) = g(\mathbf{x}_1 \cup \mathbf{x}_2)$,*

$$\left\{ \sum_{\mathbf{x}_1\in X_1} g(\mathbf{x}_1) \right\} \left\{ \sum_{\mathbf{x}_2\in X_2} g(\mathbf{x}_2) \right\} = \sum_{\mathbf{x}\in X_1\otimes X_2} g(\mathbf{x}).$$

This equation may further be generalized as follows:

$$\prod_{X\in\mathcal{X}} \sum_{\mathbf{x}\in X} g(\mathbf{x}) = \sum_{\mathbf{x}\in\bigotimes_{X\in\mathcal{X}} X} g(\mathbf{x}).$$

Proof) Since the second equation is easily derived from the first, we give a proof of the first. For some $\mathbf{x} \in X_2$,

$$\begin{aligned}
& \left\{ \sum_{\mathbf{x}_1\in X_1} g(\mathbf{x}_1) \right\} \left\{ \sum_{\mathbf{x}_2\in X_2} g(\mathbf{x}_2) \right\} \\
= \; & \left\{ \sum_{\mathbf{x}_1\in X_1} g(\mathbf{x}_1) \right\} g(\mathbf{x}) \left\{ \sum_{\mathbf{x_2}\in X_2\setminus\{\mathbf{x}\}} g(\mathbf{x_2}) \right\} \\
= \; & \left\{ \sum_{\mathbf{x}_1\in X_1\otimes\{\mathbf{x}\}} g(\mathbf{x}_1) \right\} \left\{ \sum_{\mathbf{x_2}\in X_2\setminus\{\mathbf{x}\}} g(\mathbf{x_2}) \right\}.
\end{aligned}$$

The recursive application of the above transformation gives the lemma.

Since $\exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right)$ satisfies the condition above, we derive:

$$\prod_{C\in\mathcal{C}}\sum_{\mathbf{c}\in C}\exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right) = \sum_{\mathbf{c}\in\bigotimes_{C\in\mathcal{C}} C}\exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right).$$

For disjoint union,

$$\sum_{X\in\mathcal{X}}\sum_{\mathbf{x}\in X} g(\mathbf{x}) = \sum_{\mathbf{x}\in\bigoplus_{X\in\mathcal{X}} X} g(\mathbf{x}).$$

We then start the proof of the three theorems.

Proof of Theorem 4.1

$$\begin{aligned}
\varphi_c\psi_c &= \left\{\sum_{\mathbf{c_1}\in\iota(c)}\exp\left(\sum_j \lambda_j f_j(\mathbf{c_1})\right)\right\}\left\{\sum_{\mathbf{c_2}\in o(c)}\exp\left(\sum_j \lambda_j f_j(\mathbf{c_2})\right)\right\} \\
&= \sum_{\mathbf{c}\in\iota(c)\otimes o(c)}\exp\left(\sum_j \lambda_j f_j\,(\mathbf{c})\right) \\
&\quad \text{(by Lemma 4.2)} \\
&= \sum_{\mathbf{c}\in\Theta_c(\Omega)}\exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right). \\
&\quad \text{(by Lemma 4.1)}
\end{aligned}$$

From this, we find

$$\begin{aligned}
\mu_i &= \sum_{x\in\mathcal{X}}\tilde{p}(x)\frac{1}{Z(x)}\sum_{c\in C} f_i(c)\varphi_c\psi_c \\
&= \sum_{x\in\mathcal{X}}\tilde{p}(x)\frac{1}{Z(x)}\sum_{c\in C} f_i(c)\left\{\sum_{\mathbf{c}\in\Theta_c(\Omega(\Phi(x)))}\exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right)\right\} \\
&= \sum_{x\in\mathcal{X}}\tilde{p}(x)\frac{1}{Z(x)}\sum_{c\in C}\sum_{\mathbf{c}\in\Theta_c(\Omega(\Phi(x)))} f_i(c)\exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right).
\end{aligned}$$

The summation in the last formula is transformed as follows.

$$
\begin{aligned}
& \sum_{c\in C}\sum_{\mathbf{c}\in\Theta_c(\Omega)} f_i(c)\exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right) \\
= & \sum_{(c,\mathbf{c})\in\{(c,\mathbf{c})|c\in C.\mathbf{c}\in\Omega.c\in\mathbf{c}\}} f_i(c)\exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right) \\
& \text{(by Definition 4.9)} \\
= & \sum_{(c,\mathbf{c})\in\{(c,\mathbf{c})|\mathbf{c}\in\Omega.c\in\mathbf{c}\}} f_i(c)\exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right) \\
& \text{(since } \forall c\in\mathbf{c}.c\in C\text{)} \\
= & \sum_{\mathbf{c}\in\Omega}\sum_{c\in\mathbf{c}} f_i(c)\exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right). \qquad (4.2)
\end{aligned}
$$

Hence, we finally obtain:

$$
\begin{aligned}
\mu_i &= \sum_{x\in\mathcal{X}} \tilde{p}(x)\frac{1}{Z(x)} \sum_{\mathbf{c}\in\Omega(\Phi(x))}\sum_{c\in\mathbf{c}} f_i(c)\exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right) \\
& \text{(by Equation 4.2)} \\
&= \sum_{x\in\mathcal{X}} \tilde{p}(x)\frac{1}{Z(x)} \sum_{\mathbf{c}\in\Omega(\Phi(x))} \left\{\sum_{c\in\mathbf{c}} f_i(c)\right\}\exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right) \\
&= \sum_{x\in\mathcal{X}} \tilde{p}(x)\frac{1}{Z(x)} \sum_{\mathbf{c}\in\Omega(\Phi(x))} f_i(\mathbf{c})\exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right) \\
& \text{(by Definition 4.4)} \\
&= \sum_{x\in\mathcal{X}} \tilde{p}(x) \sum_{\mathbf{c}\in\Omega(\Phi(x))} f_i(\mathbf{c})\frac{1}{Z(x)}\exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right).
\end{aligned}
$$

The last equation is equal to Definition 4.5. $Z(x)$ is obtained as φ_r as follows.

$$
\begin{aligned}
Z(x) &= \varphi_r \\
&= \sum_{\mathbf{c}\in\iota(r)} \exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right) \\
&= \sum_{\mathbf{c}\in\Omega(\Phi(x))} \exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right).
\end{aligned}
$$

q.e.d.

Proof of Theorem 4.2 The inside α-product at conjunctive node c is:

$$\begin{aligned}
\varphi_c &= \left(\prod_{d\in\delta(c)} \varphi_d\right) \exp\left(\sum_j \lambda_j f_j(c)\right) \\
&= \left\{\prod_{d\in\delta(c)} \sum_{\mathbf{c}\in\iota(d)} \exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right)\right\} \exp\left(\sum_j \lambda_j f_j(c)\right) \\
&\quad \text{(by Definition 4.6)} \\
&= \left\{\sum_{\mathbf{c}\in\bigotimes_{d\in\delta(c)}\iota(d)} \exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right)\right\} \exp\left(\sum_j \lambda_j f_j(c)\right) \\
&\quad \text{(by Lemma 4.2)} \\
&= \sum_{\mathbf{c}\in\{\{c\}\}\otimes\bigotimes_{d\in\delta(c)}\iota(d)} \exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right) \\
&\quad \text{(by Lemma 4.2)} \\
&= \sum_{\mathbf{c}\in\iota(c)} \exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right). \\
&\quad \text{(by Definition 4.7)}
\end{aligned}$$

The inside α-product at disjunctive node d is:

$$\begin{aligned}
\varphi_d &= \sum_{c\in\gamma(d)} \varphi_c \\
&= \sum_{c\in\gamma(d)} \left\{\sum_{\mathbf{c}\in\iota(c)} \exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right)\right\} \\
&\quad \text{(by Definition 4.6)} \\
&= \sum_{\mathbf{c}\in\bigoplus_{c\in\gamma(d)}\iota(c)} \exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right) \\
&= \sum_{\mathbf{c}\in\iota(d)} \exp\left(\sum_j \lambda_j f_j(\mathbf{c})\right). \\
&\quad \text{(by Definition 4.7)}
\end{aligned}$$

q.e.d.

Proof of Theorem 4.3 The outside α-product at conjunctive node c is:

$$\begin{aligned}
\psi_c &= \sum_{d \in \gamma^{-1}(c)} \psi_d \\
&= \sum_{d \in \gamma^{-1}(c)} \sum_{\mathbf{c} \in o(d)} \exp\left(\sum_j \lambda_j f_j(\mathbf{c}) \right) \\
&\quad \text{(by Definition 4.6)} \\
&= \sum_{\mathbf{c} \in \bigoplus_{d \in \gamma^{-1}(c)} o(d)} \exp\left(\sum_j \lambda_j f_j(\mathbf{c}) \right) \\
&= \sum_{\mathbf{c} \in o(c)} \exp\left(\sum_j \lambda_j f_j(\mathbf{c}) \right). \\
&\quad \text{(by Definition 4.8)}
\end{aligned}$$

The outside α-product at disjunctive node d is:

$$\begin{aligned}
\psi_d &= \sum_{c \in \delta^{-1}(d)} \exp\left(\sum_j \lambda_j f_j(c) \right) \psi_c \prod_{d' \in \delta(c), d' \neq d} \varphi_{d'} \\
&= \sum_{c \in \delta^{-1}(d)} \exp\left(\sum_j \lambda_j f_j(c) \right) \left\{ \sum_{\mathbf{c} \in o(c)} \exp\left(\sum_j \lambda_j f_j(\mathbf{c}) \right) \right\} \\
&\qquad \prod_{d' \in \delta(c), d' \neq d} \sum_{\mathbf{c} \in \iota(d')} \exp\left(\sum_j \lambda_j f_j(\mathbf{c}) \right) \\
&\quad \text{(by Definition 4.6)} \\
&= \sum_{c \in \delta^{-1}(d)} \exp\left(\sum_j \lambda_j f_j(c) \right) \left\{ \sum_{\mathbf{c} \in o(c)} \exp\left(\sum_j \lambda_j f_j(\mathbf{c}) \right) \right\} \\
&\qquad \left\{ \sum_{\mathbf{c} \in \bigotimes_{d' \in \delta(c), d' \neq d} \iota(d')} \exp\left(\sum_j \lambda_j f_j(\mathbf{c}) \right) \right\} \\
&\quad \text{(by Lemma 4.2)} \\
&= \sum_{c \in \delta^{-1}(d)} \exp\left(\sum_j \lambda_j f_j(c) \right) \left\{ \sum_{\mathbf{c} \in o(c) \otimes \bigotimes_{d' \in \delta(c), d' \neq d} \iota(d')} \exp\left(\sum_j \lambda_j f_j(\mathbf{c}) \right) \right\} \\
&\quad \text{(by Lemma 4.2)} \\
&= \sum_{c \in \delta^{-1}(d)} \sum_{\mathbf{c} \in \{\{c\}\} \otimes o(c) \otimes \bigotimes_{d' \in \delta(c), d' \neq d} \iota(d')} \exp\left(\sum_j \lambda_j f_j(\mathbf{c}) \right) \\
&\quad \text{(by Lemma 4.2)}
\end{aligned}$$

$$= \sum_{\mathbf{c} \in \bigoplus_{c \in \delta^{-1}(c)} \left\{ \{\{c\}\} \otimes o(c) \otimes \bigotimes_{d' \in \delta(c), d' \neq d} \iota(d') \right\}} \exp\left(\sum_j \lambda_j f_j(\mathbf{c}) \right)$$

$$= \sum_{\mathbf{c} \in o(d)} \exp\left(\sum_j \lambda_j f_j(\mathbf{c}) \right).$$

(by Definition 4.8)

q.e.d.

4.4 Discussion and Related Work

A new algorithm was presented for maximum entropy modeling and shown to provide a solution to the parameter estimation of probabilistic models of complete structures without independence assumption. We first defined feature forest, which is a packed representation of an exponential number of trees of features. When training data is represented with feature forests, model parameters are estimated at a tractable cost without unpacking the forests. The method provides a more flexible modeling scheme than previous methods of application of maximum entropy models to natural language processing. Furthermore, it is applicable to complex data structures where an event is difficult to decompose into independent sub-events. For example, as presented in Chapter 5, feature forest models are applicable to probabilistic modeling of feature structure grammars including HPSG.

The model described in this chapter was first published in Miyao and Tsujii (2002), and has been applied to probabilistic models for parsing with lexicalized grammars. Applications to CCG parsing (Clark and Curran, 2003, 2004b) and LFG parsing (Kaplan et al., 2004) demonstrated that feature forest models attained higher accuracy than other models. We will describe application of feature forest models to probabilistic HPSG parsing in Chapter 5, and demonstrate the effectiveness in a series of experiments in Chapter 6.

The work of Geman and Johnson (2002) independently developed a dynamic programming algorithm for maximum entropy models. The solution was similar to our approach, although their method was designed to traverse LFG parse results represented with disjunctive feature structures proposed by Maxwell III and Kaplan (1995). The difference between the two approaches is that feature forests have a simpler generic data structure to represent packed forest structures. Therefore, without assuming what feature forests represent, our algorithm can be applied to various tasks, including theirs.

Another approach to the probabilistic modeling of complete structures is a method of approximation. The work on whole sentence maximum entropy models (Chen and Rosenfeld, 1999b; Rosenfeld, 1997) proposed an approximation algorithm to estimate parameters of maximum entropy models on whole sentence structures. However, the algorithm suffered from slow convergence, and the model was basically a sequence model. Therefore, it could not produce a solution for complex structures as our model can. Osborne (2000) proposed another approximation method, which extracts *informative samples* from all parse candidates. The method was successfully applied to Dutch HPSG parsing (Malouf and van Noord, 2004). This issue will be discussed in Section 5.6.

We should also mention Conditional Random Fields (CRFs) (Lafferty et al., 2001) for solving a similar problem in the context of maximum entropy Markov models. Their solution was an algorithm similar to the computation of forward/backward probabilities of hidden Markov models (HMMs). Their algorithm is a special case of our algorithm in which each conjunctive node has only one daughter. This is obvious because feature forests can represent Markov chains. In an analogy, CRFs correspond to HMMs, while feature forest models correspond to PCFGs. Extensions of

CRFs, such as semi-Markov CRFs (Sarawagi and Cohen, 2004), are also regarded as instances of feature forest models. This fact implies that our algorithm is applicable to not only parsing but also to other tasks. CRFs are now widely used for sequence-based tasks, such as part-of-speech tagging and named entity recognition, and have been shown to achieve the best performance in various tasks (McCallum and Li, 2003; McCallum et al., 2004; Peng and McCallum, 2004; Pinto et al., 2003; Roark et al., 2004; Settles, 2004; Sha and Pereira, 2003; Sutton et al., 2004). The results imply that the method proposed in this chapter will achieve high accuracy when applied to various statistical models with tree structures. Dynamic CRFs (McCallum et al., 2004; Sutton et al., 2004) provide us with an interesting inspiration for extending feature forest models. The purpose of dynamic CRFs is to incorporate feature functions that are not represented locally, and the solution is to apply a variational method, which is an algorithm of numerical computation, to obtain approximate solutions. A similar method may be developed to overcome a bottleneck of feature forest models, i.e., feature functions are localized to conjunctive nodes.

The structure of feature forests is common in natural language processing and computational linguistics. As is easily seen, lattices, Markov chains, and CFG parse trees are represented by feature forests. Furthermore, since conjunctive nodes do not necessarily represent CFG nodes/rules and terminals of feature forests need not be words, feature forests can express any forest structures in which ambiguities are packed in local structures. Examples include derivation trees of LTAG and CCG. Chiang (2003) proved that feature forests were considered as derivation forests of *linear context-free rewriting systems* (LCFRSs) (Vijay-Shanker et al., 1987; Weir, 1988). LCFRSs define a wide variety of grammars, including LTAG and CCG, while preserving polynomial-time complexity of parsing. This demonstrates that feature forest models are applicable to probabilistic models far beyond PCFGs. Feature forests are also isomorphic to *support graphs* (or *explanation graphs*) used in the graphical EM algorithm (Kameya and Sato, 2000). In their framework, a program in a logic programming language, PRISM (Sato and Kameya, 1997), is converted into support graphs, and parameters of probabilistic models are automatically learned by an EM algorithm. Support graphs have been proved to represent various statistical structural models, including HMMs, PCFGs, Bayesian networks and many other graphical structures (Sato, 2005; Sato and Kameya, 2001). Taken together, these results imply the high applicability of feature forest models to various real tasks.

Since feature forests have a structure isomorphic to parse forests of PCFG, it might be asserted that they can represent only immediate dominance relations of CFG rules as in PCFG, and result in only a slight, trivial extension of PCFG. As described above, however, feature forests can represent structures beyond CFG parse trees. Furthermore, since feature forests are generalized representation of ambiguous structures, each node in feature forests need not correspond to a node in PCFG parse forests. That is, a node in feature forests may represent any linguistic entity, including a fragment of syntactic structures, semantic relations, and other sentence-level information. In Chapter 5, we will see that HPSG parse trees and predicate argument relations including nonlocal dependencies are represented with tractable-size feature forests.

The idea of feature forest models could be applied to non-probabilistic machine learning methods. Taskar et al. (2004) proposed a dynamic programming algorithm of the learning of large-margin classifiers including support vector machines (Vapnik, 1995), and presented its application to the disambiguation in CFG parsing. Their algorithm resembles feature forest models; an optimization function is computed by a dynamic programing algorithm without unpacking packed forest structures. From the discussion in this chapter, it is evident that if the main part of an update formula is represented with (the exponential of) linear combinations, a method similar to feature forest models should be applicable.

The feature forest model provides new insight into the relationship between a linguistic structure and a unit of probabilities. Traditionally, a unit of probability was implicitly assumed to correspond to

a meaningful linguistic structure; a tagging to a word or an application of a rewriting rule. One reason for the assumption is to allow for dynamic programming algorithms such as the Viterbi algorithm as discussed in Section 4.1. A probability of a complete structure must be decomposed into atomic structures in which ambiguities are limited to a tractable size. Another reason is to estimate plausible probabilistic models. Since a probability is defined over atomic linguistic structures, they should also be meaningful so as to be assigned a probability. In feature forest models, however, conjunctive nodes are responsible for the former, while feature functions are responsible for the latter. That is, we are free from tying ambiguity packing of linguistic structures to probabilistic modeling. Conjunctive nodes may represent any fragments that are not linguistically meaningful. They should be designed to pack ambiguities and enable us to define useful features. Meanwhile, feature functions must represent linguistically meaningful entities to capture statistical regularity of the target problem. We expect the separation of a unit of probability from linguistic structures to open up a new framework of flexible probabilistic modeling.

Chapter 5

Probabilistic Modeling of Lexicalized Grammars

Lexicalized grammars have been studied extensively from both linguistic and computational points of view. However, despite research on processing efficiency (Oepen et al., 2002a), the application of lexicalized grammars remains limited to specific domains and short sentences (Oepen et al., 2002b; Toutanova and Manning, 2002). Scaling up parsing with lexicalized grammars to process real-world texts is an emerging research field with both theoretical and practical applications.

Statistical modeling of lexicalized grammars is indispensable for the practical application of syntactic analyzers. This is because applications usually require disambiguated or ranked parse results, and statistical modeling of preference is one of the most promising methods for disambiguation. We have developed a wide-coverage grammar and a large treebank of English HPSG in Chapter 3. A large treebank can be used as training and test data for statistical models. Therefore, we now have the basis for development and evaluation of statistical disambiguation models for wide-coverage HPSG parsing.

The aim of this chapter is to develop a maximum entropy model for disambiguation in wide-coverage HPSG parsing. As discussed in Chapter 4, maximum entropy models are necessary for the probabilistic modeling of feature structure grammars. Existing models of probabilistic parsing decompose the probability of a parse result into probabilities of rule applications assuming the independence of the probabilities (Charniak, 1997; Chiang, 2000; Clark et al., 2002; Collins, 1999, 1996, 1997; Hockenmaier and Steedman, 2002b; Magerman, 1995). Such models, however, cannot be applied to HPSG parsing, because rule applications in HPSG parsing violate the independence assumption and the resulting probabilistic model will be inconsistent (Abney, 1997). However, maximum entropy modeling of HPSG parsing is challenging because the estimation of maximum entropy models is computationally expensive, and we require solutions to make the model estimation tractable.

We have two difficulties in the development of maximum entropy models for HPSG parsing. One is that maximum entropy estimation requires enumeration of parse candidates assigned to a sentence by grammar. However, as discussed in Section 4.1, simple enumeration of parse trees is intractable because of exponential explosion. The solution in this chapter is the application of feature forest models proposed in Chapter 4. We propose methods of representing HPSG parse trees (Section 5.2) and predicate argument structures (Section 5.3) with feature forests (Miyao et al., 2003b; Miyao and Tsujii, 2003, 2005).

The other difficulty is that enumeration of parse candidates requires the parsing of all sentences in a training treebank. Despite extensive research on parsing efficiency, exhaustive HPSG parsing is still computationally expensive. The solution we adopted here is to filter out lexical entries ac-

cording to a preliminary probability distribution (Section 5.4). When parsing a treebank, we restrict the number of lexical entries assigned to words. Since ambiguity of lexical entries is a significant source of inefficiency in processing, such filtering of lexical entries greatly reduces the cost of parsing (Miyao and Tsujii, 2005).

This chapter also introduces feature functions employed in probabilistic HPSG parsing (Section 5.5). Feature functions are classified into two types: one captures preferences of syntax and the other represents semantic preference. In Chapter 6, we will investigate the contributions of features to the accuracy of parsing.

This chapter provides a theoretical framework for the probabilistic modeling of syntactic structures generated by lexicalized grammars. Since syntactic structures express various linguistic relations, such as non-local dependencies among words, a probabilistic model based on them is expected to attain high accuracy of parsing.

5.1 Probabilistic Models for HPSG Parsing

Discriminative log-linear models, or maximum entropy models, are now becoming a de facto standard of disambiguation models for parsing with lexicalized grammars (Clark and Curran, 2003, 2004b; Geman and Johnson, 2002; Johnson et al., 1999; Kaplan et al., 2004; Riezler et al., 2002, 2000). Previous studies on probabilistic models for HPSG (Baldridge and Osborne, 2003; Malouf and van Noord, 2004; Oepen et al., 2002b; Toutanova and Manning, 2002) have also adopted log-linear models. HPSG exploits feature structures to represent linguistic constraints. Such constraints are known to introduce inconsistencies in probabilistic models estimated using simple relative frequency as discussed in Section 4.1 (Abney, 1997). Maximum entropy models are required for credible probabilistic models and are also beneficial for incorporating various overlapping features.

This thesis follows previous studies on the probabilistic models for HPSG (Baldridge and Osborne, 2003; Malouf and van Noord, 2004; Oepen et al., 2002b; Toutanova and Manning, 2002). The probability, $p(t|\mathbf{w})$, of producing parse result t of a given sentence $\mathbf{w}$ is defined as

$$p(t|\mathbf{w}) = \frac{1}{Z_{\mathbf{w}}} p_0(t|\mathbf{w}) \exp\left(\sum_i f_i(t,\mathbf{w})\lambda_i(t,\mathbf{w})\right)$$

$$Z_{\mathbf{w}} = \sum_{t' \in T(\mathbf{w})} p_0(t'|\mathbf{w}) \exp\left(\sum_i f_i(t',\mathbf{w})\lambda_i(t',\mathbf{w})\right),$$

where $p_0(t|\mathbf{w})$ is a reference distribution (usually assumed to be a uniform distribution), and $T(\mathbf{w})$ is a set of parse candidates assigned to $\mathbf{w}$. The feature function $f_i(t,\mathbf{w})$ represents the characteristics of t and $\mathbf{w}$, while the corresponding model parameter $\lambda_i(t,\mathbf{w})$ is its weight. Model parameters that maximize the log-likelihood of the training data are computed using a numerical optimization method (Malouf, 2002).

Estimation of the above model requires a set of pairs $\langle t_{\mathbf{w}}, T(\mathbf{w})\rangle$, where $t_{\mathbf{w}}$ is the correct parse for sentence $\mathbf{w}$. While $t_{\mathbf{w}}$ is provided by a treebank, $T(\mathbf{w})$ is computed by parsing each $\mathbf{w}$ in the treebank. Previous studies assumed $T(\mathbf{w})$ could be enumerated; however, the assumption is impractical because the size of $T(\mathbf{w})$ is exponentially related to the length of $\mathbf{w}$. The problem of exponential explosion is inevitable in the wide-coverage parsing of real-world texts because many parse candidates are produced to treat various constructions in long sentences.

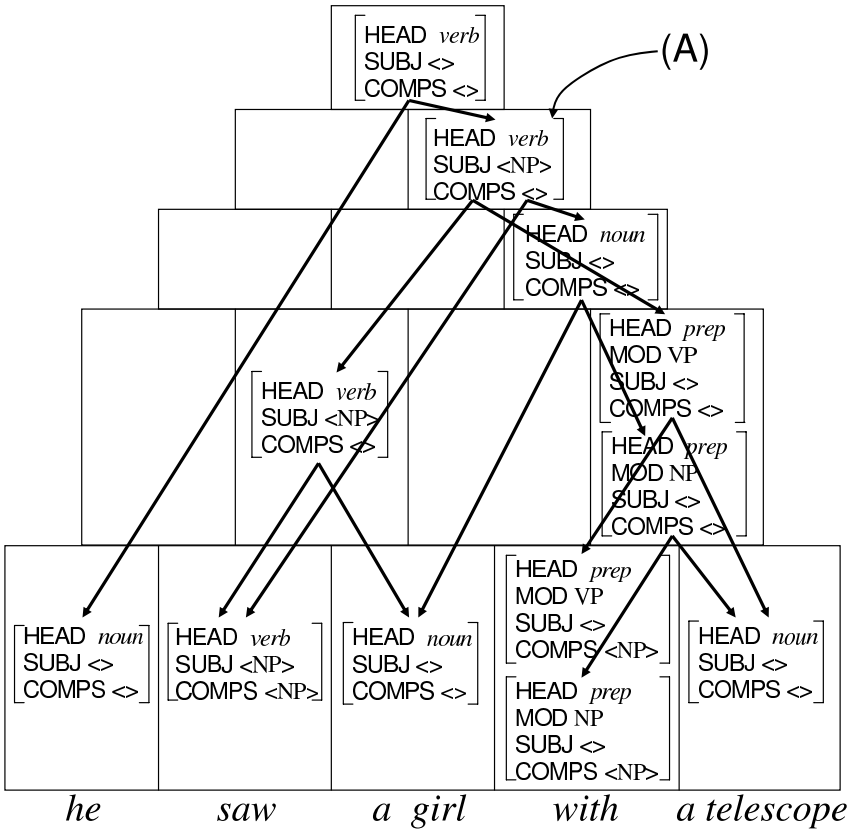


Figure 5.1: Chart for parsing *"he saw a girl with a telescope"*

5.2 Packed Representation of HPSG Parse Trees

While estimating discriminative log-linear models requires a set of $\langle t_{\mathbf{w}}, T(\mathbf{w})\rangle$, enumeration of $T(\mathbf{w})$ is intractable because the size of $T(\mathbf{w})$ is exponential to the length of $\mathbf{w}$. To avoid exponential explosion, we represent $T(\mathbf{w})$ in a packed form of HPSG parse trees. A parse tree of HPSG is represented as a set of tuples $\langle m, l, r\rangle$, where $m, l,$ and r are the signs of mother, left daughter, and right daughter, respectively[1]. In chart parsing, partial parse candidates are stored in a *chart*, in which phrasal signs are identified and packed into an equivalence class if they are judged to be equivalent and dominate the same word sequence. A set of parse trees is then represented as a set of relations among equivalence classes.

Figure 5.1 shows a chart for parsing *"he saw a girl with a telescope"*, where the modifiee (*"saw"* or *"girl"*) of *"with"* is ambiguous. Each feature structure expresses an equivalence class, and the arrows represent immediate-dominance relations. The phrase, *"saw a girl with a telescope"*, has two trees (A in the figure). Since the signs of the top-most nodes are equivalent, they are packed into an equivalence class. The ambiguity is represented as the two pairs of arrows leaving the node. The algorithm for parsing is presented in Section 2.3.

Formally, a set of HPSG parse trees is represented in a chart as a tuple $\langle E, E_r, \alpha\rangle$, where E is a set of equivalence classes, $E_r \subseteq E$ is a set of root nodes, and $\alpha : E \rightarrow 2^{E\times E}$ is a function to represent immediate-dominance relations.

[1]For simplicity, only binary trees are considered. Extension to unary and n-ary ($n > 2$) trees is trivial.

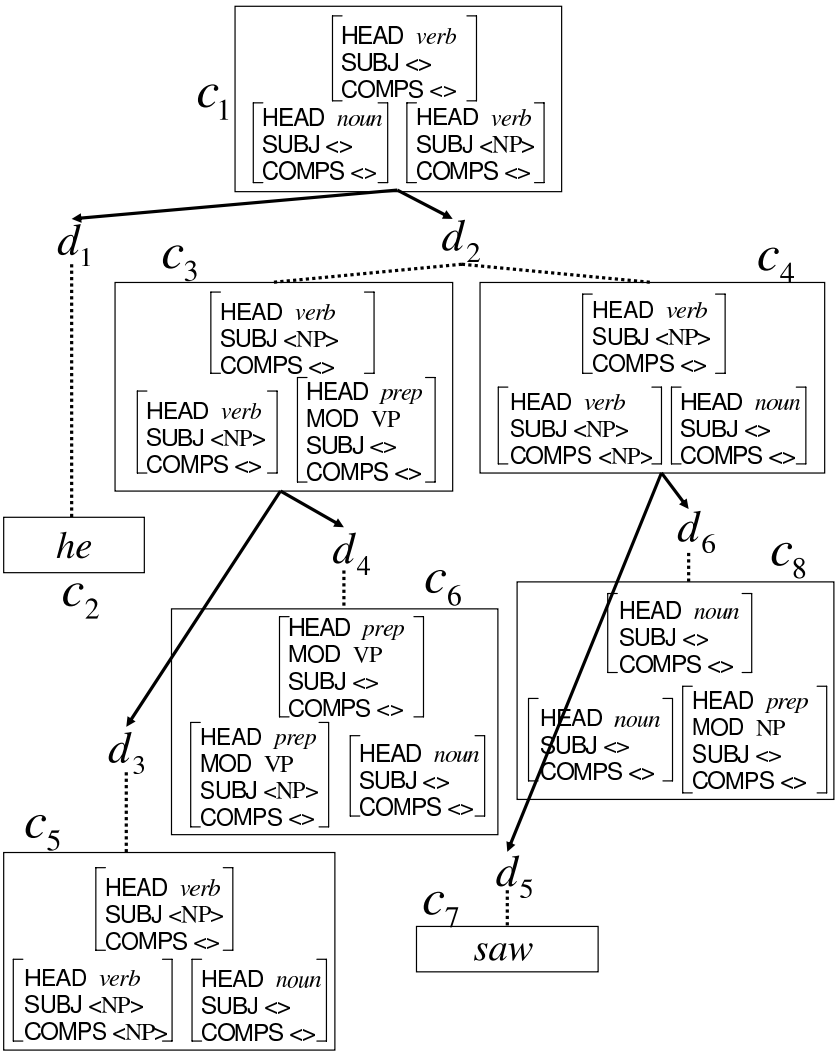


Figure 5.2: Packed representation of HPSG parse trees in Figure 5.1

Our representation of a chart can be interpreted as an instance of *feature forest* proposed in Chapter 4. A feature forest is an "and/or" graph to represent exponentially many tree structures in a packed form. If $T(\mathbf{w})$ is represented in a feature forest, $p(t|T(\mathbf{w}))$ can be estimated using dynamic programming without unpacking the chart. A feature forest is formally defined as a tuple, $\langle C, D, R, \gamma, \delta \rangle$, where C is a set of conjunctive nodes, D is a set of disjunctive nodes, $R \subseteq C$ is a set of root nodes[2], $\gamma : D \rightarrow 2^C$ is a conjunctive daughter function, and $\delta : C \rightarrow 2^D$ is a disjunctive daughter function. The feature functions $f_i(t, \mathbf{w})$ are assigned to conjunctive nodes.

The simplest way to map a chart of HPSG parse trees into a feature forest is to map each equivalence class $e \in E$ to a conjunctive node $c \in C$. However, in HPSG parsing, important features for disambiguation are combinations of a mother and its daughters, i.e., $\langle m, l, r \rangle$. Hence, we map the tuple $\langle e_m, e_l, e_r \rangle$, which corresponds to $\langle m, l, r \rangle$, into a conjunctive node.

Figure 5.2 shows (a part of) the HPSG parse trees in Figure 5.1 represented as a feature forest.

[2]For ease of explanation, the definition of root node is slightly different from the original definition given in Chapter 4. The definition here is translated into the original definition by introducing a dummy root node r' that has no features and only one disjunctive daughter whose daughters are R.

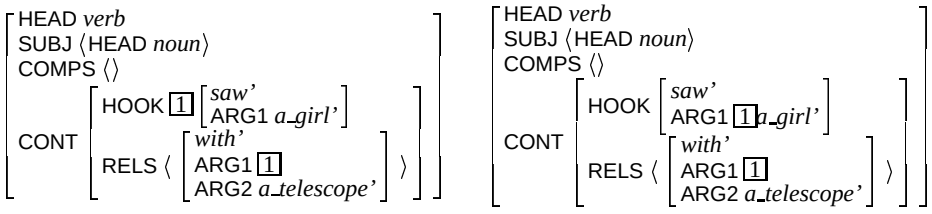


Figure 5.3: Signs with predicate argument structures

Square boxes (c_i) are conjunctive nodes, d_i are disjunctive nodes, solid arrows represent a disjunctive daughter function, and dotted lines express a conjunctive daughter function.

The mapping is formally defined as follows.

- $C = \{\langle e_m, e_l, e_r\rangle | e_m \in E \wedge (e_l, e_r) \in \alpha(e_m)\} \cup \{w | w \in \mathbf{w}\}$
- $D = E$,
- $R = \{\langle e_m, e_l, e_r\rangle | e_m \in E_r \wedge \langle e_m, e_l, e_r\rangle \in C\}$,
- $\gamma(e_m) = \begin{cases} \{\langle e_m, e_l, e_r\rangle | (e_l, e_r) \in \alpha(e_m)\} & \text{if } \alpha(e_m) \neq \emptyset \\ \{w | e_m \text{ is a lexical entry for } w\} & \text{otherwise,} \end{cases}$
- $\delta(c) = \begin{cases} \{e_l, e_r\} & \text{if } c = \langle e_m, e_l, e_r\rangle \\ \emptyset & \text{if } c \in \mathbf{w}. \end{cases}$

Restricting the domain of feature functions to $\langle e_m, e_l, e_r\rangle$ seems to limit the flexibility of feature design. Although it is true to some extent, this does not necessarily mean the impossibility of incorporating features on nonlocal dependencies into the model. This is because a feature forest model does not assume probabilistic independence of conjunctive nodes. This means that we can unpack a part of the forest without changing the model. Actually, we successfully developed a probabilistic model including features on nonlocal predicate-argument dependencies. Details are described in the next section.

5.3 Packed Representation of Predicate Argument Structures

With the method described in the previous section, we can represent a chart of HPSG parsing in a feature forest without unpacking the chart. However, equivalence classes in a chart might increase exponentially because predicate argument structures in HPSG signs represent the semantic relations of all words that the phrase dominates. For example, Figure 5.3 shows phrasal signs with predicate argument structures of "*saw a girl with a telescope*". In the chart in Figure 5.1, these signs are packed into an equivalence class. However, Figure 5.3 shows that the values of CONT, i.e., predicate argument structures, have different values, and the signs as they are cannot be equivalent. As seen in this example, predicate argument structures prevent us from packing signs into equivalence classes.

In this section, we apply feature forest model to predicate argument structures, which may include reentrant structures and non-local dependencies. It is theoretically difficult to apply the feature forest model to predicate argument structures; a feature forest cannot represent graph structures including reentrant structures in a straightforward manner. However, we found that if predicate argument structures are constructed as in the manner described below, they can be represented using feature forests of a tractable size.

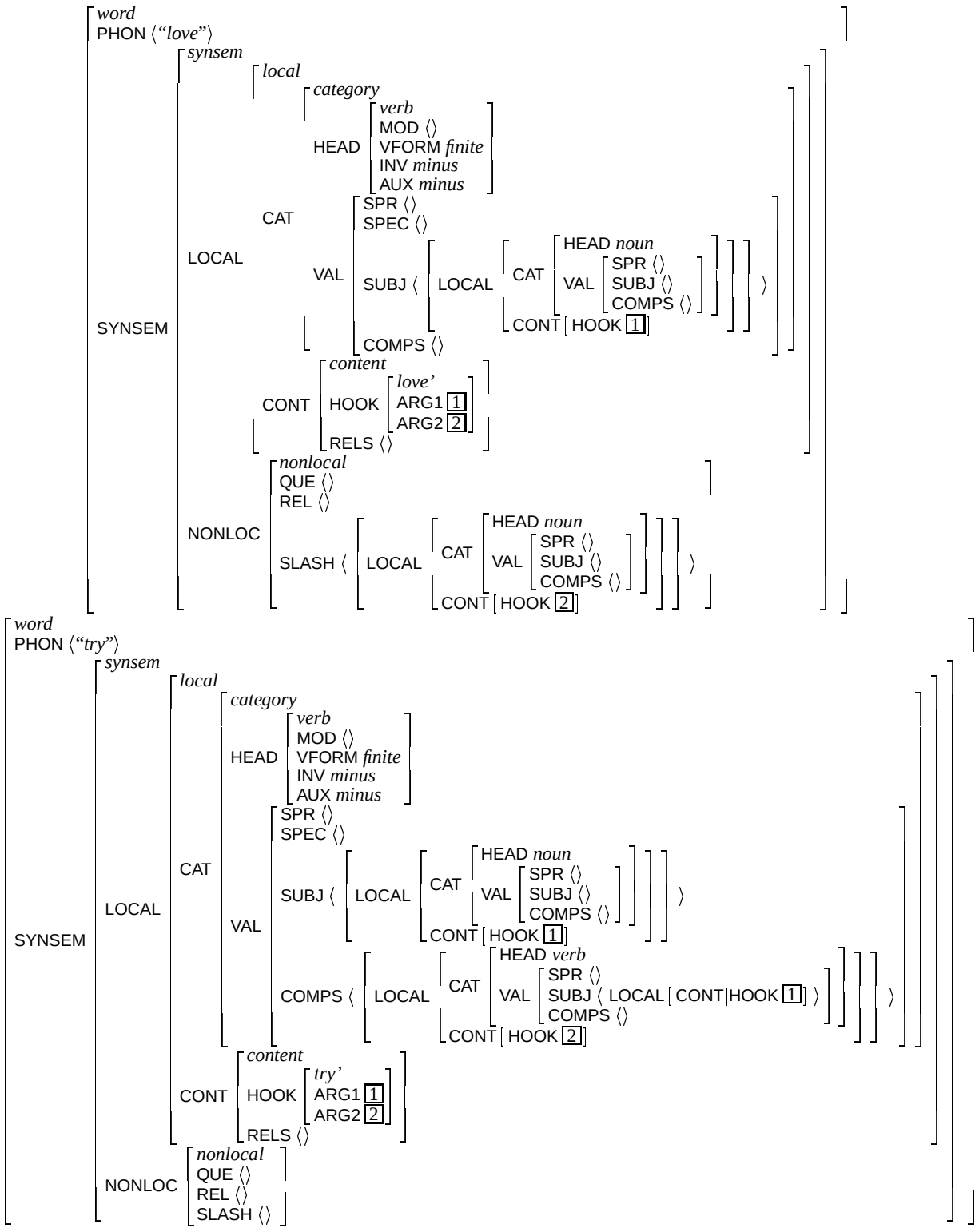


Figure 5.4: Lexical entries including non-local relations

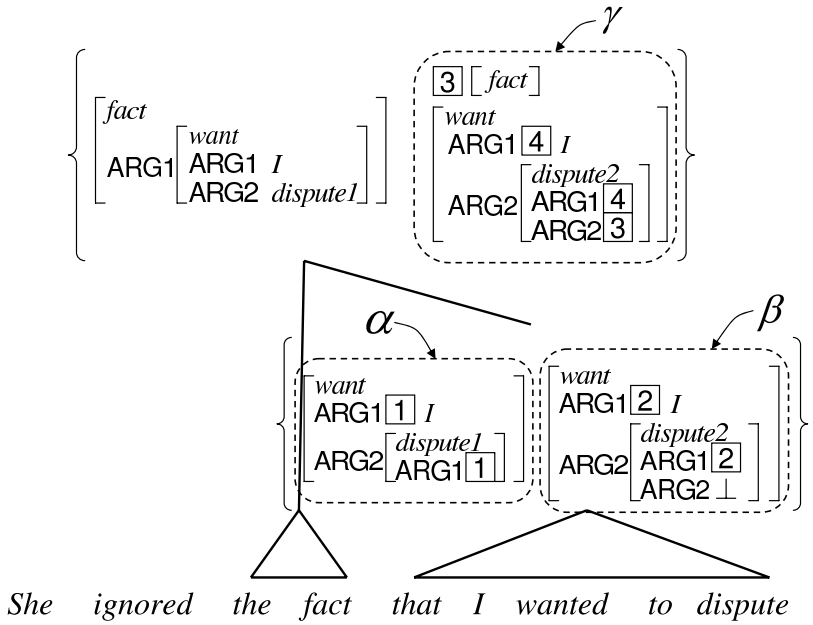


Figure 5.5: Process of composing predicate argument structures

Feature forests can represent predicate argument structures if we assume locality and monotonicity in the composition of predicate argument structures.

Locality In each step of composition of a predicate argument structure, only a limited depth of the daughters' predicate argument structures are referred to. That is, local structures in the deep descendent phrases may be ignored to construct larger phrases. This assumption means that predicate argument structures can be packed into conjunctive nodes with ignoring local structures.

Monotonicity All relations in the daughters' predicate argument structures are percolated to the mother. That is, none of the predicate argument relations in the daughter phrases disappear in the mother. Thus predicate argument structures of descendent phrases can be located at lower nodes in a feature forest without minding their disappearance in ancestor phrases.

Predicate argument structures usually obey the above conditions, even when they include non-local dependencies. For example, Figure 5.4 shows lexical entries of HPSG, which are for the *wh*-extraction of the object of "*love*" (upper) and for the control construction of "*try*" (lower). The first condition is satisfied because both lexical entries refer to CONT|HOOK of argument signs in SUBJ, COMPS, and SLASH. None of the lexical entries access directly to ARGX of the arguments. The second condition is also satisfied because the values of CONT|HOOK of all of the argument signs are percolated to ARGX of the mother. In addition, the elements in CONT|RELS are percolated to the mother by the Semantic Principle. Compositional semantics usually satisfies the above conditions, including MRS. The composition of MRS refers to HOOK features, and no internal structures of daughters. The Semantic Principle of MRS also assures that all semantic relations in RELS are percolated to the mother.

Under these conditions, local structures of predicate argument structures are encoded into a conjunctive node when the values of all of its arguments have been instantiated. We introduce the notion of *inactives* to denote such local structures.

$$\begin{bmatrix} \textit{dispute1} \\ \text{ARG1}\ \bot \end{bmatrix} \qquad \begin{bmatrix} \textit{dispute2} \\ \text{ARG1}\ \bot \\ \text{ARG2}\ \bot \end{bmatrix}$$

Figure 5.6: Predicate argument structures of "*dispute*"

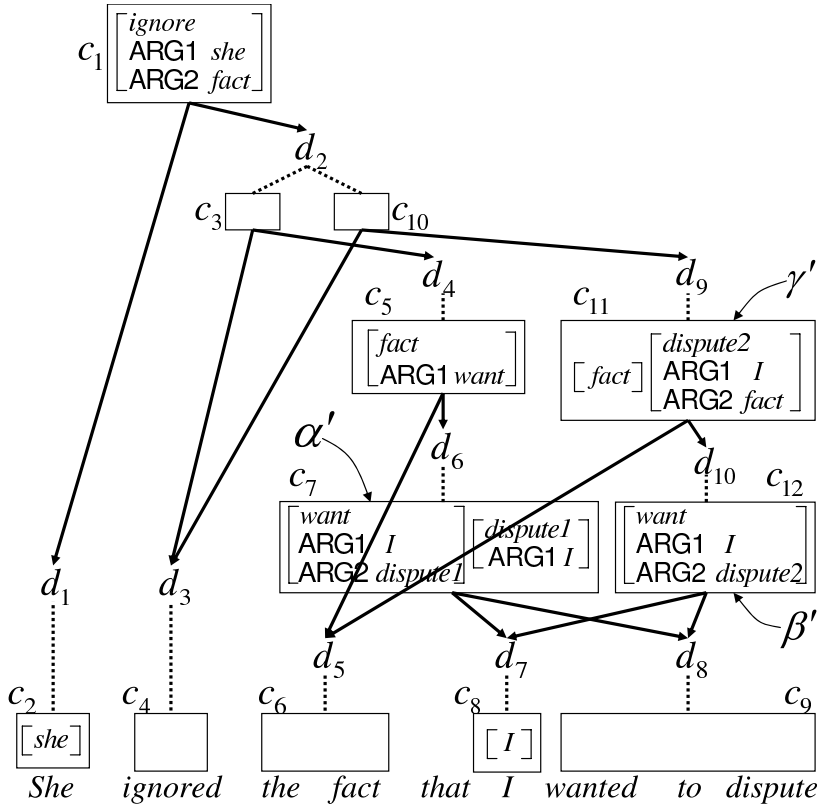


Figure 5.7: Feature forest representation of predicate argument structures

Definition 5.1 (Inactives) *An* inactive *is a subset of predicate argument structures in which all arguments have been instantiated.*

Because inactive parts will not change during the rest of the parsing process, they can be placed in a conjunctive node. By placing newly generated inactives into corresponding conjunctive nodes, a set of predicate argument structures can be represented in a feature forest by packing local ambiguities, and non-local dependencies are preserved.

Figure 5.5 illustrates a process of parsing the sentence "*She ignored the fact that I wanted to dispute*," where "*dispute*" has an ambiguity (*dispute1*, intransitive and *dispute2*, transitive). Predicate argument structures of *dispute1* and *dispute2* are shown in Figure 5.6. Curly braces express the ambiguities of partially constructed predicate argument structures. The resulting feature forest is shown in Figure 5.7. Boxes denote conjunctive nodes and d_x represent disjunctive nodes.

The clause "*I wanted to dispute*" has two possible predicate argument structures: one corresponding to *dispute1* (α in Figure 5.5) and the other corresponding to *dispute2* (β in Figure 5.5). The nodes of the predicate argument structure α are all instantiated, that is, it contains only inactives. The corresponding conjunctive node (α' in Figure 5.7) has two inactives, for *want* and *dispute1*. The

other structure β has an unfilled object in the argument (ARG2[3]) of *dispute2*, which will be filled by the non-local dependency. Hence, the corresponding conjunctive node β' has only one inactive corresponding to *want*, and the remaining part that corresponds to *dispute2* is passed on for further processing. When we process the phrase "*the fact that I wanted to dispute*," the object of *dispute2* is filled by "*fact*" (γ in Figure 5.5), and the predicate argument structure of *dispute2* is then placed into a conjunctive node (γ' in Figure 5.7).

One of the beneficial characteristics of this packed representation is that the representation is isomorphic to the parsing process, i.e., a chart. Hence, we can assign features of HPSG parse trees to a conjunctive node, together with features of predicate argument structures. In Chapter 6, we will investigate the contribution of features on parse trees and predicate argument structures to the disambiguation of HPSG parsing.

5.4 Filtering by Preliminary Distribution

The above method is an essential solution for the tractable estimation of maximum entropy models on exponentially many HPSG parse trees. However, the problem of computational cost remains. Construction of feature forests requires parsing of all of the sentences in a treebank. Despite the development of methods to improve HPSG parsing efficiency (Oepen et al., 2002a), exhaustive parsing of all sentences is still expensive.

We consider that computation of parse trees with low probabilities can be omitted in the estimation stage because $T(\mathbf{w})$ can be approximated by parse trees with high probabilities. To achieve this, we first prepared *a preliminary probabilistic model* whose estimation did not require the parsing of a treebank. The preliminary model was used to reduce the search space for parsing a training treebank.

The preliminary model in this study is a unigram model, $\bar{p}(t|\mathbf{w}) = \prod_{w \in \mathbf{w}} p(l|w)$, where $w \in \mathbf{w}$ is a word in the sentence $\mathbf{w}$, and l is a lexical entry assigned to w. This model is estimated by counting the frequency of each lexical entry assigned to w. Hence, the estimation does not require parsing of a treebank.

The preliminary model is used for filtering lexical entries when we parse a treebank. Given this model, we restrict the number of lexical entries used to parse a treebank. With a threshold n for the number of lexical entries and a threshold ϵ for the probability, lexical entries are assigned to a word in descending order of probabilities, until the number of assigned entries exceeds n, or the accumulated probability exceeds ϵ. If this procedure does not assign a lexical entry necessary to produce a correct parse, it is added to the list of lexical entries. This assures that the filtering method does not exclude correct parse trees from parse forests.

Figure 5.8 shows an example of filtering lexical entries assigned to "*saw*". With $\epsilon = 0.95$, four lexical entries are assigned. Although the lexicon includes other lexical entries, such as a verbal entry taking a sentential complement ($p = 0.01$ in the figure), they are filtered out. While this method reduces the time required for parsing a treebank, this approximation causes bias in the training data and results in lower accuracy. The trade-off between the parsing cost and the accuracy will be examined experimentally in Section 6.3.2.

We have several ways to integrate $\bar{p}$ with the estimated model $p(t|T(\mathbf{w}))$. In the experiments, we will empirically compare the following methods in terms of accuracy and estimation time.

Filtering only The unigram probability $\bar{p}$ is used only for filtering.

Product The probability is defined as the product of $\bar{p}$ and the estimated model p.

[3] As described in Section 2.2.1, $\perp$ (bottom) represents an uninstantiated value (Carpenter, 1992).

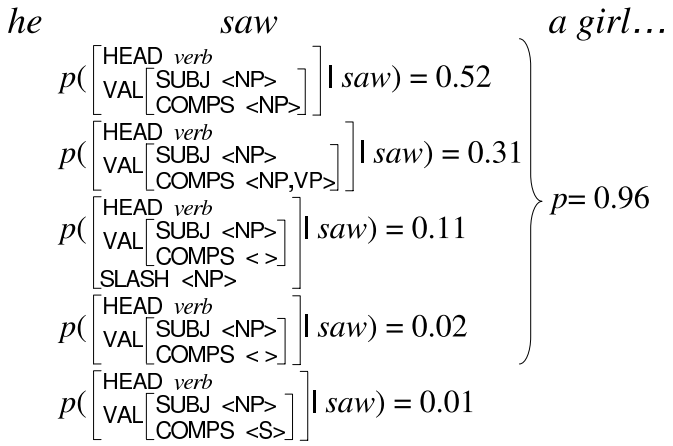


Figure 5.8: Filtering of lexical entries for *"saw"*

Table 5.1: Templates of atomic features

RULE	name of the applied schema
DIST	distance between the head words of the daughters
COMMA	whether a comma exists between daughters and/or inside of daughter phrases
SPAN	number of words dominated by the phrase
SYM	symbol of the phrasal category (e.g. NP, VP)
WORD	surface form of the head word
POS	part-of-speech of the head word
LE	lexical entry assigned to the head word
ARG	argument label of a predicate

Reference distribution $\bar{p}$ is used as a reference distribution of p.

Feature function $\log \bar{p}$ is used as a feature function of p. This method was shown to be a generalization of the reference distribution method (Johnson and Riezler, 2000).

5.5 Features

Feature functions in maximum entropy models are designed to capture the characteristics of $\langle e_m, e_l, e_r \rangle$. In this thesis, we investigate combinations of the atomic features listed in Table5.1. The following combinations are used for representing the characteristics of binary/unary schema applications.

$$f_{\text{binary}} = \left\langle \begin{array}{l} \text{RULE,DIST,COMMA,} \\ \text{SPAN}_l, \text{SYM}_l, \text{WORD}_l, \text{POS}_l, \text{LE}_l, \\ \text{SPAN}_r, \text{SYM}_r, \text{WORD}_r, \text{POS}_r, \text{LE}_r \end{array} \right\rangle,$$

$$f_{\text{unary}} = \langle \text{RULE,SYM,WORD,POS,LE} \rangle,$$

where suffixes l and r denote left and right daughters.

In addition, the following is for expressing the condition of the root node of the parse tree.

$$f_{\text{root}} = \langle \text{SYM,WORD,POS,LE} \rangle$$

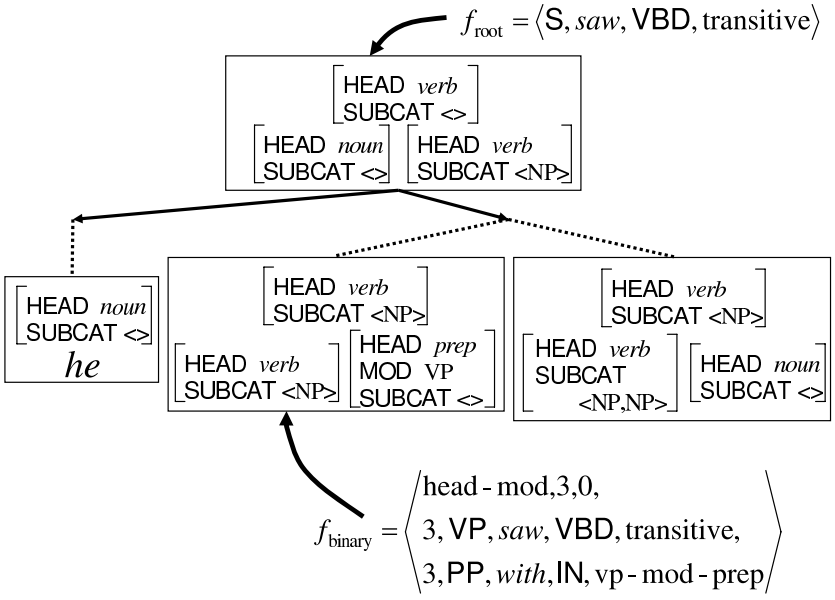


Figure 5.9: Example features for binary schema application and root condition

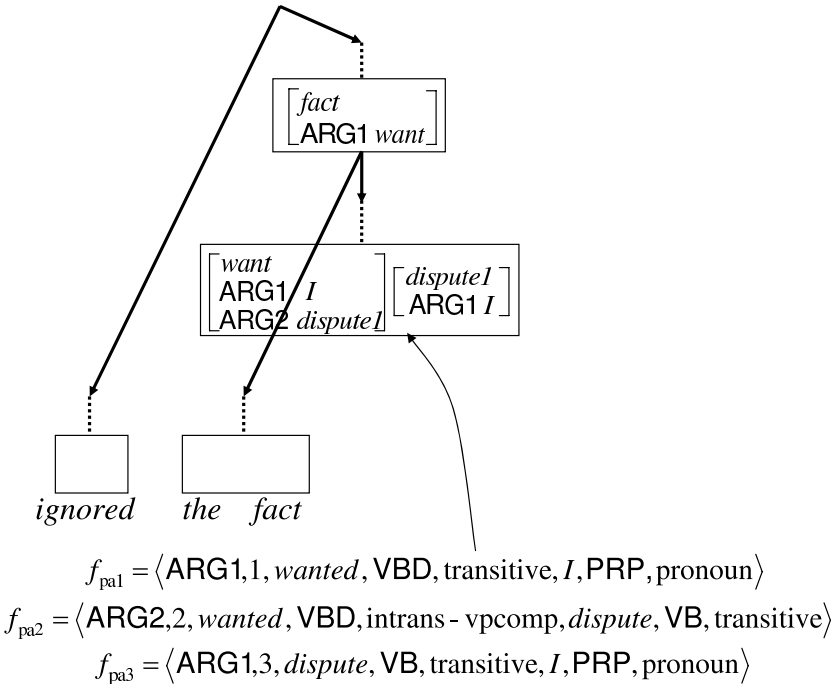


Figure 5.10: Example features for predicate argument structures

Feature functions to capture predicate argument dependencies are represented as follows.

$$f_{\text{pa}} = \langle \text{ARG}, \text{DIST}, \text{WORD}_p, \text{POS}_p, \text{LE}_p, \text{WORD}_a, \text{POS}_a, \text{LE}_a \rangle ,$$

where suffixes p and a represent predicate and argument, respectively.

Figure 5.9 shows examples: f_{root} is for the root node, in which the phrase symbol is S and the

Table 5.2: Feature templates for binary schema (left) and unary schema (right)

RULE	DIST	COMMA	SPAN	SYM	WORD	POS	LE
√	√	√	−	−	√	√	√
√	√	√	−	−	√	√	−
√	√	√	−	−	√	−	√
√	√	√	−	√	√	−	−
√	−	√	√	−	√	√	√
√	−	√	√	−	√	√	−
√	−	√	√	−	√	−	√
√	−	√	√	√	√	−	−
√	√	√	−	−	−	√	√
√	√	√	−	−	−	√	−
√	√	√	−	−	−	−	√
√	√	√	−	√	−	−	−
√	−	√	√	−	−	√	√
√	−	√	√	−	−	√	−
√	−	√	√	−	−	−	√
√	−	√	√	√	−	−	−

RULE	SYM	WORD	POS	LE
√	−	√	√	√
√	−	√	√	−
√	−	√	−	√
√	√	√	−	−
√	−	−	√	√
√	−	−	√	−
√	−	−	−	√
√	√	−	−	−

Table 5.3: Feature templates for root condition

SYM	WORD	POS	LE
−	√	√	√
−	√	√	−
−	√	−	√
√	√	−	−
−	−	√	√
−	−	√	−
−	−	−	√
√	−	−	−

surface form, part-of-speech, and lexical entry of the lexical head are "*saw*", VBD, and a transitive verb, respectively. f_{binary} is for the binary rule application to "*saw a girl*" and "*with a telescope*", in which the applied schema is the Head-Modifier Schema, the left daughter is VP headed by "*saw*", and the right daughter is PP headed by "*with*", whose part-of-speech is IN and the lexical entry is a VP-modifying preposition.

Figure 5.10 shows example features for predicate argument structures. The figure shows features assigned to the conjunctive node denoted as α' in Figure 5.7. Since inactive structures in the node have three predicate-argument relations, three features are activated. The first one is for the relation of "*want*" and "*I*", where the label of the relation is ARG1, the distance between the head words is 1, the surface string and the POS of the predicate are *want* and VBD, and those of the argument are *I* and PRP. The second and the third features are for the other two relations. We may include features on more than two relations, such as the dependencies among "*want*", "*I*", and "*dispute*", although such features are not incorporated currently.

In an actual implementation, some of the atomic features are abstracted (i.e., ignored) for smoothing. Tables 5.2, 5.3, and 5.4 show a full set of templates of combined features used in the experiments. Each row represents a template of a feature function. A check indicates the atomic feature is incor-

Table 5.4: Feature templates for predicate argument dependencies

ARG	DIST	WORD	POS	LE
√	√	√	√	√
√	√	√	−	√
√	√	−	√	√
√	−	√	√	√
√	−	√	−	√
√	−	−	√	√
√	√	√	√	−
√	√	√	−	−
√	√	−	√	−
√	−	√	√	−
√	−	√	−	−
√	−	−	√	−

porated while a hyphen indicates the feature is ignored.

5.6 Discussion and Related Work

This chapter demonstrated that feature forest models are applicable to probabilistic modeling of linguistic structures such as syntactic structures of HPSG and predicate argument structures including non-local dependencies. The presented approach can be regarded as a general solution to the probabilistic modeling of syntactic analysis with lexicalized grammars.

Before feature forest models, studies on probabilistic models of HPSG adopted conventional maximum entropy models to select the most probable parse from parse candidates given by HPSG grammars (Baldridge and Osborne, 2003; Oepen et al., 2002b; Toutanova and Manning, 2002). The difference between these studies and our work is that we used feature forest models to avoid the estimation problem — a problem of exponential increase of unpacked parse results. These studies ignored the problem of exponential explosion, and the number of parse candidates was controlled by heuristic rules or by careful implementation of a fully restrictive grammar, which requires considerable effort to develop. In fact, training data in these studies was very small and consisted only of short sentences. We argue that exponential explosion is inevitable, particularly with the large-scale wide-coverage grammars required to analyze real-world texts. In such cases, their methods of model estimation are intractable. Another approach to estimating log-linear models for HPSG was to extract a small *informative sample* from the original set $T(\mathbf{w})$ (Osborne, 2000). The method has been successfully applied to Dutch HPSG parsing (Malouf and van Noord, 2004). The problem with this method was in the approximation of exponentially many parse trees by a polynomial-size sample. However, their method had an advantage in that any features on parse results could be incorporated into a model, while our method forces feature functions to be defined locally on conjunctive nodes. The trade-off between the approximation solution and the locality of feature functions is an unresolved problem.

Non-probabilistic statistical classifiers were also applied to disambiguation in HPSG parsing: voted perceptrons (Baldridge and Osborne, 2003) and support vector machines (Toutanova et al., 2004). However, the problem of exponential explosion is also inevitable using their methods. An approach similar to ours may be applied, given the study of Taskar et al. (2004) on the learning of a discriminative classifier for a packed representation. If the main part of an update formula is represented with linear combinations, dynamic programming similar to our algorithm should be applicable.

A series of studies on parsing with LFG (Johnson et al., 1999; Riezler et al., 2002, 2000) also proposed a maximum entropy model for probabilistic modeling of LFG parsing. However, similar to the previous studies on HPSG parsing, these studies had no solution to the problem of exponential explosion of unpacked parse results. As described in Section 4.4, Geman and Johnson (2002) proposed an algorithm for maximum entropy estimation for packed representations of LFG parses. Their algorithm is similar to our model, although the feature forest model was not designed for specific data structures but for a general framework of estimating maximum entropy models on packed forest structures.

Recent studies on CCG have proposed probabilistic models of dependency structures or predicate argument dependencies, which are essentially the same as the predicate argument structures described in this chapter. Clark et al. (2002) attempted the modeling of dependency structures, but the model was inconsistent because of the violation of the independence assumption. Hockenmaier (2003) proposed a consistent generative model of predicate argument structures. The probability of a non-local dependency was conditioned on multiple words for preserving the consistency of the probability model; that is, probability $p(I|\textit{want}, \textit{dispute})$ in Section 5.3 was directly estimated. The problem was that such probabilities could not be estimated directly from the data due to data sparseness, and a heuristic method had to be employed. Probabilities were therefore estimated as the average of individual probabilities conditioned on a single word. Another problem is that the model is no longer consistent when unification constraints such as those in HPSG are introduced. Our solution is free from the above problems, and is applicable to various grammars, not only HPSG and CCG.

The latest work on parsing with LFG (Kaplan et al., 2004; Riezler and Vasserman, 2004) and CCG (Clark and Curran, 2003, 2004b) applied feature forest models to disambiguation, and reported higher accuracy than previous studies in experiments on parsing of Penn Treebank. These researchers applied feature forests to representations of packed parse results of LFG and dependency/derivation structures of CCG. Their work demonstrated the applicability and effectiveness of feature forest models in parsing with wide-coverage lexicalized grammars.

Clark and Curran (2004a) described a method for reducing the cost of parsing a training treebank in the context of CCG parsing. They first assigned each word a small number of *supertags*, which correspond to lexical entries in our case, and parsed *supertagged sentences*. Since they did not use the probabilities of supertags in a parsing stage, their method corresponds to our "filtering only" method. However, they also applied the same supertagger in a parsing stage, and this seemed to be crucial for high accuracy. This means that they estimated the probability of producing a parse tree from a supertagged sentence.

Chapter 6

Evaluation of an HPSG Parser

The methods described in the previous chapters were implemented for the development of an HPSG parser for English. This chapter reports empirical evaluation of the HPSG parser. First, performance of the grammar without disambiguation models is evaluated. Specifications of the grammar are presented, coverage of the grammar against real-world texts is measured, and correctness of treebank conversion is manually investigated. Next, performance of disambiguation models is evaluated. Efficacy of feature forest models and filtering methods is revealed experimentally. Additionally, contributions of features to accuracy and empirical data concerning parsing accuracy are reported.

6.1 Experiment Settings

The methods proposed in this dissertation were implemented in the following software packages. They are available on-line, and all of the experiments reported in this chapter can be reproduced.

The MAYZ Toolkit A toolkit for supporting corpus-oriented development of lexicalized grammars. Available at:
`http://www-tsujii.is.s.u-tokyo.ac.jp/mayz/`

Amis A maximum entropy estimator for feature forests. Available at:
`http://www-tsujii.is.s.u-tokyo.ac.jp/amis/`

Enju An HPSG parser for English developed using the MAYZ toolkit. Available at:
`http://www-tsujii.is.s.u-tokyo.ac.jp/enju/`

It should be noted that the MAYZ toolkit and Amis are not specialized to any specific grammar theories. The tools provide general solutions to the problems of grammar development and probabilistic modeling. The MAYZ Toolkit provides tools for treebank conversion, lexicon collection, grammar/treebank browsing, and the development of probabilistic models. Grammar writers can develop grammar by implementing linguistic principles and annotation rules. The only assumption in the MAYZ Toolkit is that a grammar is represented with typed feature structures in LiLFeS (Makino et al., 1997, 1998), which is a logic programming language for the efficient processing of typed feature structures. Amis is a general-purpose estimator for maximum entropy models. Given training data represented with feature forests, the tool automatically estimates optimal parameters using GIS, IIS, or limited-memory BFGS. It also supports Gaussian priors and reference distributions. Enju is a package of an English HPSG grammar and a probabilistic model for disambiguation. The development of the grammar and the disambiguation model was supported by the MAYZ toolkit and Amis. Experimental results reported in this chapter were produced with Enju version 2.1.

Table 6.1: Specification of Penn Treebank

	# sentences	# words	avg. sentence length
Section 02-21 (training set)	39,832	950,028	22.86
Section 22 (development set)	1,700	40,117	22.61
Section 23 (final test set)	2,416	56,684	22.48
Section 00 (manual evaluation set)	1,921	46,451	23.18

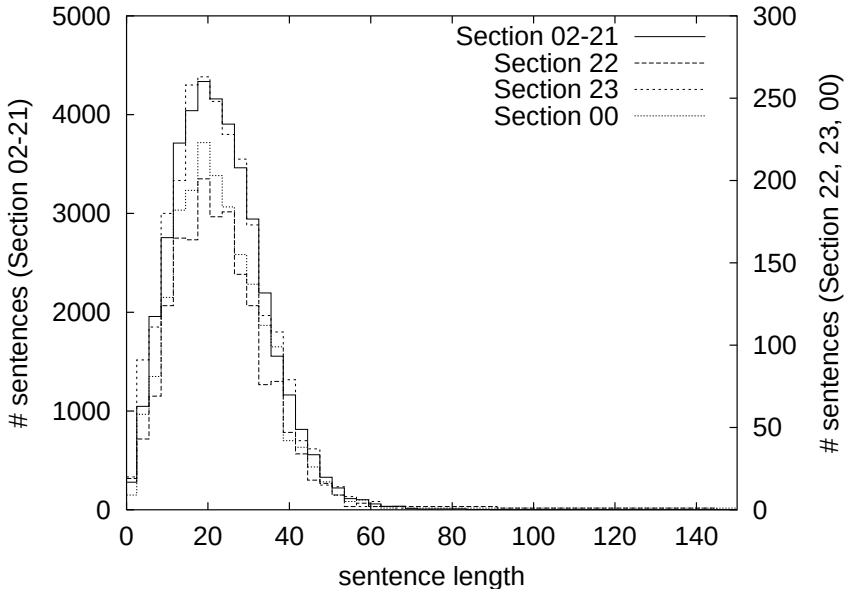


Figure 6.1: Distribution of sentence length

The test data for the evaluation is Penn Treebank (Marcus et al., 1994), which is a well-known corpus for the evaluation of CFG parsers (Charniak, 1997, 2000; Charniak and Johnson, 2005; Collins, 1996, 1997). Penn Treebank consists of sentences collected from Wall Street Journal. Following the previous studies, we used the portion of Section 02-21 for the grammar development and the training of probabilistic models. Section 22 was used as the development set; the set was used for evaluation and error analysis during the development of the parser. Section 23 was used as the final test set. This set was used only for the final evaluation. Section 00 was additionally used for the manual analysis of results because Section 23 should not be investigated manually. Prior to experiments, treebanks were preprocessed to eliminate words that were assigned part-of-speech "." (e.g. "." and "?").

Table 6.1 shows the number of sentences/words and the average sentence length in each portion of the Penn Treebank. As the average sentence length is not very different, the difficulty of parsing is expected to not vary among them. Figure 6.1 shows distributions of sentences in terms of sentence length. The Distributions again are not different among training, development, and test data.

Otherwise noted, all of the following experiments were conducted on AMD Opteron servers with a 2.4-GHz CPU and 16-GB memory. The programs are implemented in C++, and compiled with GCC version 3.2.2.

Table 6.2: Number of annotation rules

Target of annotation	# of rules
Errors of non-/pre-terminal symbols	102
Coordination	3
Insertion & apposition	3
Disagreements in constituent structures	15
Ambiguous symbols	5
Determiners	11
Head/argument/modifier annotation and binarization	63
Subject extraction	2
Auxiliary and control verbs	6
Slash & filler-head schema	13
Relative clauses	8
Category and rule name assignment	85
Predicate-argument structures	13
Others	14
Total	343

Table 6.3: Specification of the HPSG treebank

	# sentences	# words	avg. sentence length
Section 02-21	37,886	854,606	22.56
Section 22	1,644	36,995	22.50
Section 23	2,299	51,097	22.23
Section 00	1,811	41,192	22.75

6.2 Evaluation of a Grammar

We implemented linguistic principles introduced in Section 3.2 and empirical annotation rules described in Section 3.3. Table 6.2 lists the number of empirical annotation rules. "Target of annotation" represents categories of rules, which were explained in Section 3.3. The number of rules imply the variety of constructions in the target, but does not necessarily indicate the complexity of annotation targets. For example, "category and rule name assignment" requires 85 rules, but most of the rules are simple mappings from Penn Treebank-style non-/pre-terminal symbols into HPSG-style categories as described in Section 3.3. On the contrary, "coordination" is annotated using only three rules, but they are implemented as complex LiLFeS programs because the detection of coordinations requires the computation of similarity between coordinated phrases.

The application of the linguistic principles succeeded for 37,886 out of 39,832 sentences in Section 02-21 of Penn Treebank. The reasons for the failures of principle applications will be discussed later in Section 6.2.3. The same algorithm was applied to Section 22 (1,700 sentences), Section 23 (2,416 sentences), and Section 00 (1,921 sentences) of the Penn Treebank, and we obtained HPSG treebanks for a development set (1,644 sentences) from Section 22, a final test set (2,299 sentences) from Section 23, and a manual evaluation set (1,811 sentences) from Section 00. Specifications of the HPSG treebanks are shown in Table 6.3.

Table 6.4: Number of words/lexical entry templates

	words	templates	templates per word
noun	21,925	186	1.14
verb	4,094	945	1.94
adjective	8,078	62	1.28
adverb	1,295	72	2.75
preposition	159	193	9.17
particle	58	10	1.69
determiner	36	33	3.86
conjunction	94	321	9.46
punctuation	15	120	22.00
total	34,765	1,942	1.43

6.2.1 Grammar Specifications

Table 6.4 lists the number of words/lexical entry templates[1] in the lexicon collected from the HPSG treebank[2]. From 37,886 HPSG derivation trees, we collected 1,942 lexical entry templates for 34,765 words. As expected, nouns occupy a dominant portion of the lexicon, while verbs dominate a large number of templates of lexical entries. The right column shows the average number of templates assigned to each word. Distributions of average numbers are clearly distinct for open and closed words; prepositions, conjunctions, and punctuations are assigned many templates, while nouns, verbs, adjectives, and adverbs are assigned fewer entries. Although particles and determiners are closed words, they are assigned fewer templates because they have fewer syntactic variants.

Compared to the automatic extraction of LTAG (Xia, 1999), the number of lexical entry templates is significantly reduced despite fine-grained feature constraints of HPSG. This implies that the HPSG grammar achieved a greater degree of abstraction. For example, because LTAG cannot deal with the sharing of subjects in VP coordination, the method of Xia (1999) extracted many fragment elementary trees from VP coordination. However, the number of templates extracted by our experiment is greater than that of the CCG grammar (Hockenmaier and Steedman, 2002a). A possible reason is that the HPSG grammar deals with syntactic variety using a limited number of construction rules (in our case, twelve), while the CCG grammar exploits lots of construction rules extracted from a treebank. We can claim that HPSG lexical entries represent syntactic constraints that are represented with CCG construction rules, although we need thorough analysis of the relation between HPSG and CCG grammars to make such claims.

Figure 6.2 shows lexical entry templates frequently observed in the treebank[3]. The most frequent template is for a head noun, the second is for a determiner, and the third is for a noun-modifying noun. Figure 6.3 gives frequent lexical entries for other syntactic categories. They are for an intransitive verb, a noun-modifying adjective, and a post-verb modifier, respectively. Generally, frequently observed lexical entries were linguistically sound. This is because our grammar development method was focusing on extending grammar coverage against real-world texts, and concentrated on the refinement of frequent lexical entries. Corpus-oriented development greatly supported this strategy because we could obtain frequency of linguistic phenomena and entities, such as lexical entries.

[1] Lexical entry templates are types of lexical entries without word-specific constraints. See Section 3.4 for details.

[2] The summation of the number of words is not equal to the total number because a word might be assigned more than one part of speech and be double-counted.

[3] Detailed feature constraints are omitted from the figures for readability.

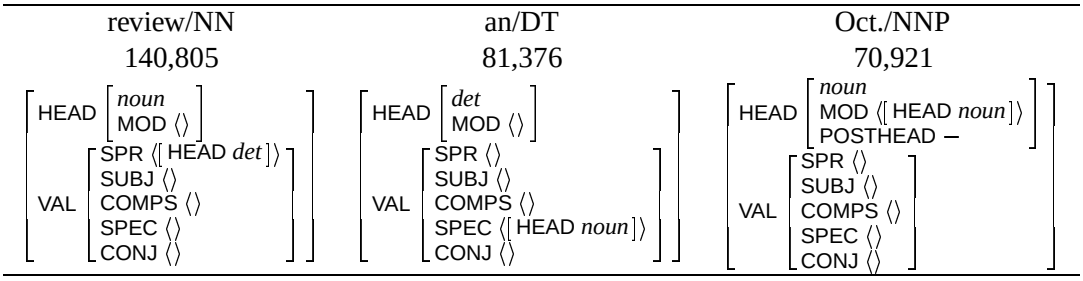


Figure 6.2: Frequent lexical entry templates obtained from Penn Treebank

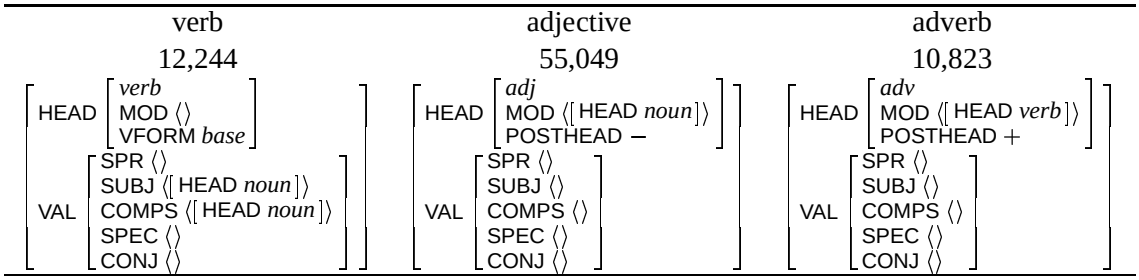


Figure 6.3: Frequent lexical entry templates for verb, adjective, and adverb

Table 6.5: Corpus size vs. number of words/lexical entry templates

# sentences	# words	# lexical entry templates
1,000	4,463	467
2,000	6,915	602
4,000	9,944	764
8,000	14,677	1,060
12,000	18,184	1,214
16,000	21,409	1,357
24,000	26,694	1,589
32,000	30,991	1,772
39,832	34,765	1,942

Table 6.5 and Figure 6.4 show the number of words/lexical entry templates collected from various sizes of treebanks. The number of lexical entry templates is significantly suppressed in comparison to the number of words. This shows the grammar development method proposed in this thesis successfully extracted generalized linguistic regularities from corpus examples. However, the number of templates does not seem to be saturated. This does not necessarily indicate that we will find new types of lexical entry templates in a larger corpus. Observing lexical entries newly found in a latter part of the treebank, we found that most of them were not linguistically sound. They were extracted from ill-formed HPSG derivation trees produced by incorrect applications of heuristic annotation rules, especially for coordination and quotations. That is, most lexical entry templates newly found in a latter part of the treebank are errors. Currently, we apply a method of automatically filtering out erroneous lexical entry templates by thresholding out infrequent templates. Obviously, this method is

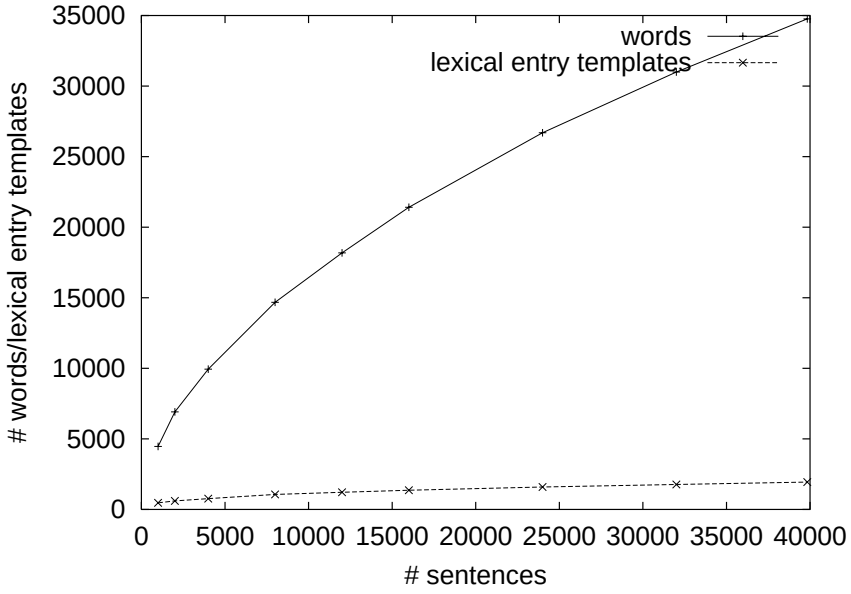


Figure 6.4: Corpus size vs. number of words/lexical entry templates

Table 6.6: Lexical/sentential coverage for Section 23

	seen	unseen				sentential
	⟨sw,sc⟩	⟨sw,sc⟩	⟨sw,uc⟩	⟨uw,sc⟩	⟨uw,uc⟩	
G	96.52%	0.87%	0.38%	1.21%	1.03%	54.7%
G_0	99.15%	0.58%	0.27%	0.00%	0.00%	84.8%
G_1	99.13%	0.56%	0.31%	0.00%	0.00%	84.5%
G_3	99.09%	0.54%	0.37%	0.00%	0.00%	84.1%
G_5	99.05%	0.51%	0.43%	0.00%	0.00%	83.7%
G_{10}	98.97%	0.48%	0.55%	0.00%	0.00%	82.5%

not theoretically sound, but is a simple and effective temporary solution.

Since our annotation rules are imperfect at present, there is always the possibility of extracting erroneous lexical entries from a treebank. This is not a disadvantage of our method, but we believe that it can be an advantage. That is, grammar writers can identify errors in a treebank by observing a lexicon. Errors are corrected by the modification of annotation rules and linguistic principles, and the side effect of the modification can be detected empirically. In addition, the quality of a treebank is quantified since the suppression of the increase of lexical entry templates can be an indicator of the quality.

6.2.2 Evaluation of Coverage

Table 6.6 and Figure 6.5 show lexical/sentential coverage for Section 23. Coverage was measured by comparing the acquired lexicon to lexical entries in the HPSG treebank Section 23. The table shows the coverage of a grammar without/with unknown word handling. The method of handling unknown words is similar to Hockenmaier and Steedman (2002a); words occurring fewer than 20 times

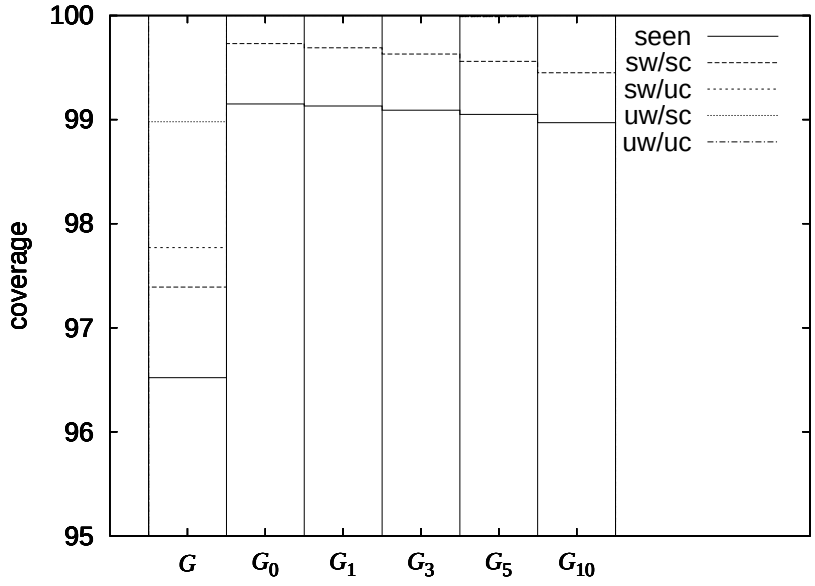


Figure 6.5: Lexical coverage for Section 23

in Section 02-21 were defined as "unknown words". Lexical entries for "unknown words" would be assigned to unknown words in open texts. In the table, G/G_x denotes grammars without/with unknown word handling, and a suffix denotes a threshold of the frequency of lexical entry templates; a grammar includes a lexical entry template only if its frequency is more than the threshold. The "seen" and "unseen" columns represent lexical coverage, which is the same measure as Xia (1999) for LTAG and Hockenmaier and Steedman (2002a) for CCG. That is, the "seen" column has the ratio of word/template pairs covered by the grammar. The results are comparable to the existing studies, despite the fine-grained constraints of HPSG. The "unseen" columns have the ratio of pairs not covered by the grammar, where "sw"/"uw" mean that words were seen/unseen, and "sc"/"uc" mean that templates were seen/unseen in the grammar. Without unknown word handling, the ratio of unknown words was higher. This is natural because we often find unknown open class words such as new proper nouns in open texts. With unknown word handling, we observed that the ratio of "sw"/"sc" was relatively higher. This means that both word and template existed in the grammar, but they were not related. This can be improved by a more sophisticated method of treating unknown words.

The "sentential" column indicates sentential coverage in a strong sense, where a sentence was judged to be covered only if the grammar included correct lexical entries for all words in the sentence. This measure is considered to be the "ideal" accuracy attained by the grammar, i.e., sentential accuracy when a parser and a disambiguation model worked perfectly. The results demonstrate that impressively high coverage against real-world sentences is attained for the analysis by the HPSG grammar. The results also reveal that unknown word handling is crucial for sufficient sentential coverage.

Table 6.7 and Figure 6.6 show lexical/sentential coverage of grammars extracted from different sizes of treebanks. The grammar "G_3" was used in this experiment. Lexical coverage is sufficiently high with a very small treebank, while we need a larger treebank to have higher sentential coverage. To obtain over 80% sentential coverage, we need around 20,000 sentences. The sentential coverage seems to reach a saturation point at around 40,000 sentences. This indicates that we cannot improve sentential coverage simply by using a larger corpus. To obtain higher sentential coverage, we need to

Table 6.7: Corpus size vs. coverage

# sentences	lexical coverage	sentential coverage
1,000	94.14	36.1
2,000	96.18	52.3
4,000	97.49	64.9
8,000	98.22	73.0
12,000	98.49	75.9
16,000	98.66	78.2
24,000	98.87	81.7
32,000	98.97	82.9
39,832	99.09	84.1

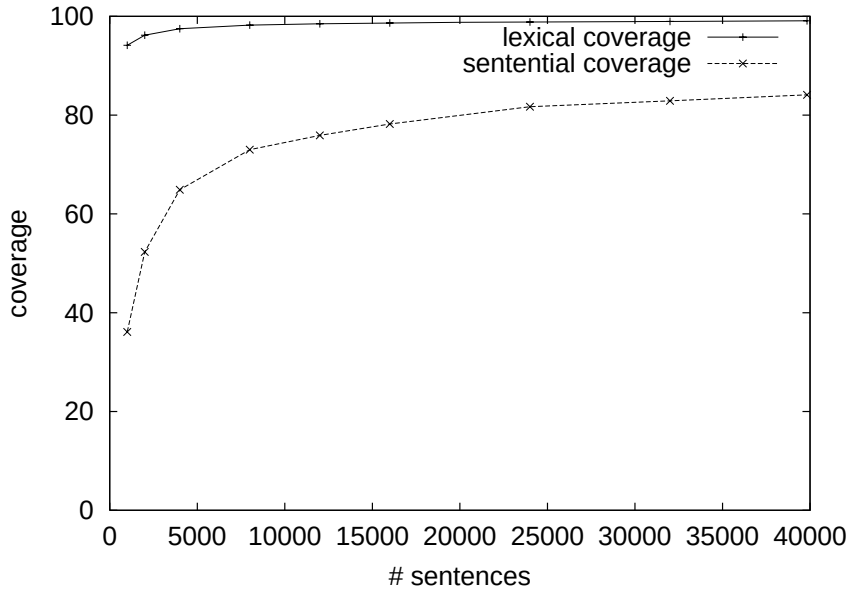


Figure 6.6: Corpus size vs. coverage

augment empirical annotation rules and linguistic principles, as we will discuss below.

Table 6.8 and Figure 6.7 show sentential coverage for different sentence lengths. A dotted line in the figure shows $y = (0.9909^x) \times 100$, which represents the assumption that an uncovered lexical entry is found independently of sentence length. The figure supports this assumption. A consequence is that sentence length is irrelevant to lexical coverage, but is critical for sentential coverage.

Table 6.9 shows manual classification of causes of uncovered lexical entries in Section 00. The lexical/sentential coverage of G_3 against Section 00 was 98.63%/83.4%, respectively. We investigated 100 randomly selected lexical entries in Section 00 that are not covered by G_8. Since one sentence (1,855th sentence) contains a very long listing of proper names and includes too many (89) lexical entries that are not covered by the grammar, we eliminated the sentence from the classification. The result shows that nearly half of the degradation of the coverage is due to errors of the test data. Most are caused by the incorrect application of empirical annotation rules. This means that the evaluation of coverage is underestimated by the wrong HPSG parse trees in the test data. We can therefore

Table 6.8: Coverage vs. sentence length

sentence length	sentential coverage	# sentences
0-2	94.7	19
3-5	96.7	90
6-8	92.8	111
9-11	92.2	179
12-14	88.1	194
15-17	88.7	247
18-20	88.8	251
21-23	87.1	233
24-26	81.1	217
27-29	78.5	200
30-32	75.9	162
33-35	77.6	107
36-38	74.8	103
39-41	70.7	75
42-44	80.6	36
45-47	62.9	35
48-50	63.6	11
51-53	66.7	12
54-56	85.7	7
57-59	50.0	2
60-62	40.0	5
63-65	50.0	2
66-68	0.0	1

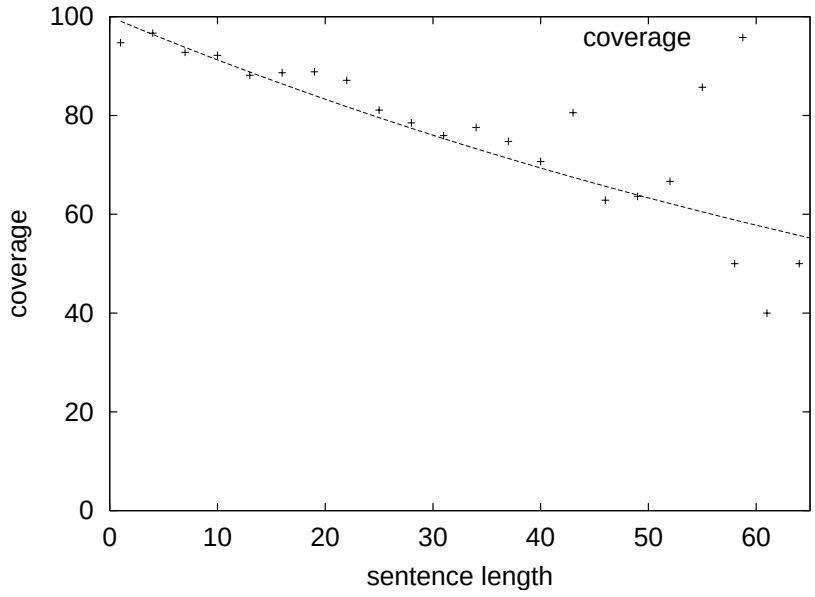


Figure 6.7: Coverage vs. sentence length

Table 6.9: Causes of uncovered lexical entries (Section 00)

Errors of Penn Treebank		10
Errors of test data	Coordinations	15
	Noun phrases	11
	Prepositional phrases	4
	Participle constructions	3
	Others	15
Lack of lexical entries	Modifiers	6
	Adjective with complement	3
	Problem of unknown word handling	3
	Unlike coordination	2
	Noun with complement	2
	Others	7
Constructions currently unsupported	Free relative	4
	Extraposition	3
	Insertion of participle construction	2
Idioms		6
Non-linguistic constructions		4

improve the measure of coverage through the refinement of heuristic annotation rules applied to the test data. The second cause is the lack of lexical entries. This is a defect of the grammar, and a method of unknown word handling and a grammar design should be enhanced to treat them properly. Constructions currently unsupported includes free relatives, extraposition, and insertion of participle constructions. Since the HPSG theory already provides their explanations, they will be supported by the grammar in the near future. Idioms and non-linguistic constructions (such as listings) will require special treatment. These improvements will be topics in future research.

6.2.3 Investigation of Treebank Conversion

The application of linguistic principles failed for 1,946 sentences (4.9%) in Section 02-21, and the reasons for the failures were manually investigated for the sentences in Section 02 (67 failures). The results listed in Table 6.10 reveal that dominant reasons are shortcomings in annotation rules and errors in the treebank. We intend to reduce both of these by enhancing annotation rules, which should lead to further improvements in the grammar. We rarely found constructions that are not handled by the theory of HPSG ("argument cluster coordination" in the table). The results indicate that the fragility of syntactic analysis was not due to syntactic theory itself.

Table 6.11 shows results of manual evaluation of treebank conversion, and Table6.12 lists causes of incorrect conversions. 100 randomly selected parse trees in the HPSG treebank Section 00 (1,984 words) were manually judged. Lexical entries were judged as "incorrect" when they were not linguistically sound, or undecidable (i.e., needs discussion); for example, while the head of "12 %" was annotated as "12", this does not seem proper treatment of the expression and was judged as "incorrect" in the evaluation. The results indicate that a significant portion of incorrect conversions are due to misapplications of empirical annotation rules. This is because we developed empirical annotation rules to be applied to a closed set of data. That is, they were not intended to be applied to an open treebank. However, we cannot ignore constructions which are not treated by our grammar currently. In

Table 6.10: Reasons for the violation of linguistic principles (Section 02)

Argument cluster coordination		13
Errors in Penn Treebank	Errors in tree structures	8
	Errors of part-of-speech	6
	Nonsentences	2
Constructions currently unsupported	*Tough* construction	6
	Predicative sentence without *be*	3
	Topicalization of VP	3
	Slash in coordination/insertion	3
	To-relative	2
	Others	3
Misapplication of empirical annotation rules	Insertion & apposition	4
	Determiners	3
	Coordination	3
	Unexpected constructions	3
	Other annotation rules	5

Table 6.11: Manual evaluation of treebank conversion (Section 00)

Correct lexical entries	1,790
Incorrect lexical entries	52
Conversion failures (# words)	142
Correctly converted sentences	66
Sentences containing incorrect lexical entries	27
Conversion failures (# sentences)	7

Table 6.12: Causes of incorrect lexical entries

Errors of Penn Treebank		5
Constructions currently unsupported	Idioms	7
	Modification of pronouns	3
Misapplication of empirical annotation rules	Coordination	10
	Complement of adverb/adjective	6
	Category assignment	5
	Head detection	4
	Insertion	3
	Quantifier phrases	3
	Others	6

Table 6.13: Specification of test data for the evaluation of parsing accuracy

	# sentences	# avg. length	# avg. dependencies
Test set (Section 23, $<$ 40 words)	2,144	20.52	20.95
Test set (Section 23, $<$ 100 words)	2,299	22.23	22.74
Development set (Section 22, $<$ 40 words)	1,525	20.69	21.08
Development set (Section 22, $<$ 100 words)	1,641	22.43	22.94

particular, we do not treat idioms, and this will be an area for future work in the ongoing improvement of the grammar.

6.3 Evaluation of Disambiguation Models

This section presents experimental results on parsing accuracy after disambiguation by feature forest models. In all of the following experiments, we used the HPSG grammar G_3 described in the previous section. The data for the training of disambiguation models was the HPSG treebank derived from Section 02-21 of the Penn Treebank, i.e., the same set used for grammar development. For the training of disambiguation models, we eliminated sentences of no less than 40 words and for which the parser could not produce the correct parses. The resulting training set consists of 33,604 sentences (when $n = 10$ and $\epsilon = 0.95$; see Section 6.3.2 for details). The treebanks derived from Section 22 and 23 were used as the development and final test sets, respectively. Following the previous studies on CFG parsing, the accuracy is measured for sentences of less than 40 words and for those of less than 100 words. Table 6.13 shows specifications of test data. The last column shows the average number of predicate-argument dependencies, which are used for the evaluation of parsing accuracy.

The measure for evaluating parsing accuracy is precision/recall of predicate-argument dependencies output by the parser. A predicate-argument dependency is defined as a tuple $\langle w_h, w_n, \pi, \rho \rangle$, where w_h is a head word of the predicate, w_n is a head word of the argument, π is a type of the predicate (e.g., adjective, intransitive verb), and ρ is an argument label (MODARG, ARG1, ..., ARG4). For example, "*Spring has come*" has three dependencies as below.

- $\langle$*has*, *spring*, auxiliary verb, ARG1$\rangle$
- $\langle$*has*, *come*, auxiliary verb, ARG2$\rangle$
- $\langle$*come*, *spring*, intransitive verb, ARG1$\rangle$

Labeled precision/recall (LP/LR) is the ratio of tuples correctly identified by the parser, while unlabeled precision/recall (UP/UR) is the ratio of w_h and w_n correctly identified regardless of π and ρ. F-score is a harmonic mean of LP and LR. These measures correspond to those used in other studies measuring the accuracy of predicate-argument dependencies in CCG parsing (Clark and Curran, 2004b; Clark et al., 2002; Hockenmaier, 2003) and LFG parsing (Burke et al., 2004). All predicate-argument dependencies in a sentence are the target of evaluation, including punctuations. The accuracy is measured by parsing test sentences with part-of-speech tags provided by the treebank.

The Gaussian prior was used for smoothing (Chen and Rosenfeld, 1999a), and its hyper-parameter was tuned for each model to maximize F-score for the development set. The algorithm for parameter estimation was the limited-memory BFGS method (Nocedal and Wright, 1999; Nocedal, 1980). The parser was implemented in C++ with the LiLFeS library, and various speed-up techniques for

Table 6.14: Accuracy of predicate-argument relations (test set, < 40 words)

	LP	LR	UP	UR	F-score	Sentence acc.	# parse failures
Baseline	78.10	77.39	82.83	82.08	77.74	18.3	8
Syntactic features	86.92	86.28	90.53	89.87	86.60	36.3	8
Semantic features	84.29	83.74	88.32	87.75	84.01	30.9	8
All	86.54	86.02	90.32	89.78	86.28	36.0	7

Table 6.15: Accuracy of predicate-argument relations (test set, < 100 words)

	LP	LR	UP	UR	F-score	Sentence acc.	# parse failures
Baseline	77.58	76.84	82.22	81.43	77.21	17.1	9
Syntactic features	86.47	85.83	90.06	89.40	86.15	34.1	9
Semantic features	83.81	83.26	87.75	87.16	83.53	28.9	9
All	86.13	85.59	89.85	89.29	85.86	33.8	8

HPSG parsing were used such as quick check and iterative beam search (Ninomiya et al., 2005; Tsuruoka et al., 2004). Other efficient parsing techniques, including global thresholding, hybrid parsing with a chunk parser, and large constituent inhibition, were not used. The results obtained using these techniques are given in Ninomiya et al. (2005). A limit on the number of constituents was set for time-out; the parser stopped parsing when the number of constituents created during parsing exceeded 50,000. In this case, the parser output nothing, and the recall was computed as zero.

Features occurring more than twice were included in the probabilistic models. A method of filtering lexical entries was applied to the parsing of training data (Section 5.4). Unless otherwise noted, parameters for filtering were $n = 10$ and $\epsilon = 0.95$, and a reference distribution method was applied. The unigram model, $p_0(t|s)$, for filtering is a maximum entropy model with two feature templates, $\langle \text{WORD}, \text{POS}, \text{LE} \rangle$ and $\langle \text{POS}, \text{LE} \rangle$. The model includes 24,847 features.

6.3.1 Efficacy of Feature Forest Models

Tables 6.14 and 6.15 show parsing accuracy for the test set, while Tables 6.16 and 6.17 show results for the development set. Table 6.18 gives computation/space costs of model estimation. In the tables, "*Syntactic features*" denotes a model with syntactic features, i.e., f_{binary}, f_{unary}, and f_{root} introduced in Section 5.5. "*Semantic features*" represents a model with features on predicate-argument structures, i.e., f_{pa}. "*All*" is a model with both syntactic and semantic features. The "*Baseline*" row shows the results by reference model, $p_0(t|s)$, used for lexical entry filtering in the estimation of the other models. This model is considered as a simple application of a traditional PCFG-style model; that is, $p(r) = 1$ for any rule r in 12 construction rules of the HPSG grammar.

The results demonstrate that feature forest models have significantly higher accuracy than a reference model. Comparing "*Syntactic features*" with "*Semantic features*", we see that the former model attained significantly higher accuracy than the latter. This indicates that syntactic features are more important for overall accuracy. We will examine the contributions of each atomic feature of syntactic features in Section 6.3.3.

Semantic features were generally considered as important for the accurate disambiguation of syntactic structures. For example, PP-attachment ambiguity cannot be resolved with only syntactic pref-

Table 6.16: Accuracy of predicate-argument relations (development set, < 40 words)

	LP	LR	UP	UR	F-score	Sentence acc.	# parse failures
Baseline	77.40	76.89	82.15	81.61	77.14	16.9	4
Syntactic features	87.12	86.71	90.66	90.23	86.91	37.2	5
Semantic features	84.81	84.38	88.76	88.31	84.59	30.5	5
All	86.86	86.45	90.42	89.99	86.65	35.7	5

Table 6.17: Accuracy of predicate-argument relations (development set, < 100 words)

	LP	LR	UP	UR	F-score	Sentence acc.	# parse failures
Baseline	76.83	75.67	81.53	80.29	76.25	15.7	10
Syntactic features	86.72	85.68	90.26	89.18	86.20	34.9	10
Semantic features	84.31	83.16	88.26	87.05	83.73	28.5	11
All	86.42	85.31	90.00	88.84	85.86	33.5	11

Table 6.18: Computation/space costs of model estimation

	# features	Estimation time (sec.)	Data size (MByte)
Baseline	24,847	499	21
Syntactic features	599,104	511	727
Semantic features	334,821	278	375
All	933,925	716	1,093

erences. However, the results show that a model with only semantic features performs significantly worse than one with syntactic features. Even when combined with syntactic features, semantic features do not improve accuracy. Obviously, semantic preferences are necessary for accurate parsing, but features used in this work were not sufficient to capture semantic preferences. A possible reason is that, as reported in Gildea (2001), bilexical dependencies may be too sparse to capture semantic preferences. For further investigation of the probabilistic modeling of predicate argument structures, we must explore other features to capture semantic preferences, such as semantic-class features.

For reference, our results are competitive with the best corresponding results reported in CCG parsing (LP/LR = 86.6/86.3) (Clark and Curran, 2004b), although our results cannot be compared strictly with other grammar formalisms because each formalism represents predicate-argument dependencies differently. Different from the results of CCG and PCFG (Charniak, 2000; Collins, 1999, 1997, 2003), the recall is clearly lower than precision. This resulted from the HPSG grammar having stricter feature constraints and the parser not being able to produce parse results for around one percent of the sentences. To improve recall, we need techniques of robust processing with HPSG.

From Table 6.18, we conclude that feature forest models are estimated at a tractable computational cost and a reasonable data size. Even when a model includes semantic features including non-local dependencies, estimation cost is tractable and reasonable. The results reveal that feature forest models solve the problem of the estimation of probabilistic models of sentence structures without any independence assumption.

Table 6.19: Estimation method vs. accuracy and estimation time

	LP	LR	F-score	Sentence acc.	Estimation time (sec.)
Filtering only	51.70	49.89	50.78	5.6	449
Product	86.50	85.94	86.22	35.0	1,568
Reference distribution	86.92	86.28	86.60	36.3	511
Feature function	84.81	84.09	84.45	31.8	945

Table 6.20: Filtering threshold vs. accuracy

n, ϵ	LP	LR	F-score	Sentence acc.
5, 0.80	85.09	84.30	84.69	32.4
5, 0.90	85.44	84.61	85.02	32.5
5, 0.95	85.52	84.66	85.09	32.7
5, 0.98	85.50	84.63	85.06	32.6
10, 0.80	85.60	84.65	85.12	32.5
10, 0.90	86.49	85.92	86.20	34.7
10, 0.95	86.92	86.28	86.60	36.3
10, 0.98	87.18	86.66	86.92	37.7
15, 0.80	85.59	84.63	85.11	32.4
15, 0.90	86.48	85.80	86.14	35.7
15, 0.95	87.21	86.68	86.94	37.0
15, 0.98	87.69	87.16	87.42	39.2

6.3.2 Comparison of Filtering Methods

Table 6.19 compares estimation methods introduced in Section 5.4. In all of the following experiments, we show the accuracy for the test set (< 40 words) only. Table 6.19 reveals that our simple method of filtering causes a fatal bias in training data when a preliminary distribution is used only for filtering. However, the model combined with a preliminary model achieved sufficient accuracy. The reference distribution method achieved higher accuracy and lower cost. The feature function method achieved lower accuracy in our experiments. A possible reason for this is that a hyper-parameter of the prior was set to the same value for all the features including the feature of the log-probability given by the preliminary distribution.

Tables 6.20, 6.21 and Figures 6.8, 6.9 show the results of changing the filtering threshold[4]. We can determine the correlation between the estimation/parsing cost and accuracy. In our experiment, $n \geq 10$ and $\epsilon \geq 0.90$ seem necessary to preserve the F-score over 86.0. Figure 6.8 indicates the possibility of further improving the accuracy, although it will require unreasonable computation cost as shown in Figure 6.9. We need to investigate more sophisticated methods of estimation with higher accuracy and less cost.

Table 6.21: Filtering threshold vs. estimation cost

n, ϵ	Estimation time (sec.)	Parsing time (sec.)	Data size (MByte)	# training sentences
5, 0.80	108	5,103	341	33,610
5, 0.90	150	6,242	407	33,610
5, 0.95	190	7,724	469	33,610
5, 0.98	259	9,604	549	33,610
10, 0.80	130	6,003	370	33,610
10, 0.90	268	8,855	511	33,610
10, 0.95	511	15,393	727	33,604
10, 0.98	1,395	36,009	1,230	33,521
15, 0.80	123	6,298	372	33,610
15, 0.90	259	9,543	526	33,610
15, 0.95	735	20,508	854	33,596
15, 0.98	3,777	86,844	2,031	33,146

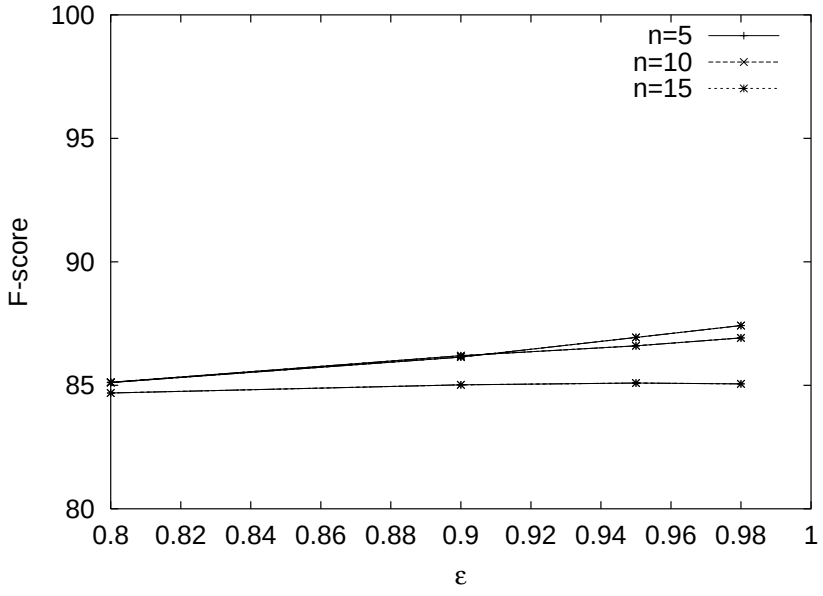


Figure 6.8: Filtering threshold vs. accuracy

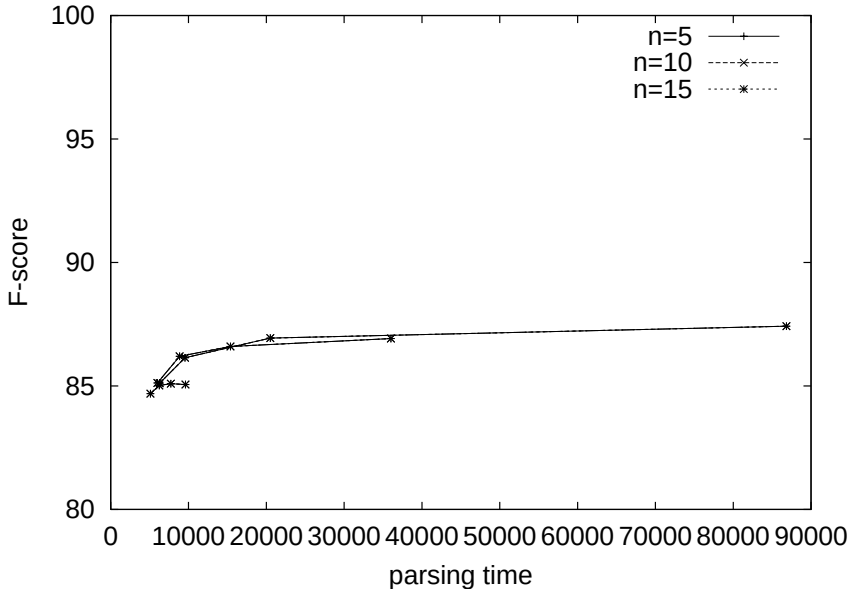


Figure 6.9: Time for parsing a treebank vs. accuracy

Table 6.22: Accuracy with different feature sets (1)

Features	LP	LR	F-score	Sentence acc.	# features
All	86.92	86.28	86.60	36.3	599,104
−RULE	86.83	86.19	86.51	36.3	596,446
−DIST	**86.52**	**85.96**	86.24	35.7	579,666
−COMMA	**86.31**	**85.81**	86.06	34.4	584,040
−SPAN	**86.32**	**85.75**	86.03	35.5	559,490
−SYM	86.74	86.16	86.45	35.4	406,545
−WORD	**86.39**	**85.77**	86.08	35.3	91,004
−POS	**86.18**	**85.61**	85.89	34.1	406,545
−LE	86.91	86.32	86.61	36.8	387,938
None	**78.10**	**77.39**	77.74	18.3	0

6.3.3 Contribution of Features

Tables 6.22 and 6.23 show the accuracy with difference feature sets. Accuracy was measured by removing some of the atomic features from the final model. The last row denotes the accuracy attained by the unigram model (i.e., reference distribution). The numbers in bold type represent a significant difference from the final model according to stratified shuffling tests (Cohen, 1995) with p-value < 0.05. The results indicate that DIST, COMMA, SPAN, WORD, and POS features contributed to the final accuracy, although the differences were slight. In contrast, RULE, SYM, and LE features did not affect accuracy. However, when each was removed together with another feature, the accuracy decreased drastically. This implies that such features have overlapping information.

[4]Because of the shortage of memory, estimation with $n = 15, \epsilon = 0.98$ was run on a machine with 32 giga byte memory.

Table 6.23: Accuracy with different feature sets (2)

Features	LP	LR	F-score	Sentence acc.	# features
All	86.92	86.28	86.60	36.3	599,104
−DIST,SPAN	**85.39**	**84.82**	85.10	33.1	270,467
−DIST,SPAN,COMMA	**83.75**	**83.25**	83.50	28.9	261,968
−RULE,DIST,SPAN,COMMA	**83.44**	**82.93**	83.18	27.6	259,372
−WORD,LE	**86.40**	**85.81**	86.10	34.7	25,429
−WORD,POS	**85.44**	**84.87**	85.15	32.7	40,102
−WORD,POS,LE	**84.68**	**84.12**	84.40	31.1	8,899
−SYM,WORD,POS,LE	**82.77**	**82.14**	82.45	24.9	1,914
None	**78.10**	**77.39**	77.74	18.3	0

Table 6.24: Accuracy for covered/uncovered sentences

	LP	LR	F-score	sentence acc.	# sentences
covered sentences	89.36	88.96	89.16	42.2	1,825
uncovered sentences	75.57	74.04	74.80	2.5	319

Table 6.25: Corpus size vs. accuracy

# sentences	LP	LR	F-score	Sentence acc.
1,000	80.23	79.37	79.80	25.2
2,000	81.68	81.07	81.37	27.1
4,000	83.84	83.29	83.56	30.7
8,000	84.88	84.39	84.63	33.6
12,000	85.75	85.26	85.50	34.6
16,000	85.66	85.16	85.41	34.6
24,000	86.28	85.74	86.01	35.4
32,000	86.67	86.06	86.36	36.0
39,832	86.92	86.28	86.60	36.3

6.3.4 Factors for Parsing Accuracy

Table 6.24 shows parsing accuracy for covered and uncovered sentences. It reveals clear differences in accuracy between covered/uncovered sentences. The accuracy for covered sentences is around 2.5 points higher than the overall accuracy, while the accuracy is more than 10 points lower for uncovered sentences. This result indicates improvement of sentential coverage is an important factor for higher accuracy.

Table 6.25 and Figure 6.10 show the learning curve. A feature set was fixed, while the parameter of the Gaussian prior was optimized for each model. High accuracy is attained even with small data, and the accuracy seems to be saturated. This indicates that we cannot further improve the accuracy simply by increasing the size of training data. The exploration of new types of features is necessary for higher accuracy.

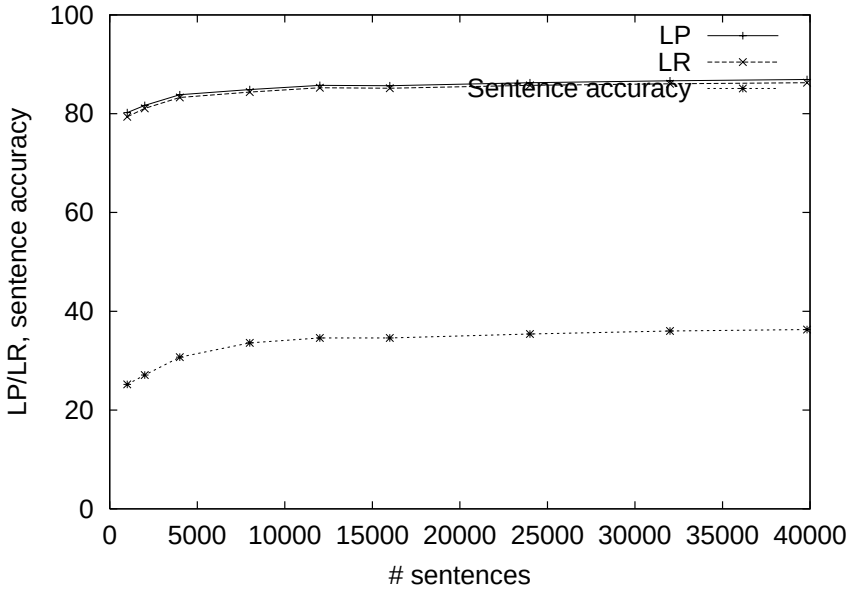


Figure 6.10: Corpus size vs. accuracy

Table 6.26: Sentence length vs. accuracy

Sentence length	LP	LR	F-score	# sentences
0–2	100.00	100.00	100.00	19
3–5	97.11	95.73	96.42	90
6–8	91.74	91.48	91.61	111
9–11	92.11	91.95	92.03	179
12–14	88.81	87.75	88.28	194
15–17	87.26	86.93	87.10	247
18–20	87.60	87.38	87.49	251
21–23	86.96	86.44	86.70	233
24–26	86.36	85.34	85.85	217
27–29	85.13	84.62	84.88	200
30–32	86.32	85.57	85.94	162
33–35	85.91	85.48	85.70	107
36–38	87.11	85.85	86.47	103
39–41	84.42	84.09	84.26	75
42–44	84.89	82.65	83.76	36
45–47	83.71	83.16	83.44	35
48–50	86.47	85.71	86.09	11
51–53	82.90	83.28	83.09	12
54–56	77.34	77.34	77.34	7
57–59	52.14	51.69	51.91	2
60–62	82.85	83.39	83.12	5
63–65	86.36	85.71	86.04	2

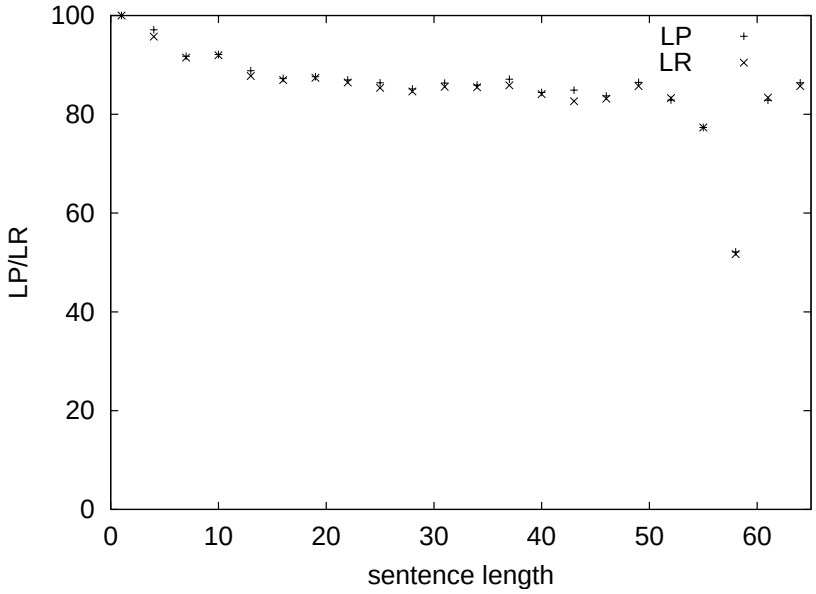


Figure 6.11: Sentence length vs. accuracy

Table 6.26 and Figure 6.11 show the accuracy for each sentence length. It is apparent from this figure that the accuracy is significantly higher for shorter sentences ($<$ 10 words). This implies that experiments with only short sentences overestimate the performance of parsers. Sentences with at least 10 words are necessary to properly evaluate the performance of parsing real-world texts. However, the accuracy for longer sentences is not very different from that for shorter sentences, although data points for sentences with more than 50 words are not reliable.

6.3.5 Analysis of Disambiguation Errors

Table 6.27 shows manual classification of causes of disambiguation errors in 100 sentences randomly chosen from Section 00. In our evaluation, one error source may cause multiple errors of dependencies. For example, if an incorrect lexical entry is assigned to a verb, all of the argument dependencies of the verb are counted as errors. The numbers in the table include such double-counting. Figure6.12 shows examples of disambiguation errors. The figure shows outputs by the parser.

Major causes are classified into three types: attachment ambiguity, argument/modifier distinction, and lexical ambiguity. As attachment ambiguities are well-known error sources, PP-attachment is the largest source of errors in our evaluation. Our disambiguation model cannot accurately resolve PP-attachment ambiguities because it does not include dependencies among a modifiee and the argument of the preposition. Since previous studies revealed that such dependencies are effective features for PP-attachment resolution, we should incorporate them into our model. Some of the attachment ambiguities, including adjective and adverb should also be resolved with an extension of features. However, we cannot identify any effective features for the disambiguation of attachment of verbal phrases, including relative clauses, verb phrases, subordinate clauses, and to-infinitive. For example, Figure 6.12 shows an example error of the attachment of a relative clause. The correct answer is that the subject of "*yielded*" is "*acre*", but this cannot be determined only by the relation among "*yield*", "*grapes*", and "*acre*." The resolution of these errors requires a novel type of feature functions.

Errors of argument/modifier distinction are peculiar to deep syntactic analysis because arguments

Table 6.27: Classification of disambiguation errors

Error cause		# of errors
Attachment ambiguity	prepositional phrase	32
	relative clause	14
	adjective	7
	adverb	6
	verb phrase	5
	subordinate clause	3
	to-infinitive	3
	others	6
Argument/modifier distinction	to-infinitive	19
	noun phrase	7
	verb phrase	7
	subordinate clause	7
	others	9
Lexical ambiguity	preposition/modifier	13
	verb subcategorization frame	13
	participle/adjective	12
	others	6
Test set errors	Errors of treebank conversion	18
	Errors of Penn Treebank	4
Comma		32
Noun phrase identification		15
Coordination/insertion		15
Zero-pronoun resolution		9
Others		4

and modifiers are not distinguished in the evaluation of CFG parsers. Figure 6.12 shows an example of the argument/modifier distinction of a to-infinitive clause. In this case, the to-infinitive clause is a complement of "*tempts*." The subcategorization frame of "*tempts*" seems responsible for this problem. However, the disambiguation model wrongly assigned a template for a transitive verb because of the sparseness of the training data ("*tempts*" occurred only once in the training data). The resolution of this sort of ambiguity requires the refinement of a probabilistic model of lexical entries. Errors of verb phrases and subordinate clauses are similar to this example. Errors of argument/modifier distinction of noun phrases are mainly caused by temporal nouns and cardinal numbers. The resolution of these errors seems to require the identification of temporal expressions and usage of cardinal numbers.

Errors of lexical ambiguities were mainly caused by idioms, as shown in Figure 6.12: "*compared with*" is a compound preposition, but the parser recognized it as a verb phrase. This indicates that the grammar or the disambiguation model requires special treatment of idioms. Errors of verb subcategorization frames were mainly caused by difficult constructions like insertions. Figure 6.12 shows that the parser could not identify the inserted clause ("*says John Siegel*...") and a lexical entry for a declarative transitive verb was chosen.

Attachment errors of commas are also significant, but can be ignored because they do not contribute to the computation of semantics. It should also be noted that commas were ignored in the

Attachment of a subordinate clause It's made only in years when the grapes ripen perfectly (the last was 1979) and comes from a single acre of [NP grapes [S' that yielded a mere 75 cases in 1987]].
Argument/modifier distinction of a to-infinitive More than a few CEOs say the red-carpet treatment tempts them [VP-modifier to return to a heartland city for future meetings].
Prepositional phrase or modifier Mitsui Mining & Smelting Co. posted a 62 % rise in pretax profit to 5.276 billion yen ($ 36.9 million) in its fiscal first half ended Sept. 30 [VP compared with 3.253 billion yen a year earlier].
Wrong assignment of a subcategorization frame [NP-subject "Nasty innuendoes,"] [VP says [NP-object John Siegal, Mr. Dinkins's issues director, "designed to prosecute a case of political corruption that simply doesn't exist."]]

Figure 6.12: Examples of disambiguation errors

evaluation of CFG parsers because the annotation of commas in Penn Treebank is not consistent. We did not eliminate punctuation from the evaluation since punctuation sometimes contributes to semantics, as in coordination and insertion. Errors of commas representing coordination/insertion are classified into "coordination/insertion" in the table.

Errors of "noun phrase identification" mean that a noun phrase was split into two phrases. These errors were mainly caused by the indirect effects of other errors.

Errors of identifying coordination/insertion structures sometimes resulted in catastrophic analyses. While accurate analysis of such constructions is indispensable, it is also known to be difficult because disambiguation of coordination/insertion requires the computation of preferences of global structures, such as similarity of syntactic/semantic structures of coordinates. Incorporating features for representing similarity of global structures is difficult for feature forest models.

Zero-pronoun resolution is also a difficult problem. However, we found that most were indirectly caused by errors of argument/modifier distinction of to-infinitive clauses.

A significant portion of the errors discussed above cannot be resolved by the features we investigated in this study, and the design of other features will be necessary for improving parsing accuracy.

6.4 Discussion and Related Work

Table 6.28 summarizes the best performance shown in this chapter by the HPSG parser with the grammar G_3. The parser demonstrates impressively high coverage and accuracy that have hardly been attained by syntactic analysis including HPSG parsing. We therefore conclude that the HPSG parser for English is moving toward a practical level of use in real-world applications.

As discussed in Section 3.5, methods of the extraction of lexicalized grammars from Penn Treebank was studied extensively for LTAG (Chen and Vijay-Shanker, 2000; Chiang, 2000; Xia, 1999), CCG (Hockenmaier and Steedman, 2002a), and LFG (Cahill et al., 2002; Frank et al., 2003b). The grammar coverage reported here is comparable to or superior to their results. Our results additionally demonstrate the practicality and applicability of lexicalized grammars to the analysis of real-world sentences.

Table 6.28: Final results

Coverage against Section 23	
Lexical coverage	99.09%
Sentential coverage	84.1%
Parsing accuracy for Section 23 (< 40 words)	
# parsed sentences	2,137/2,144 (99.7%)
Precision/recall	87.69%/87.16%
Sentential accuracy	39.2%

Experiments on the parsing of Penn Treebank with lexicalized grammars have been reported for CCG (Clark and Curran, 2004b) and LFG (Kaplan et al., 2004). They developed log-linear models on their own packed representations of parse forests, which are essentially based on our work (Miyao and Tsujii, 2002). Although HPSG exploits further complicated feature constraints and requires high computational cost, our work has proved that feature forest models can be applied to HPSG parsing and attain accurate and wide-coverage parsing.

From the extensive investigation of HPSG parsing, we observed that exploration of new types of features is indispensable to the further improvement of parsing accuracy. A possible research direction is to encode larger contexts of parse trees, which have been shown to improve accuracy (Toutanova et al., 2004; Toutanova and Manning, 2002). Future work includes not only the investigation of these features but also the abstraction of predicate argument dependencies using semantic classes. Experimental results also suggest that an improvement of grammar coverage is crucial for higher accuracy. This indicates that an improvement in the quality of a grammar is a key factor for the improvement of parsing accuracy.

Chapter 7

Conclusions

In this thesis, we explored methods for establishing a syntactic analyzer based on HPSG to process real-world texts. Two obstacles to this goal were discussed: the scalability of grammar development and probabilistic modeling of structured data.

The first problem was the difficulty in expanding a lexicalized grammar to a practical level. Wide-coverage lexicalized grammars have required comprehensive rules/lexical entries, while they have also involved complicated representations of in-depth syntactic constraints. The scaling up of lexicalized grammars has been almost infeasible because the maintenance of a grammar becomes extremely difficult as the grammar grows larger.

A new methodology, *corpus-oriented development of lexicalized grammars*, provided a solution to the systematic control of consistency of a large grammar. The method is corpus-oriented, in the sense that the target of the development is a treebank rather than a lexicon. Grammar writers define linguistic principles in conformity with a syntactic theory, and convert existing linguistic resources into a treebank of the target syntactic theory. Linguistic principles are exploited to validate consistency of a grammar. That is, inconsistencies are automatically detected as violations of applications of principles to the treebank. This means that consistency maintenance is inherently involved in the development process.

The second problem was the difficulty in the probabilistic modeling of structured data that cannot be decomposed into independent probabilistic events. Statistical models are necessary for the practical application of large-scale lexicalized grammars because the grammars produce ambiguous parse results but applications usually require disambiguated or ranked results. However, probabilistic modeling of lexicalized grammars has not been straightforward because popular methods of probabilistic models cannot be applied. Lexicalized grammars express in-depth syntactic constraints with expressive data structures such as typed feature structures. Since typed feature structures are graph structures that may include reentrant structures, they cannot be decomposed into atomic probabilistic events under the assumption of statistical independence.

The solution proposed here was *the feature forest model*. Feature forests are generic data structures to express an exponential number of trees in a packed forest structure. If probabilistic events are represented with feature forests, maximum entropy models are estimated without any independence assumption. The essential idea is an algorithm of estimating parameters of maximum entropy models without unpacking feature forests. A dynamic programming algorithm was proposed for the computation of model expectations by traversing nodes in feature forests. Proof of the correctness of the algorithm was also presented. We successfully applied feature forest models to the probabilistic modeling of HPSG parsing by demonstrating that parse trees of HPSG and predicate argument structures are represented by feature forests.

The proposals in this dissertation were implemented in the development of an HPSG parser for

English. The parser could parse 99.7% of unseen sentences from Penn Treebank Section 23. The accuracy of identifying predicate argument dependencies was 87.69% precision and 87.16% recall. The results are comparative to or better than state-of-the-art parsers and it is concluded that the HPSG parser is at a practical level.

The following sections present achievements of this thesis in each sub-topic.

7.1 Corpus-Oriented Development of Lexicalized Grammars

The principal idea proposed in this thesis for grammar development was to develop a treebank of a target grammar theory, and to collect a lexicon from the treebank. Grammar developers formulate generic structures and constraints of a grammar as linguistic principles, and develop empirical annotation rules to convert an existing treebank into a treebank of the target grammar theory. The consistency of annotated constraints is verified by applying linguistic principles to the treebank. A lexicon is then obtained by collecting terminal nodes of parse trees in the treebank.

This strategy was applied to the development of a wide-coverage HPSG grammar of English. We defined 12 schemas following the definition of Pollard and Sag (1994), and developed 343 annotation rules for converting Penn Treebank into an HPSG treebank. A lexicon was successfully collected from 37,886 sentences out of 39,832 sentences in Sections 02-21 of the Penn Treebank. The obtained lexicon achieved impressively high lexical coverage (99.15%) and sentential coverage (84.8%) for unseen sentences (Section 23 of the Penn Treebank).

The results of the experiments reveal that our grammar is sufficiently robust and accurate for the analysis of real-world texts. This demonstrates that corpus-oriented development is a promising alternative to manual development of a lexicon. Our claim is that the fragility of syntactic analysis was the result of difficulties with the development of a wide-coverage grammar based on syntactic theories, as opposed to most researchers who believe in the inherent impossibility of syntactic theories for the processing of real-world texts. This study enabled us to develop and maintain wide-coverage grammars based on syntactic theories, and opened up the possibility of in-depth syntactic analysis of real-world texts.

The concept of corpus-oriented development of lexicalized grammars is summarized in the following three points. First, grammar writers define linguistic principles and develop a treebank instead of a lexicon. Second, they verify the consistency of the treebank using the linguistic principles, and collect a lexicon from the treebank. Third, they continuously refine the linguistic principles and the treebank through the consistency verification. Through the cycle of the adjustment of a treebank and the assessment by linguistic principles, grammar developers improve the treebank in conformity with the target grammar theory.

This methodology will be extended to the implementation of massive knowledge other than lexicalized grammars. We suspect that the difficulties in the implementation of knowledge have been caused by the ambivalence of complicated structure and exhaustive distribution. That is, knowledge must be represented with complicated structure in general, while real-world applications require knowledge of exhaustive entities. The manual development was interested in the former property, while the machine learning from real data focused on the latter. The development of lexicalized grammars involved this problem, and we solved the problem by the construction of a treebank and consistency verification by linguistic principles. Treebank development was a process of the externalization of complicated metaphysical structures of massive sentences, and the consistency verification helped the manual management of the formulation of the treebank. The success of corpus-oriented development reveals that exhaustive knowledge with complicated structure can be implemented through the cycle of externalizing metaphysical structure of instance phenomena and the verification by generic

principles.

7.2 Feature Forest Model and Probabilistic HPSG Parsing

The proposal for the probabilistic modeling of lexicalized grammars was the feature forest model. Feature forests are generic data structures to represent exponentially many tree structures. When ambiguous probabilistic events are represented by feature forests, parameters of maximum entropy models are estimated in a tractable cost without unpacking feature forests. The essential idea was a dynamic programming algorithm of computing inside/outside α-products, which roughly correspond to inside/outside probabilities of probabilistic context-free grammars.

Feature forest models were applied to the probabilistic modeling of HPSG parse trees and predicate argument structures. It seemed difficult that feature forests represent predicate argument structures including non-local dependencies because feature forests can represent tree structures while predicate-argument structures include reentrant structures. However, we found an algorithm to represent predicate argument structures with feature forests. This demonstrates the applicability of feature forest models to wide-ranging problems.

Disambiguation models of HPSG parsing were trained with the treebank converted from Penn Treebank Sections 02-21. Feature functions of maximum entropy models were designed to represent characteristics of syntactic structures and predicate argument dependencies. The obtained disambiguation models attained 87.69%/87.16% accuracy (labeled precision/recall) for the identification of predicate argument dependencies for unseen sentences in Penn Treebank Section 23. The manual investigation of parsing errors revealed that some of the errors might be reduced by introducing semantic preferences, although the majority of the errors are very difficult to resolve. Invention of new types of feature functions seems necessary to reach a human level.

Feature forest models constitute a new framework of probabilistic modeling in natural language processing. Traditionally, the design of probabilistic models was equal to the decomposition of probabilities into independent sub-events and smoothing of probabilities of sub-events. This was the only solution to cope with structured data of natural languages. However, feature forest models inspire a new insight into the relationship between a linguistic structure and a unit of probabilities. Traditionally, a unit of probability was implicitly assumed to correspond to a meaningful linguistic structure; a tagging to a word or an application of a rewriting rule. This was to estimate plausible probabilistic models. Since a probability is defined over atomic linguistic structures, they should also be meaningful to be assigned a probability. In feature forest models, however, conjunctive nodes are responsible for packing linguistic structures, while feature functions are for probabilistic models. That is, we are free from tying ambiguity packing of linguistic structures with probabilistic modeling. Conjunctive nodes may represent any fragments that are not linguistically meaningful. They should be designed to pack ambiguities and enable us to define useful features. Meanwhile, feature functions should represent linguistically meaningful entities to capture statistical regularity of the target problem. An outcome of feature forest models is the separation of a unit of probability from linguistic structures.

7.3 Future Work

The success of the establishment of an HPSG parser opens up new possibilities of applications of natural language processing. Several projects have already started using the parser/grammar developed in this study. Applications include semantic role labeling (Miyao and Tsujii, 2004), information extraction from biomedical papers (Yakushiji et al., 2005, 2004), sentence generation (Nakanishi et al., 2005), and the automatic construction of semantic representations of dynamic logic. Since these

applications exploit syntactic structures and predicate argument structures, the HPSG parser is a necessary component for them. Furthermore, the HPSG parser will be an important component for intelligent natural language processing systems, including dialog systems, machine translation, and text mining.

In the beginning of this thesis, we discussed the problem of the scalability of syntactic analyzers in comparison with corpus-driven methods. Two antagonisms were described: theory-oriented vs. corpus-driven, and structure vs. statistics. The final problem in this thesis is how we should comprehend the methodology proposed here.

From the viewpoint of corpus-driven statistical methods, corpus-oriented development introduced a model-theoretic strategy for organizing corpus annotations. The proposal was to incorporate a linguistic theory into a cycle of corpus development by exploiting linguistic principles. This concept should be regarded as integrating theoretically motivated rules into annotations to a corpus. In addition, statistical models were successfully applied to typed feature structures. This demonstrates that the application of statistical models should not be limited to problems of simple structures such as classifications and sequence labeling. They may be applied to problems dealing with complicated data structures, such as HPSG parsing.

From the perspective of theory-oriented structuralism, this thesis provided a solution to scaling up syntactic theories to assess real-world texts. While preserving model-theoretic arrangement of grammatical structures, quality and consistency of a grammar are maintained through verification with empirical data. The key idea is to exploit a treebank as a testbed of development and verification, not as training data of statistical models. Since grammar writers can manually maintain a treebank and a grammar through linguistic principles, the heart of theory-based methods has not been lost. In addition, we presented a method of constructing probabilistic models of complex data structures which are required to represent in-depth linguistic constraints. We have had no reason to refuse probabilistic models in syntactic analysis, and feature forest models provide a theoretically sound solution to the problem.

To conclude, this thesis presented a fusion of two main streams of linguistic processing in the context of syntactic analysis. Although strengths have been claimed for each approach, theoretical refusal of either approach has never been established. The two concepts are not conflicting, but focus on different sides of the nature of syntactic analysis. That is, theory-oriented methods are indispensable to represent complicated structure of language, while corpus-driven statistical methods are essential for scaling up linguistic processing. The most significant contribution of this thesis was to exhibit the possibility of integrating the two methodologies. We expect that this consequence will not be limited to syntactic analysis.

Bibliography

Steven P. Abney. Stochastic attribute-value grammars. *Computational Linguistics*, 23(4), 1997.

Hassan Aït-Kaci. *Warren's Abstract Machine: A tutorial reconstruction*. MIT Press, 1991.

J. K. Baker. Trainable grammars for speech recognition. In Jared J. Wolf and Dennis H. Klatt, editors, *Speech communicatioin papers presented at the 97th Meeting of the Acoustical Society of America*, 1979.

J. Baldridge and M. Osborne. Active learning for HPSG parse selection. In *CoNLL-03*, 2003.

Timothy Baldwin, John Beavers, Emily M. Bender, Dan Flickinger, Ara Kim, and Stephan Oepen. Beauty and the Beast: What running a broad-coverage precision grammar over the BNC taught us about the grammar — and the corpus. In *Proceedings of the International Conference on Linguistic Evidence: Empirical, Theoretical, and Computational Perspectives*, 2004a.

Timothy Baldwin, Emily M. Bender, Dan Flickinger, Ara Kim, and Stephan Oepen. Road-testing the English Resource Grammar over the British National Corpus. In *Proceedings of the Fourth International Conference on Language Resource and Evaluation (LREC 2004)*, pages 2047–2050, 2004b.

L. E. Baum and J. A. Eagon. An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model of ecology. *American Mathematical Society Bulletin*, 73:360–363, 1967.

Emily M. Bender and Dan Flickinger. Rapid prototyping of scalable grammars: Towards modularity in extensions to a language-independent core. In *Proceedings of the 2nd International Joint Conference on Natural Language Processing (IJCNLP-05) Posters/Demos*, 2005.

Emily M. Bender, Dan Flickinger, and Stephan Oepen. The Grammar Matrix: An open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. In John Carroll, Nelleke Oostdijk, and Richard Sutcliffe, editors, *Proceedings of the Workshop on Grammar Engineering and Evaluation at the 19th International Conference on Computational Linguistics*, pages 8–14, 2002.

Steven J. Benson and Jorge Moré. A limited-memory variable-metric algorithm for bound-constrained minimization. Technical Report ANL/MCS-P909-0901, Mathematics and Computer Science Division, Argonne National Laboratory, 2001.

Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

Philippe Blache. Disambiguating with controlled disjunctions. In *Proceedings of the 5th International Workshop on Parsing Technologies*, 1997.

Philippe Blache. Parsing ambiguous structures using controlled disjunctions and unary quasi-trees. In *Proceedings of 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (COLING–ACL '98)*, pages 124–130, 1998.

Francis Bond, Sanae Fujita, Chikara Hashimoto, Kaname Kasahara, Shigeko Nariyama, Eric Nichols, Akira Ohtani, Takaaki Tanaka, and Shigeaki Amano. The Hinoki Treebank: A treebank for text understanding. In *Proceedings of the First International Joint Conference on Natural Language Processing*, pages 554–559, Sanya, China, 2004.

Andrew Borthwick. *A maximum entropy approach to named entity recognition*. PhD thesis, New York University, 1999.

Gosse Bouma. Defaults in unification grammar. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, pages 165–172, 1990.

Gosse Bouma, Gertjan van Noord, and Robert Malouf. Alpino: Wide-coverage computational analysis of Dutch. In W. Daelemans, K. Sima'an, J. Veenstra, and J. Zavrel, editors, *Computational Linguistics in the Netherlands 2000*, pages 45–59. Rodopi, 2001.

Joan Bresnan, editor. *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, MA, 1982.

C.G. Broyden. The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematics and its Applications*, 6:76–90, 1970.

Michael Burke, Aoife Cahill, Ruth O'Donovan, Josef van Genabith, and Andy Way. Treebank-based acquisition of wide-coverage, probabilistic LFG resources: Project overview, results and evaluation. In *Proceedings of the IJCNLP-04 Workshop "Beyond Shallow Analyses"*, 2004.

Lou Burnard. User reference guide for the British National Corpus. Technical report, Oxford University Computing Services, 2000.

Miriam Butt, Helge Dyvik, Tracy Holloway King, Hiroshi Masuichi, and Christian Rohrer. The parallel grammar project. In *Proceedings of the COLING-2002 Workshop on Grammar Engineering and Evaluation*, pages 1–7, 2002.

Aoife Cahill, Martin Forst, Mairead McCarthy, Ruth O'Donovan, Christian Rohrer, Josef van Genabith, and Andy Way. Treebank-based multilingual unification-grammar development. In *Proceedings of the Workshop on Ideas and Strategies for Multilingual Grammar Development*, pages 18–29, 2003.

Aoife Cahill, Mairead McCarthy, Josef van Genabith, and Andy Way. Parsing with PCFGs and automatic f-structure annotation. In *Proceedings of the Seventh International Lexical-Functional Grammar Conference*, 2002.

Ulrich Callmeier. PET — a platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering Special Issue – Efficient Processing with HPSG: Methods, Systems, Evaluation*, 6(1):99–108, 2000.

Sharon A. Caraballo and Eugene Charniak. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24(2):275–298, 1998.

Bob Carpenter. *The Logic of Typed Feature Structures*. Cambridge University Press, 1992.

Bob Carpenter. Skeptical and credulous default unification with applications to templates and inheritance. In *Inheritance, Defaults, and the Lexicon*, pages 13–37. Cambridge University Press, 1993.

Bob Carpenter and Gerald Penn. ALE 2.0 user's guide. Technical report, Carnegie Mellon University Laboratory for Computational Linguistics, 1994.

Bob Carpenter and Yan Qu. An abstract machine for attribute-value logics. In *Proceedings of the 4th Internaional Workshop on Parsing Technologies*, 1995.

David Carter. Efficient disjunctiive unification for bottom-up parsing. In *Proceedings of the 13th International Conference on Computational Linguistics*, volume 3, pages 70–75, 1990.

Eugene Charniak. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 598–603, 1997.

Eugene Charniak. A maximum-entropy-inspired parser. In *Proceedings of the First Conference on North American Chapter of the Association for Computational Linguistics (NAACL 2000)*, pages 132–139, 2000.

Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL 2005)*, 2005.

Ciprian Chelba, David Engle, Frederick Jelinek, Victor Jimenez, Sanjeev Khudanpur, Lidia Mangu, Harry Printz, Eric Ristad, Ronald Rosenfeld, Andreas Stolcke, and Dekai Wu. Structure and performance of a dependency language model. In *Proceedings of Eurospeech '97*, 1997.

John Chen and K. Vijay-Shanker. Automated extraction of TAGs from the Penn Treebank. In *Proceedings of the Sixth International Workshop on Parsing Technologies*, 2000.

Stanley Chen and Ronald Rosenfeld. A gaussian prior for smoothing maximum entropy models. Technical Report CMUCS-99-108, Carnegie Mellon University, 1999a.

Stanley F. Chen and Ronald Rosenfeld. Efficient sampling and feature selection in whole sentence maximum entropy language models. In *Proceedings of the 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1999b.

David Chiang. Statistical parsing with an automatically-extracted tree adjoining grammar. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL 2000)*, pages 456–463, 2000.

David Chiang. Mildly context sensitive grammars for estimating maximum entropy parsing models. In *Proceedings of the 8th Conference on Formal Grammar*, 2003.

Mahesh V. Chitrao and Ralph Grishman. Edge-based best-first chart parsing. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 263–266, 1990.

Noam Chomsky. *Syntactic Structures*. The Hague: Mouton, 1957.

Stephen Clark and James R. Curran. Log-linear models for wide-coverage CCG parsing. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing (EMNLP 2003)*, pages 97–104, 2003.

Stephen Clark and James R. Curran. The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING 2004)*, 2004a.

Stephen Clark and James R. Curran. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL 2004)*, 2004b.

Stephen Clark, Julia Hockenmaier, and Mark Steedman. Building deep dependency structures with a wide-coverage CCG parser. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL 2002)*, pages 327–334, 2002.

John Cocke and Jacob T. Schwartz. Programming languages and their compilers: Preliminary notes. Technical report, Courant Institute of Mathematical Sciences, New York University, 1970.

Paul R. Cohen. *Empirical Methods for Artificial Intelligence*. The MIT Press, 1995.

M. Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, Univ. of Pennsylvania, 1999.

Michael Collins. A new statistical parser based on bigram lexical dependencies. In *Proceedings of 34th ACL*, pages 184–191, 1996.

Michael Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL'97)*, pages 16–23, 1997.

Michael Collins. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4):589–637, 2003.

Ann Copestake. Representation issues in semi-automatic acquisition of large lexicons. In *Proceedings of the Third ACL Conference on Applied Natural Language Processing*, pages 88–96, 1992.

Ann Copestake. *The Representation of Lexical Semantic Information*. PhD thesis, University of Sussex, 1993.

Ann Copestake and Dan Flickinger. An open-source grammar development environment and broad-coverage English grammar using HPSG. In *Proceedings of the Second Conference on Language Resources and Evaluation (LREC-2000)*, Athens, Greece, 2000.

Ann Copestake, Dan Flickinger, Rob Malouf, Susanne Riehemann, and Ivan Sag. Translation using minimal recursion semantics. In *Proceedings of the Sixth International Conference on Theoretical and Methodological Issues in Machine Translation (TMI95)*, 1995.

Ann Copestake, Dan Flickinger, Ivan A. Sag, and Carl Pollard. Minimal recursion semantics: An introduction, 1999.

Berthold Crysmann, Anette Frank, Bernd Kiefer, Stefan Mueller, Guenter Neumann, Jakub Piskorski, Ulrich Schaefer, Melanie Siegel, Hans Uszkoreit, Feiyu Xu, Markus Becker, and Hans-Ulrich Krieger. An integrated architecture for shallow and deep processing. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL 2002)*, 2002.

J. N. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. *Annals of Mathematical Statistics*, pages 1470–1480, 1972.

Michael Daum, Kilian A. Foth, and Wolfgang Menzel. Constraint-based integration of deep and shallow parsing techniques. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2003)*, 2003.

Stephen Della Pietra, Vincent Della Pietra, and John Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, 1997.

Jochen Dörre and Andreas Eisele. Feature logic with disjunctive unification. In *Proceedings of the 13th International Conference on Computational Linguistics*, volume 2, pages 100–105, 1990.

J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 6(8):451–455, 1970.

EDR. EDR (Japan Electronic Dictionary Research Institute, Ltd.) electronic dictionary version 1.5 technical guide, 1996. Second edition is available via
`http://www.iijnet.or.jp/edr/E_TG.html`.

Andreas Eisele and Jochen Dörre. Unification of disjunctive feature descriptions. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, pages 286–294, 1988.

Gregor Erbach. Profit — prolog with features, inheritance and templates. In *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics*, 1995.

Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

R. Fletcher. A new approach to variable metric algorithms. *Computer Journal*, 13(3):317–322, 1970.

R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *Computer Journal*, 7: 148–154, 1964.

Dan Flickinger. On building a more efficient grammar by exploting types. In Stephan Oepen, Dan Flickinger, Jun'ichi Tsujii, and Hans Uszkoreit, editors, *Collaborative Language Engineering*, pages 1–17. CSLI Publications, 2002.

Anette Frank. Constraint-based RMRS construction from shallow grammars. In *Proceedings of the 20th International Conference on Computational Linguistics*, 2004.

Anette Frank, Markus Becker, Berthold Crysmann, Bernd Kiefer, and Ulrich Schaefer. Integrated shallow and deep parsing: TopP meets HPSG. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL 2003)*, pages 104–111, 2003a.

Anette Frank, Louisa Sadler, Josef van Genabith, and Andy Way. From treebank resources to LFG f-structures: Automatic f-structure annotation of treebank trees and CFGs extracted from treebanks. In Anne Abeille, editor, *Building and Using Syntactically Annotated Corpora*, pages 367–389. Kluwer Academic Publishers, 2003b.

Stuart Geman and Mark Johnson. Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL 2002)*, 2002.

Daniel Gildea. Corpus variation and parser performance. In *Proceedings of EMNLP 2001*, pages 167–202, 2001.

D. Goldfarb. A family of variable metric updates derived by variational means. *Mathematics of Computing*, 24:23–26, 1970.

Seth David Goldstein. Using an active chart parser to convert any context-free grammar to Backus-Naur Form. Master's thesis, Massachusetts Institute of Technology, January 1988.

Joshua Goodman. Global thresholding and multiple pass parsing. In *Proceedings of the 2nd Conference on Empirical Methods in Natural Language Processing (EMNLP-2)*, pages 11–25, 1997.

Joshua Goodman. Exponential priors for maximum entropy models. In *Proceedings of the Human Language Technology conference / North American chapter of the Assocation for Computational Linguistics annual meeting*, 2004.

John Griffith. Optimizing feature structure unification with dependent disjunctions. In *Proceedings of the Workshop on Grammar Formalism for NLP at ESSLLI-94*, pages 37–59, 1995.

John Griffith. Modularizing contexted constraints. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING '96)*, pages 448–453, 1996.

Kôiti Hasida. Conditioned unification for natural language processing. In *Proceedings of the 11th International Conference on Computational Linguistics*, pages 85–87, 1986.

L. Hellan and P. Haugereid. The NorSource Grammar — an excercise in the Matrix Grammar building design. In *Proceedings of the ESSLLI Workshop on Ideas and Strategies for Multilingual Grammar Development*, 2003.

Julia Hockenmaier. Parsing with generative models of predicate-argument structure. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL 2003)*, pages 359–366, 2003.

Julia Hockenmaier and Mark Steedman. Acquiring compact lexicalized grammars from a cleaner treebank. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC-2002)*, Las Palmas, Spain, 2002a.

Julia Hockenmaier and Mark Steedman. Generative models for statistical parsing with Combinatory Categorial Grammar. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL 2002)*, pages 335–342, 2002b.

Mark Johnson, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. Estimators for stochastic "unification-based" grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL'99)*, pages 535–541, 1999.

Mark Johnson and Stefan Riezler. Exploiting auxiliary distributions in stochastic unification-based grammars. In *Proceedings of the First Conference on North American Chapter of the Association for Computational Linguistics*, 2000.

Yoshitaka Kameya and Taisuke Sato. Efficient EM learning with tabulation for parameterized logic programs. In *Proceedings of the 1st International Conference on Computational Logic (CL2000)*, volume 1861 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 269–294, 2000.

Hiroshi Kanayama, Kentaro Torisawa, Yutaka Mitsuishi, and Jun'ichi Tsujii. A hybrid japanese parser with hand-crafted grammar and statist ics. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING 2000)*, pages 411–417, 2000.

Ronald M. Kaplan, Stefan Riezler, Tracy H. King, John T. Maxwell III, Alexander Vasserman, and Richard Crouch. Speed and accuracy in shallow and deep stochastic parsing. In *Proceedings of the Human Language Technology Conference and the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2004)*, 2004.

T. Kasami. An efficient recognition and syntax algorithm for context-fre languages. Technical Report AFCRL-65-758, Air Force Cambrige Research Lab., Bedford, Mass., 1965.

Robert Kasper, Bernd Kiefer, Klaus Netter, and K. Vijay-Shanker. Compilation of HPSG to TAG. In *Proceedings of the 33th Annual Meeting of the Association for Computational Linguistics (ACL '95)*, pages 92–99, 1995.

Robert T. Kasper. A unification method for disjunctive feature descriptions. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, pages 235–242, 1987.

Martin Kay. Algorithm schemata and data structures in syntactic processing. In *Readings in Natural Language Processing*, pages 35–70. Morgan Kaufmann Publishers Inc., 1986.

Jun'ichi Kazama. *Improving Maximum Entropy Natural Language Processing by Uncertainty-aware Extensions and Unsupervised Learning*. PhD thesis, The University of Tokyo, 2004.

Jun'ichi Kazama, Yusuke Miyao, and Jun'ichi Tsujii. A maximum entropy tagger with unsupervised hidden Markov models. In *Proceedings of the Natural Language Processing Pacific Rim Symposium 2001*, 2001.

Jun'ichi Kazama and Jun'ichi Tsujii. Evaluation and extension of maximum entropy models with inequality constraints. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing (EMNLP 2003)*, pages 137–144, 2003.

Jun'ichi Kazama and Jun'ichi Tsujii. Maximum entropy models with inequality constraints: A case study on text categorization. *Machine Learning Journal special issue on Learning in Speech and Language Technologies*, 2005.

Bernd Kiefer, Hand-Ulrich Krieger, John Carroll, and Robert Malouf. A bag of useful techniques for efficient and robust parsing. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL '99)*, pages 473–480, 1999.

Bernd Kiefer and Hans-Ulrich Krieger. A context-free approximation of Head-Driven Phrase Structure Grammar. In *Proceedings of the 6th International Workshop on Parsing Technologies (IWPT'00)*, pages 135–146, 2000.

Paul Kingsbury and Martha Palmer. From Treebank to PropBank. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC-2002)*, Las Palmas, Spain, 2002.

Paul Kingsbury, Martha Palmer, and Mitch Marcus. Adding semantic annotation to the Penn TreeBank. In *Proceedings of the Human Language Technology Conference (HLT-2002)*, San Diego, California, 2002.

Dan Klein and Christopher D. Manning. A* parsing: Fast exact viterbi parse selection. In *Proceedings of the Human Language Technology Conference 2003 (HLT-NAACL 2003)*, 2003.

Esther Konig. An efficient decision algorithm for feature logic. In *Proceedings of the 16th German Conference on Artificial Intelligence*, pages 255–266, August 1992.

Valia Kordoni and Julia Neu. Deep grammar development for Modern Greek. In *Proceedings of the ESSLLI Workshop on Ideas and Strategies for Multilingual Grammar Development*, pages 65–72, 2003.

Taku Kudo and Yuji Matsumoto. Japanese dependency analysis based on support vector machines. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000)*, 2000.

Taku Kudo and Yuji Matsumoto. Japanese dependency analysis using cascaded chunking. In *Proceedings of the Sixth Conference on Natural Language Lerning (CoNLL-2002)*, 2002.

John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning 2001*, 2001.

K. Lari and S. J. Young. The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

Alex Lascarides, Ted Briscoe, Nicolas Asher, and Ann Copestake. Order independent and persistent typed default unification. *Linguistics and Philosophy*, 19:1–89, 1996.

Alex Lascarides and Ann Copestake. Default representation in constraint-based frameworks. *Computational Linguistics*, 25(1):55–105, 1999.

Yang Liu, Qun Liu, and Shouxun Lin. Log-linear models for word alignment. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL 2005)*, 2005.

David M. Magerman. Statistical decision-tree models for parsing. In *Proceedings of 33rd ACL*, pages 276–283, 1995.

Takaki Makino, Yusuke Miyao, Kentaro Torisawa, and Jun'ichi Tsujii. Native-code compilation of feature structures. In Stephen Oepen, Dan Flickinger, Jun'ichi Tsujii, and Hans Uszkoreit, editors, *Collaborative Language Engineering: A Case Study in Efficient Grammar-based Parsing*. CSLI Publications, 2002.

Takaki Makino, Kentaro Torisawa, and Jun'ichi Tsujii. LiLFeS - practical programming language for typed feature structures. In *Proceedings of Natural Language Processing Pacific Rim Symposium (NLPRS) '97*, 1997.

Takaki Makino, Minoru Yoshida, Kentaro Torisawa, and Jun'ichi Tsujii. LiLFeS — towards a practical HPSG parser. In *Proceedings of 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (COLING–ACL '98)*, pages 807–811, 1998.

Robert Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the Sixth Conference on Natural Language Lerning (CoNLL-2002)*, 2002.

Robert Malouf, John Carroll, and Ann Copestake. Efficient feature structure operations without compilation. *Natural Language Engineering Special Issue – Efficient Processing with HPSG: Methods, Systems, Evaluation*, 6(1):29–46, 2000.

Robert Malouf and Gertjan van Noord. Wide coverage parsing with stochastic attribute value grammars. In *Proceedings of the IJCNLP-04 Workshop "Beyond Shallow Analyses"*, 2004.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The Penn Treebank: Annotating predicate argument structure. In *ARPA Human Language Technology Workshop*, 1994.

John T. Maxwell III and Ronald M. Kaplan. A method for disjunctive constraint satisfaction. In Mary Dalrymple, Ronald M. Kaplan, John T. Maxwell III, and Annie Zaenen, editors, *Formal Issues in Lexical-Functional Grammar*, number 47 in CSLI Lecture Notes Series, chapter 14, pages 381–481. CSLI Publications, 1995.

Andrew McCallum and Wei Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the 7th Conference on Natural Language Learning (CoNLL)*, 2003.

Andrew McCallum, Khashayar Rohanimanesh, and Charles Sutton. Dynamic conditional random fields for jointly labeling multiple sequences. In *Workshop on Syntax, Semantics, Statistics; 16th Annual Conference on Neural Information Processing Systems (NIPS 2003)*, 2004.

George Miller. Five papers on WordNet. *Special Issue of International Journal of Lexicography*, 3 (4), 1990.

Yutaka Mitsuishi. A disambiguation method for the Head-driven Phrase Structure Grammar. Master's thesis, University of Tokyo, February 1998.

Yutaka Mitsuishi, Kentaro Torisawa, and Jun'ichi Tsujii. HPSG-style underspecified japanese grammar with wide coverage. In *Proceedings of 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (COLING–ACL '98)*, pages 876–880, August 1998.

Yusuke Miyao. Packing of feature structures for efficient unification of disjunctive feature structures. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL '99)*, pages 579–584, 1999.

Yusuke Miyao, Takaki Makino, Kentaro Torisawa, and Jun'ichi Tsujii. The lilfes abstract machine and its evaluation with the lingo grammar. *Natural Language Engineering Special Issue – Efficient Processing with HPSG: Methods, Systems, Evaluation*, pages 47–61, 2000.

Yusuke Miyao, Takashi Ninomiya, and Jun'ichi Tsujii. Lexicalized grammar acquisition. In *Companion Volume to the Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL03)*, pages 127–130, 2003a.

Yusuke Miyao, Takashi Ninomiya, and Jun'ichi Tsujii. Probabilistic modeling of argument structures including non-local dependencies. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2003)*, pages 285–291, 2003b.

Yusuke Miyao, Takashi Ninomiya, and Jun'ichi Tsujii. Corpus-oriented grammar development for acquiring a Head-driven Phrase Structure Grammar from the Penn Treebank. In Keh-Yih Su, Jun'ichi Tsujii, Jong-Hyeok Lee, and Oi Yee Kwong, editors, *Natural Language Processing - IJCNLP 2004*, volume 3248 of *LNAI*, pages 684–693. Springer-Verlag, 2005.

Yusuke Miyao, Kentaro Torisawa, Yuka Tateisi, and Jun'ichi Tsujii. Packing of feature structures for optimizing the HPSG-style grammar translated from TAG. In *Proceedings of the TAG+4 Workshop*, pages 104–107, 1998.

Yusuke Miyao and Jun'ichi Tsujii. Maximum entropy estimation for feature forests. In *Proceedings of the Human Language Technology Conference (HLT-2002)*, 2002.

Yusuke Miyao and Jun'ichi Tsujii. A model of syntactic disambiguation based on lexicalized grammars. In *Proceedings of the Seventh Conference on Natural Language Learning (CoNLL)*, pages 1–8, 2003.

Yusuke Miyao and Jun'ichi Tsujii. Deep linguistic analysis for the accurate identification of predicate-argument relations. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING 2004)*, pages 1392–1397, 2004.

Yusuke Miyao and Jun'ichi Tsujii. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL 2005)*, pages 83–90, 2005.

Stefan Müller and Walter Kasper. HPSG analysis of German. In Wolfgang Wahlster, editor, *Verbmobil: Foundations of Speech-to-Speech Translation*, Artificial Intelligence, pages 238–253. Springer-Verlag, 2000.

Hiroko Nakanishi, Yusuke Miyao, and Jun'ichi Tsujii. An empirical investigation of the effect of lexical rules on parsing with a treebank grammar. In *Proceedings of the Third Workshop on Treebanks and Linguistic Theories*, pages 103–114, 2004a.

Hiroko Nakanishi, Yusuke Miyao, and Jun'ichi Tsujii. Using inverse lexical rules to acquire a wide-coverage lexicalized grammar. In *Proceedings of the IJCNLP-04 Workshop on "Beyond Shallow Analyses"*, 2004b.

Hiroko Nakanishi, Yusuke Miyao, and Jun'ichi Tsujii. Probabilistic models for disambiguation of an HPSG-based chart generator. In *Proceedings of the 9th International Workshop on Parsing Technologies (IWPT 2005)*, 2005.

Mikio Nakano. Constraint projection: An efficient treatment of disjunctive feature descriptions. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 307–314, 1991.

Takashi Ninomiya, Yusuke Miyao, and Jun'ichi Tsujii. Lenient default unification for robust processing within unification based grammar formalisms. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002)*, pages 744–750, 2002.

Takashi Ninomiya, Yoshimasa Tsuruoka, Yusuke Miyao, and Jun'ichi Tsujii. Efficacy of beam thresholding, unification filtering and hybrid parsing in probabilistic HPSG parsing. In *Proceedings of the 9th International Workshop on Parsing Technologies*, 2005.

Kenji Nishida, Kentaro Torisawa, and Jun'ichi Tsujii. Efficient HPSG parsing algorithm with array unification. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium*, pages 144–149, 1999.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 1999.

Jorge Nocedal. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation*, 35:773–782, 1980.

Stephan Oepen and John Carroll. Performance profiling for parser engineering. *Natural Language Engineering*, 6 (1) (Special Issue on Efficient Processing with HPSG):81–97, 2000.

Stephan Oepen, Dan Flickinger, and Francis Bond. Towards holistic grammar engineering and testing — grafting treebank maintenance into the grammar revision cycle. In *Proceedings of the IJCNLP-04 Workshop "Beyond Shallow Analyses"*, 2004.

Stephan Oepen, Dan Flickinger, Jun'ichi Tsujii, and Hans Uszkoreit, editors. *Collaborative Language Engineering: A Case Study in Efficient Grammar-Based Processing*. CSLI Publications, 2002a.

Stephan Oepen, Kristina Toutanova, Stuart Shieber, Christopher Manning, Dan Flickinger, and Thorsten Brants. The LinGO, Redwoods treebank. motivation and preliminary applications. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002)*, 2002b.

Miles Osborne. Estimation of stochastic attribute-value grammar using an informative sample. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING 2000)*, 2000.

Fuchun Peng and Andrew McCallum. Accurate information extraction from research papers using conditional random fields. In *Proceedings of Human Language Technology Conference and North American Chapter of the Association for Computational Linguistics (HLT/NAACL-04)*, 2004.

David Pinto, Andrew McCallum, Xen Lee, and W. Bruce Croft. Table extraction using conditional random fields. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2003)*, 2003.

Carl Pollard and Ivan A. Sag. *Information-Based Syntax and Semantics, Volume 1: Fundamentals*. CSLI Lecture Notes no. 13. University of Chicago Press, 1987.

Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, 1994.

Allan Ramsay. Disjunction without tears. *Computational Linguistics*, 16(3):171–174, March 1990.

Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, 1996.

Adwait Ratnaparkhi. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the 1997 Conference on Empirical Methods in Natural Language Processing*, 1997.

Adwait Ratnaparkhi. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34(1-3):151–175, 1999.

Adwait Ratnaparkhi, Jeff Reynar, and Salim Roukos. A maximum entropy approach for prepositional phrase attachment. In *Proceedings of the ARPA Workshop on Human Language Technology*, 1994.

Stefan Riezler, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell III, and Mark Johnson. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL 2002)*, 2002.

Stefan Riezler, Detlef Prescher, Jonas Kuhn, and Mark Johnson. Lexicalized stochastic modeling of constraint-based grammars using log-linear measures and EM training. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL 2000)*, 2000.

Stefan Riezler and Alexander Vasserman. Incremental feature selection and l_1 regularization for relaxed maximum-entropy modeling. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP 2004)*, 2004.

Brian Roark. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2):249–276, 2001.

Brian Roark, Murat Saraclar, Michael Collins, and Mark Johnson. Discriminative language modeling with conditional random fields and the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL 2004)*, 2004.

Ronald Rosenfeld. A whole sentence maximum entropy language model. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding*, 1997.

W. C. Rounds and R. T. Kasper. A complete logical calculus for record structures representing linguistic information. In *Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science*, 1986.

Graham Russell, John Carroll, and Susan Warwick-Armstrong. Multiple default inheritance in a unification-based lexicon. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 215–221, 1991.

Ivan A. Sag. English relative clause constructions. *Journal of Linguistics*, 33:431–483, 1997.

Ivan A. Sag and Thomas Wasow. *Syntactic Theory – A Formal Introduction*. CSLI Lecture Notes no. 92. CSLI Publications, 1999.

Sunita Sarawagi and William W. Cohen. Semi-Markov conditional random fields for information extraction. In *Advances in Neural Information Processing Systems*, 2004.

Taisuke Sato. A generic approach to em learning for symbolic-statistical models. In *Proeedings of ICML05 Workshop on Learning Language in Logic (LLL05)*, 2005.

Taisuke Sato and Yoshitaka Kameya. PRISM: a language for symbolic-statistical modeling. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI '97)*, pages 1330–1335, 1997.

Taisuke Sato and Yoshitaka Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, 15:391–454, 2001.

Yves Schabes. Stochastic lexicalized tree-adjoining grammars. In *Proceedings of the 14th International Conference on Computational Linguistics*, pages 426–432, 1992.

Yves Schabes, Anne Abeillé, and Aravind K. Joshi. Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammars. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING '88)*, pages 578–583, 1988.

Burr Settles. Biomedical named entity recognition using conditional random fields and rich feature sets. In *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (NLPBA)*, 2004.

Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *Proceedings of the 2003 Human Language Technology Conference and North American Chapter of the Association for Computational Linguistics (HLT/NAACL-03)*, 2003.

D. F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computing*, 24:647–656, 1970.

Melanie Siegel. HPSG analysis of japanese. In Wolfgang Wahlster, editor, *Verbmobil: Foundations of Speech-to-Speech Translation*. Springer Verlag, 2000.

Melanie Siegel and Emily Bender. Efficient deep processing of Japanese. In *Proceedings of the Third Workshop on Asian Language Resources and International Standardization (COLING-2002 Post-Conference Workshop)*, 2002.

Wojciech Skut and Thorsten Brants. A maximum-entropy partial parser for unrestricted text. In *Proceedings of the 6th Workshop on Very Large Corpora (WVLC-6)*, 1998.

Kathrin Spreyer and Anette Frank. Projecting RMRS from TIGER dependencies. In Stefan Müller, editor, *Proceedings of the HPSG 2005 Conference*. CSLI Publications, 2005.

Mark Steedman. *The Syntactic Process*. The MIT Press, 2000.

Charles Sutton, Khashayar Rohanimanesh, and Andrew McCallum. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. In *Proceedings of the 21st International Conference on Machine Learning (ICML 2004)*, 2004.

Ben Taskar, Dan Klein, Michael Collins, Daphne Koller, and Chris Manning. Max-margin parsing. In *EMNLP 2004*, 2004.

Yuka Tateisi, Kentaro Torisawa, Yusuke Miyao, and Jun'ichi Tsujii. Translating the XTAG English grammar to HPSG. In *Proceedings of TAG+4 Workshop*, pages 172–175, 1998.

Kentaro Torisawa, Kenji Nishida, Yusuke Miyao, and Jun'ichi Tsujii. An HPSG parser with CFG filtering. *Natural Language Engineering Special Issue – Efficient Processing with HPSG: Methods, Systems, Evaluation*, 6(1):63–80, 2000.

Kentaro Torisawa and Jun'ichi Tsujii. Computing phrasal-signs in HPSG prior to parsing. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING '96)*, pages 949–955, 1996.

K. Toutanova, P. Markova, and C. Manning. The leaf projection path view of parse trees: Exploring string kernels for HPSG parse selection. In *EMNLP 2004*, 2004.

Kristina Toutanova and Christopher D. Manning. Feature selection for a rich HPSG grammar using decision trees. In *Proceedings of the Sixth Conference on Natural Language Lerning (CoNLL-2002)*, 2002.

Yoshimasa Tsuruoka, Yusuke Miyao, and Jun'ichi Tsujii. Towards efficient probabilistic HPSG parsing: integrating semantic and syntatic preference to guide the parsing. In *Proceedings of the IJCNLP-04 Workshop "Beyond Shallow Analyses"*, 2004.

Yoshimasa Tsuruoka, Yuka Tateishi, Jin-Dong Kim, Tomoko Ohta, John McNaught, Sophia Ananiadou, and Jun'ichi Tsujii. Developing a robust part-of-speech tagger for biomedical text. In *Proceedings of the 10th Panhellenic Conference on Informatics*, 2005.

Kiyotaka Uchimoto, Masaki Murata, Satoshi Sekine, and Hitoshi Isahara. Dependency model using posterior context. In *Proceedings of the 6th International Workshop on Parsing Technologies*, 2000.

Kiyotaka Uchimoto, Satoshi Sekine, and Hitoshi Isahara. Japanese dependency structure analysis based on maximum entropy models. In *Proceedings of the 9th Conference on European Chapter of the Association for Computational Linguistics*, pages 196–203, 1999.

Takehito Utsuro, Takashi Miyata, and Yuji Matsumoto. Maximum entropy model learning of subcategorization preference. In *Proceedings of the 5th Workshop on Very Large Corpora (WVLC-5)*, pages 246–260, 1997.

Peter Lodewijk van Roy. Can logic programming execute as fast as imperative programming? Technical Report CSD-90-600, University of California, Berkeley, 1990.

Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, pages 104–111, 1987.

Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–267, 1967.

David J. Weir. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. PhD thesis, University of Pennsylvania, 1988.

Shuly Wintner and Nissim Francez. Efficient implementation of unification-based grammars. *Journal of Language and Computation*, 1(1):53–92, 1999.

Fei Xia. Extracting tree adjoining grammars from bracketed corpora. In *Proceedings of 5th Natural Language Processing Pacific Rim Symposium (NLPRS '99)*, 1999.

XTAG Research Group. A lexicalized tree adjoining grammar for English. Technical Report IRCS-01-03, IRCS, University of Pennsylvania, 2001.

Akane Yakushiji, Yusuke Miyao, and Jun'ichi Tsujii. Biomedical information extraction with predicate-argument structure patterns. In *Proceedings of the 1st International Symposium on Semantic Mining in Biomedicine*, pages 60–69, 2005.

Akane Yakushiji, Yuka Tateisi, Yusuke Miyao, and Jun'ichi Tsujii. Finding anchor verbs for biomedical IE using predicate-argument structures. In *In the the Companion Volume to the Proceedings of 42nd Annual Meeting of the Association for Computational Linguistics*, pages 157–160, 2004.

Juntae Yoon, Chung hye Han, Nari Kim, and Mee sook Kim. Customizing the XTAG system for efficient grammar development for Korean. In *Proceedings of the 5th International Workshop on Tree Adjoining Grammars and Related Formalisms*, 2000.

Naoki Yoshinaga and Yusuke Miyao. Grammar conversion from LTAG to HPSG. In *Proceedings of the sixth ESSLLI Student Session*, pages 309–324, Helsinki, Finland, August 2001.

Naoki Yoshinaga and Yusuke Miyao. Grammar conversion from LTAG to HPSG. *WEB-SLS: the European Student Journal on Language and Speech*, 2002.

Naoki Yoshinaga, Yusuke Miyao, Kentaro Torisawa, and Jun'ichi Tsujii. Resource sharing among HPSG and LTAG communities by a method of grammar conversion from FB-LTAG to HPSG. In *Proceedings of ACL/EACL Workshop on Sharing Tools and Resources for Research and Education*, pages 39–46, Toulouse, France, July 2001. Morgan Kaufman Publishers.

Naoki Yoshinaga, Yusuke Miyao, Kentaro Torisawa, and Jun'ichi Tsujii. Parsing comparison across grammar formalisms using strongly equivalent grammars. *Journal of Traitement Automatique des Langues*, 44(3):15–39, 2003.

Naoki Yoshinaga, Yusuke Miyao, and Jun'ichi Tsujii. A formal proof of strong equivalence for a grammar conversion from LTAG to HPSG-style. In *roceedings of the sixth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+6)*, pages 187–192, May 2002.

Daniel H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 2(10):189–208, 1967.

Appendix A

Derivation of Maximum Entropy Models

This appendix describes a derivation of a parametric form of maximum entropy models. Recall the definition of maximum entropy models, which was given in Definition 2.13.

Definition A.1 (Maximum entropy model) *Maximum entropy model $p_M(y|x)$ in terms of training data E is defined as follows:*

$$p_M(y|x) = \operatorname*{argmax}_p \left\{ -\sum_{x\in\mathcal{X}} \tilde{p}(x) \sum_{y\in Y(x)} p(y|x) \log p(y|x) \right\},$$

subject to:

$$\forall f_i \in \boldsymbol{f} \qquad \sum_{x\in\mathcal{X}} \tilde{p}(x) \sum_{y\in Y(x)} p(y|x) f_i(x,y) = \sum_{x\in\mathcal{X}} \sum_{y\in Y(x)} \tilde{p}(x,y) f_i(x,y),$$

$$\forall x \in \mathcal{X} \qquad \sum_{y\in Y(x)} p(y|x) = 1.$$

This constrained optimization problem is converted into an unconstrained optimization by the method of Lagrangian multipliers.

First, we introduce the following *Lagrangian*.

$$\begin{aligned}
\Lambda(p, \boldsymbol{\lambda}, \boldsymbol{\kappa}) &= H(p) + \sum_i \lambda_i \{\mu_i - \tilde{\mu}_i\} + \sum_{x\in\mathcal{X}} \kappa_x \left\{ \sum_{y\in Y(x)} p(y|x) - 1 \right\} \\
&= -\sum_{x\in\mathcal{X}} \tilde{p}(x) \sum_{y\in Y(x)} p(y|x) \log p(y|x) \\
&\quad + \sum_i \lambda_i \left\{ \sum_{x\in\mathcal{X}} \tilde{p}(x) \sum_{y\in Y(x)} p(y|x) f_i(x,y) - \sum_{x\in\mathcal{X}} \sum_{y\in Y(x)} \tilde{p}(x,y) f_i(x,y) \right\} \\
&\quad + \sum_{x\in\mathcal{X}} \kappa_x \left\{ \sum_{y\in Y(x)} p(y|x) - 1 \right\}.
\end{aligned}$$

In the equation, $\boldsymbol{\lambda} = \{\lambda_i\}$ and $\boldsymbol{\kappa} = \{\kappa_x\}$ are Lagrange multipliers. The optimum of $\Lambda(p, \boldsymbol{\lambda}, \boldsymbol{\kappa})$ is equal to the solution of the problem in Definition A.1.

Since $\Lambda(p, \boldsymbol{\lambda}, \boldsymbol{\kappa})$ is convex, the optimum of this function is the zero point of partial derivatives. A partial derivative of $\Lambda(p, \boldsymbol{\lambda}, \boldsymbol{\kappa})$ according to $p(y|x)$ is:

$$\begin{aligned}\frac{\partial \Lambda(p, \boldsymbol{\lambda}, \boldsymbol{\kappa})}{\partial p(y|x)} &= -\tilde{p}(x)\left(\log p(y|x) + 1\right) \\ &\quad + \tilde{p}(x)\sum_i \lambda_i f_i(x, y) \\ &\quad + \kappa_x.\end{aligned}$$

Letting the above function be zero, we obtain:

$$-\log p(y|x) - 1 + \sum_i \lambda_i f_i(x, y) + \kappa_x = 0$$

$$\Longleftrightarrow \qquad p(y|x) = \exp\left(\sum_i \lambda_i f_i(x, y) - 1 + \kappa_x\right).$$

From partial derivatives of $\Lambda(p, \boldsymbol{\lambda}, \boldsymbol{\kappa})$ according to κ_x, we obtain:

$$\sum_{y \in Y(x)} p(y|x) - 1 = 0.$$

By substituting $p(y|x)$, this constraint is rewritten as follows:

$$\sum_{y \in Y(x)} \exp\left(\sum_i \lambda_i f_i(x, y) - 1 + \kappa_x\right) - 1 = 0$$

$$\Longleftrightarrow \qquad \exp\left(-1 + \kappa_x\right) \sum_{y \in Y(x)} \exp\left(\sum_i \lambda_i f_i(x, y)\right) - 1 = 0$$

$$\Longleftrightarrow \qquad \exp\left(-1 + \kappa_x\right) = \frac{1}{\sum_{y \in Y(x)} \exp\left(\sum_i \lambda_i f_i(x, y)\right)}.$$

Hence, given parameters $\boldsymbol{\lambda} = \{\lambda_i\}$, $p(y|x)$ is formulated as follows:

$$p(y|x) = \frac{1}{Z(x)} \exp\left(\sum_i \lambda_i f_i(x, y)\right),$$

$$\text{where} \qquad Z(x) = \sum_{y \in Y(x)} \exp\left(\sum_i \lambda_i f_i(x, y)\right).$$

Then, by substituting $p(y|x)$ in $\Lambda(p, \boldsymbol{\lambda}, \boldsymbol{\kappa})$ to the above formula, we obtain:

$$\begin{aligned}\Psi(\boldsymbol{\lambda}) &= -\sum_{x \in \mathcal{X}} \tilde{p}(x) \sum_{y \in Y(x)} p(y|x) \log p(y|x) + \sum_i \lambda_i \left(\mu_i - \tilde{\mu}_i\right) + \sum_{x \in \mathcal{X}} \kappa_x \left(\sum_{y \in Y(x)} p(y|x) - 1\right) \\ &= -\sum_{x \in \mathcal{X}} \tilde{p}(x) \sum_{y \in Y(x)} \frac{1}{Z(x)} \exp\left(\sum_i \lambda_i f_i(x, y)\right) \log \frac{1}{Z(x)} \exp\left(\sum_i \lambda_i f_i(x, y)\right) \\ &\quad + \sum_i \lambda_i \left\{\sum_{x \in \mathcal{X}} \tilde{p}(x) \sum_{y \in Y(x)} f_i(x, y) \frac{1}{Z(x)} \exp\left(\sum_j \lambda_j f_j(x, y)\right) - \tilde{\mu}_i\right\} \\ &= \sum_x \tilde{p}(x) \log Z(x) - \sum_i \lambda_i \tilde{\mu}_i.\end{aligned}$$

$\Psi(\boldsymbol{\lambda})$ is shown to be equal to the negative log-likelihood $-L(p, E)$ of the training data E according to the probabilistic model $p(y|x)$.

$$\begin{aligned}
-L(p,E) &= -\log \prod_{\langle x,y\rangle \in E} p(y|x)^{\tilde{p}(x,y)} \\
&= -\sum_{\langle x,y\rangle \in E} \tilde{p}(x,y) \log p(y|x) \\
&= -\sum_{\langle x,y\rangle \in E} \tilde{p}(x,y) \log \frac{1}{Z(x)} \exp\left(\sum_i \lambda_i f_i(x,y)\right) \\
&= \sum_{\langle x,y\rangle \in E} \tilde{p}(x,y) \log Z(x) - \sum_{\langle x,y\rangle \in E} \tilde{p}(x,y) \sum_i \lambda_i f_i(x,y) \\
&= \sum_{x \in \mathcal{X}} \log Z(x) \sum_{y \in Y(x)} \tilde{p}(x,y) - \sum_i \lambda_i \sum_{\langle x,y\rangle \in E} \tilde{p}(x,y) f_i(x,y) \\
&= \sum_{x \in \mathcal{X}} \tilde{p}(x) \log Z(x) - \sum_i \lambda_i \tilde{\mu}_i \\
&= \Psi(\boldsymbol{\lambda}).
\end{aligned}$$

Hence, maximum entropy models are considered to be the result of optimizing the likelihood of the training data given a set of feature functions and the parametric form in a log-linear formula.

We finally obtain the following formalization of maximum entropy models, which is the same as Definition 2.14.

Definition A.2 (Maximum entropy model (parametric form)) *A maximum entropy model is defined as the solution of the following optimization problem.*

$$p_M(y|x;\boldsymbol{\lambda}) = \operatorname*{argmax}_p \left\{ -\sum_x \tilde{p}(x) \log Z(x) + \sum_i \lambda_i \tilde{\mu}_i \right\},$$

where:

$$p(y|x;\boldsymbol{\lambda}) = \frac{1}{Z(x)} \exp\left(\sum_i \lambda_i f_i(x,y)\right),$$

$$Z(x;\boldsymbol{\lambda}) = \sum_{y \in Y(x)} \exp\left(\sum_i \lambda_i f_i(x,y)\right).$$

Appendix B

Algorithms of Estimation of Maximum Entropy Models

As shown in Appendix A, parameter estimation for maximum entropy models is equivalent to solving the optimum of the following function.

Definition B.1 (Optimization function)

$$L(\boldsymbol{\lambda}) = \left\{ -\sum_x \tilde{p}(x) \log Z(x;\boldsymbol{\lambda}) + \sum_i \lambda_i \tilde{\mu}_i \right\},$$

$$\textit{where} \qquad Z(x;\boldsymbol{\lambda}) = \sum_{y \in Y(x)} \exp\left(\sum_i \lambda_i f_i(x,y) \right).$$

In general, optimization of Definition B.1 does not have a closed form solution. Several numerical algorithms have been proposed for finding model parameters, i.e., $\boldsymbol{\lambda}$.

One family of estimation algorithms is *Iterative Scaling*, such as *Generalized Iterative Scaling (GIS)* (Darroch and Ratcliff, 1972) and *Improved Iterative Scaling (IIS)* (Della Pietra et al., 1997). They are well known algorithms for parameter estimation specialized for maximum entropy estimation. In the Iterative Scaling approach, parameters $\boldsymbol{\lambda}$ are iteratively updated according to the following formula:

$$\lambda_i \leftarrow \lambda_i + \Delta\lambda_i,$$

where $\Delta\lambda_i$ is determined so as to increase the log-likelihood of the training data. The change of the log-likelihood, i.e., $\Delta L(p(y|x;\boldsymbol{\lambda}), E) = L(p(y|x;\boldsymbol{\lambda}+\Delta\boldsymbol{\lambda}), E) - L(p(y|x;\boldsymbol{\lambda}), E)$, is bound as follows:

$$\begin{aligned}
\Delta L(p(y|x;\boldsymbol{\lambda}), E) &= \sum_{\langle x,y\rangle \in E} \tilde{p}(x,y) \log p(y|x;\boldsymbol{\lambda}+\Delta\boldsymbol{\lambda}) - \sum_{\langle x,y\rangle \in E} \tilde{p}(x,y) \log p(y|x;\boldsymbol{\lambda}) \\
&= \sum_{\langle x,y\rangle \in E} \tilde{p}(x,y) \sum_i \Delta\lambda_i f_i(x,y) - \sum_{x \in \mathcal{X}} \tilde{p}(x) \log \frac{Z(x;\boldsymbol{\lambda}+\Delta\boldsymbol{\lambda})}{Z(x;\boldsymbol{\lambda})} \\
&\geq \sum_{\langle x,y\rangle \in E} \tilde{p}(x,y) \sum_i \Delta\lambda_i f_i(x,y) \\
&\quad +1 - \sum_{x \in \mathcal{X}} \tilde{p}(x) \sum_{y \in Y(x)} p(y|x;\boldsymbol{\lambda}) \exp \sum_i \Delta\lambda_i f_i(x,y). \\
&\quad (\text{since } -\log x \geq 1 - x \text{ for all } x > 0)
\end{aligned}$$

Call the last formula $\mathcal{A}(\Delta\boldsymbol{\lambda})$. Since $\mathcal{A}(\Delta\boldsymbol{\lambda})$ is a lower bound of the change of the log-likelihood, the log-likelihood increases when $\mathcal{A}(\Delta\boldsymbol{\lambda}) > 0$.

We further transform the formula to obtain a formula containing only one variable, $\Delta\lambda_i$.

$$\begin{aligned}
& \mathcal{A}(\Delta\boldsymbol{\lambda}) \\
= \; & \sum_{x,y} \tilde{p}(x,y) \sum_i \Delta\lambda_i f_i(x,y) + 1 - \sum_x \tilde{p}(x) \sum_y p(y|x;\boldsymbol{\lambda}) \exp\left(f^{\#}(x,y) \sum_i \Delta\lambda_i \frac{f_i(x,y)}{f^{\#}(x,y)} \right) \\
& \quad \textit{where} \quad f^{\#}(x,y) = \sum_i f_i(x,y).
\end{aligned}$$

Since $\sum_i \frac{f_i(x,y)}{f^{\#}(x,y)} = 1$, we can apply Jensen's inequality.

$$\begin{aligned}
& \mathcal{A}(\Delta\boldsymbol{\lambda}) \\
\geq \; & \sum_{x,y} \tilde{p}(x,y) \sum_i \Delta\lambda_i f_i(x,y) + 1 - \sum_x \tilde{p}(x) \sum_y p(y|x;\boldsymbol{\lambda}) \sum_i \frac{f_i(x,y)}{f^{\#}(x,y)} \exp(\Delta\lambda_i f^{\#}(x,y)).
\end{aligned}$$

Call this new bound $\mathcal{B}(\Delta\boldsymbol{\lambda})$. We choose $\Delta\boldsymbol{\lambda}$ that maximizes the above formula. Let us get partial derivatives of the formula with respect to $\Delta\lambda_i$.

$$\begin{aligned}
& \frac{\partial \mathcal{B}(\Delta\boldsymbol{\lambda})}{\partial \Delta\lambda_i} \\
= \; & \sum_{x,y} \tilde{p}(x,y) f_i(x,y) - \sum_x \tilde{p}(x) \sum_y p(y|x;\boldsymbol{\lambda}) f_i(x,y) \exp\left(\Delta\lambda_i f^{\#}(x,y) \right) \\
= \; & 0.
\end{aligned}$$

Since the above formula includes only $\Delta\lambda_i$ and no other variables, the solution to the equation of each variable is individually computed. Finally, we find the following equation to get $\Delta\lambda_i$ that increases the log-likelihood.

$$\sum_{x,y} \tilde{p}(x,y) f_i(x,y) = \sum_{x,y} \tilde{p}(x) p(y|x) f_i(x,y) \exp\left(\Delta\lambda_i f^{\#}(x,y) \right). \tag{B.1}$$

We can solve Equation B.1 by Newton's method. By factoring the terms having the same $f^{\#}(x,y)$, we define *factored model expectation*, $\mu_{\langle i, f^{\#} \rangle}$, which is the expectation of feature f_i given by the model.

$$\mu_{\langle i, f^{\#} \rangle} = \sum_{x \in \mathcal{X}} \tilde{p}(x) \sum_{y \in Y(x), f^{\#}(x,y) = f^{\#}} f_i(x,y) p(y|x).$$

Then, Equation B.1 is rewritten as follows:

$$\tilde{\mu}_i = \sum_{f^{\#}} \exp\left(\Delta\lambda_i f^{\#} \right) \mu_{\langle i, f^{\#} \rangle}.$$

Since this equation is polynomial, Newton's method can be applied.

The algorithm shown in Figure B.1 implements this. The main part of the algorithm is the computation of factored model expectation. For each iteration, all events in a training data are traversed and model expectations are accumulated into $\mu_{\langle i, f^{\#} \rangle}$.

Input: training data $E = \{\langle x, y\rangle\}$,
feature functions $\boldsymbol{f} = \{f_i\}$, initial parameters $\boldsymbol{\lambda} = \{\lambda_i\}$
Output: optimal parameters $\boldsymbol{\lambda}$

 # Computing relative frequencies and empirical expectations
foreach $\langle x, y\rangle \in E$
  $\tilde{p}(x) \leftarrow \tilde{p}(x) + \frac{1}{|E|}$
  foreach $f_i \in \boldsymbol{f}$ such that $f_i(x, y) \neq 0$
    $\tilde{\mu}_i \leftarrow \tilde{\mu}_i + \frac{1}{|E|} f_i(x, y)$
  end_foreach
end_foreach
 # Main loop of iterative scaling
loop until $\boldsymbol{\lambda}$ converges
  # Computing model expectations
  foreach $\langle x, y\rangle \in E$
    foreach $y' \in Y(x)$
      $f^{\#}(x, y) = \sum_i f_i(x, y)$
      foreach $f_i \in \boldsymbol{f}$ such that $f_i(x, y') \neq 0$

$$\mu_{\langle i, f^{\#}(x,y')\rangle} \leftarrow \mu_{\langle i, f^{\#}(x,y')\rangle} + \tilde{p}(x) f_i(x, y') \frac{1}{Z(x)} \exp\left(\sum_k \lambda_k f_k(x, y')\right)$$

      end_foreach
    end_foreach
  end_foreach
  # Updating parameters
  foreach $f_i \in \boldsymbol{f}$
    Let $\Delta\lambda_i$ be the solution of the following equation:
    $\sum_k \mu_{\langle i,k\rangle} \exp(k\Delta\lambda_i) = \tilde{\mu}_i$
    $\lambda_i \leftarrow \lambda_i + \Delta\lambda_i$
  end_foreach
end_loop

Figure B.1: Improved iterative scaling

Another update formula is derived by transforming $\mathcal{A}(\Delta\boldsymbol{\lambda})$ in a slightly different way. Instead of introducing $f^{\#}(x, y)$, we define a constant, $C = \max_{x,y} f^{\#}(x, y)$. $\mathcal{A}(\Delta\boldsymbol{\lambda})$ is then transformed and bound as follows:

$$\begin{aligned}
& \mathcal{A}(\Delta\boldsymbol{\lambda}) \\
= \; & \sum_{x,y} \tilde{p}(x, y) \sum_i \Delta\lambda_i f_i(x, y) + 1 - \sum_x \tilde{p}(x) \sum_y p(y|x; \boldsymbol{\lambda}) \exp(C \sum_i \Delta\lambda_i \frac{f_i(x, y)}{C}) \\
\geq \; & \sum_{x,y} \tilde{p}(x, y) \sum_i \Delta\lambda_i f_i(x, y) + 1 - \sum_x \tilde{p}(x) \sum_y p(y|x; \boldsymbol{\lambda}) \sum_i \frac{f_i(x, y)}{C} \exp(\Delta\lambda_i C).
\end{aligned}$$

Input: training data $E = \{\langle x, y\rangle\}$,
feature functions $\boldsymbol{f} = \{f_i\}$, initial parameters $\boldsymbol{\lambda} = \{\lambda_i\}$
Output: optimal parameters $\boldsymbol{\lambda}$

 # Computing relative frequencies and empirical expectations
foreach $\langle x, y\rangle \in E$
  $\tilde{p}(x) \leftarrow \tilde{p}(x) + \frac{1}{|E|}$
  foreach $f_i \in \boldsymbol{f}$ such that $f_i(x, y) \neq 0$
    $\tilde{\mu}_i \leftarrow \tilde{\mu}_i + \frac{1}{|E|} f_i(x, y)$
  end_foreach
end_foreach
 # Main loop of iterative scaling
loop until $\boldsymbol{\lambda}$ converges
  # Computing model expectations
  foreach $\langle x, y\rangle \in E$
    foreach $y' \in Y(x)$
      $f^{\#}(x, y) = \sum_i f_i(x, y)$
      foreach $f_i \in \boldsymbol{f}$ such that $f_i(x, y') \neq 0$

$$\mu_i \leftarrow \mu_i + \tilde{p}(x) f_i(x, y') \frac{1}{Z(x)} \exp\left(\sum_k \lambda_k f_k(x, y')\right)$$

      end_foreach
    end_foreach
  end_foreach
  # Updating parameters
  foreach $f_i \in \boldsymbol{f}$
    $\Delta\lambda_i = \frac{1}{f^{\#}} \log\left(\frac{\tilde{\mu}_i}{\mu_i}\right)$
    $\lambda_i \leftarrow \lambda_i + \Delta\lambda_i$
  end_foreach
end_loop

Figure B.2: Generalized iterative scaling

Let $\mathcal{B}'(\Delta\boldsymbol{\lambda})$ denote the last formula. Differentiating $\mathcal{B}'(\Delta\boldsymbol{\lambda})$ with respect to $\Delta\lambda_i$ gives:

$$\begin{aligned} & \frac{\partial \mathcal{B}'(\Delta\boldsymbol{\lambda})}{\partial \Delta\lambda_i} \\ = & \sum_{x,y} \tilde{p}(x, y) f_i(x, y) - \sum_x \tilde{p}(x) \sum_y p(y|x; \boldsymbol{\lambda}) f_i(x, y) \exp\left(\Delta\lambda_i C\right) \\ = & \ 0. \end{aligned}$$

The equation has a closed form solution:

$$\Delta\lambda_i = \frac{1}{C} \log\left(\frac{\tilde{\mu}_i}{\mu_i}\right).$$

This result is the update formula of Generalized Iterative Scaling (GIS). Figure B.2 shows an algorithm of parameter estimation by GIS.

In general, GIS has slower convergence than IIS, but requires less memory and computational cost. This is because IIS requires the computation of factored model expectations. Since model expectations must be stored separately for every $f^{\#}(x, y)$, it requires larger memory when the variation of $f^{\#}(x, y)$ is larger. For example, when we incorporate real-valued features, IIS is not applicable. Malouf (2002) provided empirical comparison of GIS and IIS.

Another family of optimization algorithms is that of gradient-based algorithms, such as the Conjugate Gradient (CG) method (Fletcher and Reeves, 1964) and the limited-memory BFGS (L-BFGS) method (Nocedal and Wright, 1999; Nocedal, 1980). They are general-purpose numerical algorithms for function optimization when a gradient of an objective function is easily computable. In our case, since the gradient of the optimization function ($L(p(y|x; \boldsymbol{\lambda}), E) = L(\boldsymbol{\lambda})$) is computed in a closed form, we can apply gradient-based algorithms. The algorithms are widely used in various tasks not limited to natural language processing because of the fast convergence speed. Recent studies on natural language processing prefer gradient-based algorithms because their convergence speed was proved to be significantly faster than Iterative Scaling methods (Malouf, 2002).

Gradient-based algorithms require the computation of an optimization function and its gradient, i.e., partial derivatives of the function at any point. In our case, the objective function is a log-likelihood of the training data:

$$L(\boldsymbol{\lambda}) \quad = \quad -\sum_{x \in \mathcal{X}} \tilde{p}(x) \log Z(x) + \sum_i \lambda_i \tilde{\mu}_i,$$

and the derivative in terms of λ_i is:

$$\begin{aligned}
\frac{\partial L(\boldsymbol{\lambda})}{\partial \lambda_i} &= -\sum_{x \in \mathcal{X}} \tilde{p}(x) \frac{1}{Z(x)} \frac{\partial Z(x)}{\partial \lambda_i} + \tilde{\mu}_i \\
&= -\sum_{x \in \mathcal{X}} \tilde{p}(x) \frac{1}{Z(x)} \sum_{y \in Y(x)} f_i(x, y) \exp\left(\sum_j \lambda_j f_j(x, y)\right) + \tilde{\mu}_i \\
&= -\sum_{x \in \mathcal{X}} \tilde{p}(x) \sum_{y \in Y(x)} p(y|x) f_i(x, y) + \tilde{\mu}_i \\
&= \tilde{\mu}_i - \mu_i.
\end{aligned}$$

In the following, we denote the gradient, i.e., the vector of partial derivatives, as $\boldsymbol{g}$:

$$\begin{aligned}
\boldsymbol{g} &= \nabla L(\boldsymbol{\lambda}) \\
&= \langle \frac{\partial L(\boldsymbol{\lambda})}{\partial \lambda_1}, \ldots, \frac{\partial L(\boldsymbol{\lambda})}{\partial \lambda_n} \rangle \\
&= \langle \tilde{\mu}_1 - \mu_1, \ldots, \tilde{\mu}_n - \mu_n \rangle.
\end{aligned}$$

The simplest algorithm of gradient-based optimization is *the steepest descent method*. That is, the parameter vector $\boldsymbol{\lambda}$ is updated along the direction of the gradient. When we denote $\boldsymbol{\lambda}$ and $\boldsymbol{g}$ of the k-th iteration as $\boldsymbol{\lambda}^{(k)}$ and $\boldsymbol{g}^{(k)}$, respectively, the update formula is:

$$\boldsymbol{\lambda}^{(k+1)} \leftarrow \boldsymbol{\lambda}^{(k)} - \alpha^{(k)} \boldsymbol{g}^{(k)}.$$

The optimal $\alpha^{(k)}$ is determined with a one-dimensional line search algorithm. However, this method was shown to have very slow convergence because the gradients of each iteration are fixed to one of the two directions.

Input: training data $E = \{\langle x, y \rangle\}$,
feature functions $\boldsymbol{f} = \{f_i\}$, initial parameters $\boldsymbol{\lambda} = \{\lambda_i\}$
Output: optimal parameters $\boldsymbol{\lambda}$

 # Computing relative frequencies and empirical expectations
foreach $\langle x, y \rangle \in E$
  $\tilde{p}(x) \leftarrow \tilde{p}(x) + \frac{1}{|E|}$
  $\tilde{p}(x, y) \leftarrow \tilde{p}(x, y) + \frac{1}{|E|}$
  foreach $f_i \in \boldsymbol{f}$ such that $f_i(x, y) \neq 0$
    $\tilde{\mu}_i \leftarrow \tilde{\mu}_i + \frac{1}{|E|} f_i(x, y)$
  end_foreach
end_foreach
 # Main loop of the conjugate gradient method
loop until $\boldsymbol{\lambda}$ converges
  # Computing model expectations and log-likelihood
  foreach $\langle x, y \rangle \in E$
    foreach $y' \in Y(x)$
      foreach $f_i \in \boldsymbol{f}$ such that $f_i(x, y') \neq 0$

$$\mu_i \leftarrow \mu_i + \tilde{p}(x) f_i(x, y') \frac{1}{Z(x)} \exp\left(\sum_k \lambda_k f_k(x, y')\right)$$

        if $y' = y$ **then** $L \leftarrow L + \tilde{\mu}_i \lambda_i$
      end_foreach
    end_foreach
    $\boldsymbol{g}^{(k)} \leftarrow \tilde{\boldsymbol{\mu}} - \boldsymbol{\mu}$
    $L \leftarrow L - \tilde{p}(x) \log Z(x)$
  end_foreach
  # Updating parameters
  $\boldsymbol{d}^{(k)} \leftarrow -\boldsymbol{g}^{(k)} + \beta_{\text{FR}} \boldsymbol{d}^{(k-1)}$ where $\beta_{\text{FR}} = \dfrac{\boldsymbol{g}^{(k)} \boldsymbol{g}^{(k)}}{\boldsymbol{g}^{(k-1)} \boldsymbol{g}^{(k-1)}}$
  $\alpha^{(k)} \leftarrow$ `line_search`$(\boldsymbol{\lambda}^{(k)}, \boldsymbol{d}^{(k)}, L)$
  $\boldsymbol{d}^{(k)} \leftarrow \alpha^{(k)} \boldsymbol{d}^{(k)}$
  $\boldsymbol{\lambda}^{(k+1)} \leftarrow \boldsymbol{\lambda}^{(k)} + \boldsymbol{d}^{(k)}$
end_loop

Figure B.3: Algorithm of parameter estimation by a Conjugate Gradient method

To overcome the problem of slow convergence, several methods have been proposed in the field of numerical computation. The basic idea was that we should choose a direction orthogonal to previous search directions. Hence, the update formula becomes as follows:

$$\boldsymbol{\lambda}^{(k+1)} \leftarrow \boldsymbol{\lambda}^{(k)} + \alpha^{(k)} \boldsymbol{d}^{(k)},$$

where $\boldsymbol{d}^{(k)}$ is a line search direction that is chosen to be orthogonal in every iteration. The most famous algorithm to determine $\boldsymbol{d}^{(k)}$ is Conjugate Gradient (CG) methods. In this algorithm, the

$\boldsymbol{s}^{(k)} \leftarrow \boldsymbol{\lambda}^{(k)} - \boldsymbol{\lambda}^{(k-1)}$
$\boldsymbol{y}^{(k)} \leftarrow \boldsymbol{g}^{(k)} - \boldsymbol{g}^{(k-1)}$
$\rho^{(k)} \leftarrow \dfrac{1}{\boldsymbol{s}^{(k)}\boldsymbol{y}^{(k)}}$
$\boldsymbol{d}^{(k)} \leftarrow -\boldsymbol{g}^{(k)}$
for $i = k$ `to` $k - m + 1$
  $a^{(i)} \leftarrow \rho^{(i)}\boldsymbol{s}^{(i)}\boldsymbol{d}^{(k)}$
  $\boldsymbol{d}^{(k)} \leftarrow \boldsymbol{d}^{(k)} - a^{(i)}\boldsymbol{y}^{(i)}$
end_for
$\boldsymbol{d}^{(k)} \leftarrow \dfrac{\boldsymbol{s}^{(k)}\boldsymbol{y}^{(k)}}{|\boldsymbol{y}^{(k)}|^2}\boldsymbol{d}^{(k)}$
for $i = k - m + 1$ `to` k
  $\boldsymbol{d}^{(k)} \leftarrow \boldsymbol{d}^{(k)} + (a^{(i)} - \rho^{(i)}\boldsymbol{y}^{(i)}\boldsymbol{d}^{(k)})\boldsymbol{s}^{(i)}$
end_for

Figure B.4: Algorithm of computing a search direction with a limited-memory BFGS method

direction is determined according to the following formula:

$$\boldsymbol{d}^{(k)} \leftarrow -\boldsymbol{g}^{(k)} + \beta\boldsymbol{d}^{(k-1)}.$$

Several methods have been proposed for computing β; the simplest one is called the *Fletcher-Reeves* method (Fletcher and Reeves, 1964) in which β is defined by:

$$\beta_{\text{FR}} = \frac{\boldsymbol{g}^{(k)}\boldsymbol{g}^{(k)}}{\boldsymbol{g}^{(k-1)}\boldsymbol{g}^{(k-1)}}.$$

Other (more complicated) methods are proposed and are shown to achieve better convergence. However, the details are beyond the scope of this thesis. Figure B.3 shows an algorithm of parameter estimation based on the Fletcher-Reeves method.

Another approach is *quasi-Newton* methods, a.k.a., *variable metric* methods. The idea was that they approximate Hessian matrix $H = \nabla^2\Psi(\boldsymbol{\lambda})$, and apply Newton's method in a multi-dimensional space. However, simple application of Newton's method is computationally expensive because it requires computation of inverse Hessian. Quasi-Newton methods thus approximate the inverse Hessian by a matrix that is iteratively updated in each iteration by a certain formula. The most famous formula of updating approximate inverse Hessian is *the BFGS update formula* (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970). However, this method requires large memory because the matrix has $|\boldsymbol{\lambda}|^2$ elements, and it is not applicable to problems with large dimensions.

Nocedal (1980) proposed a new method of approximating inverse Hessian using a small number of vectors. This method, which is called a limited-memory BFGS method (L-BFGS), is widely used in the parameter estimation of maximum entropy models. Most parts of the algorithm are the same as CG methods, while the procedure to compute a search direction, $\boldsymbol{d}^{(k)}$, is different. Figure B.4 shows a pseudo-code of computing $\boldsymbol{d}^{(k)}$ with the L-BFGS method. The method requires to store $2 \times m$ vectors of $|\boldsymbol{\lambda}|$ elements ($\boldsymbol{s}$ and $\boldsymbol{y}$ in the pseudo-code) and m scalars (ρ). Since $m = 5$ is sufficient in general, the method requires much less memory than the original BFGS method.