Collaborative and Corpus-Driven Approaches towards Lexicalized Grammar-based Natural Language Processing

by

Naoki YOSHINAGA

A Dissertation

Submitted to
the Graduate School of
the University of Tokyo
on December, 2004
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Information Science and Technology
in Computer Science

Thesis Supervisor: Jun'ichi TSUJII
Professor of Computer Science

ABSTRACT

This dissertation proposes two approaches for constructing essential components of lexicalized grammar-based natural language processing (NLP). We first provide a framework for developing *static* components such as grammar resources and parsing technologies collaboratively across the grammar formalisms. We second propose two corpus-driven methods of acquiring generic lexical resources, which are *dynamic* in nature, for lexicalized grammars.

The first part of this dissertation proposes grammar conversion as a means of abstracting away surface differences between the individual formalisms, and presents a methodology for collaborative development of the static components of lexicalized grammars. Grammar conversion has been so far proposed for sharing grammar resources between the formalisms. We show that, by constructing *a strongly equivalent grammar* in a particular formalism from one given in another formalism, one can gain a deeper insight into generic parsing techniques that are used for efficient implementation of parsers for the formalisms, through comparison between the parsers using the strongly equivalent grammars. We proposed a grammar conversion from LTAG to HPSG-style grammar, and then conducted two sets of experiments from viewpoints of resource sharing and parsing comparison. A large-scale LTAG grammar, the XTAG English grammar, was successfully converted into an HPSG-style grammar. Empirical comparisons between LTAG and HPSG parsers with dynamic programming and CFG filtering were then conducted using the strongly equivalent grammars, respectively. We thereby suggest a definite way of improving these generic parsing techniques.

The second part of this dissertation concerns two corpus-driven methods of augmenting lexical resources for lexicalized grammars. We first propose a filtering method of subcategorization frames acquired from raw corpora, and acquire a reliable set of SCFs to augment lexicons of lexicalized grammars. We perform clustering of words according to their alternation behaviors, and use the obtained clusters to guide filtering. We applied this filtering method to hand-coded lexicons of lexicalized grammars, and successfully filtered out less plausible SCFs from the noisy SCFs acquired from raw corpora. We second propose a method of constructing a probabilistic lexicon with accurate estimates of co-occurrence probabilities between words and SCF, by using the PLSA model for smoothing the co-occurrence probabilities estimated from raw frequency counts. Given accurate estimates for co-occurrence probabilities between words and SCFs, a reliable set of lexicons can be reconstructed by thresholding. We applied this smoothing method to SCFs for HPSG acquired from annotated corpora, and successfully decreased the test-set perplexity of the co-occurrence probabilities by the interpolated model based on the PLSA model.

Acknowledgements

I spent so many nights in the laboratory; the statistics says it is more than 40% of the nights during my doctor course. When I fell into a restless sleep in a sofa bed, I sometimes dreamed of ideas that finally contributed to the individual algorithms described in this dissertation. I might therefore have to thank the laboratory environment, including the sofa beds and carpets, before individual persons.

Yet first of all, I would like to thank my supervisor, Prof. Jun'ichi Tsujii, for his encouragement, helpful suggestions and criticisms throughout these five years. He showed me a range of possible directions of my research, not only from computational viewpoints but also from linguistic viewpoints. These directions help me to characterize this mysterious study as meaningful. I would also like to express my gratitude to the members of my dissertation committee: Prof. Satoru Miyano (chair), Prof. Masami Hagiya, Prof. Hiroshi Imai, and Prof. Sadao Kurohashi of the University of Tokyo and Prof. Yuji Matsumoto of Nara Institute of Science and Technology, who have been good enough to give this work a very serious review.

I am also indebted to Prof. Kentaro Torisawa of Japan Institute of Science and Technology and Mr. Yusuke Miyao for their comments on approaches to the theme focused in this dissertation. Prof. Kentaro Torisawa inculcated the philosophy as a researcher in me, with which I was impressed much during the master course. Mr. Yusuke Miyao led me to this exciting research area in computational linguistics. Most of the ideas in this dissertation were elaborated through a number of discussions with him.

I next express my gratitude to Dr. Yuka Tateisi and Mr. Takuya Matsuzaki for discussions on technical details and proofreading of this dissertation. Dr. Yuka Tateisi provided me several valuable comments on the first part of this dissertation. Since the research is initially started as an alternative to her study, I could make the research aim clearer through the meaningful discussions with her from linguistic viewpoints. Mr. Takuya Matsuzaki provided useful comments on machine learning techniques which are employed in the latter part of this dissertation to realize my original ideas. Also, I could put my thoughts in order through discussions with him on the research aim and approaches to lexical acquisition for lexicalized grammars.

I also express my gratitude to Dr. Takashi Ninomiya and Dr. Minoru Yoshida for their valuable comments on my research. Dr. Takashi Ninomiya was a person who *always* embodies what a researcher should be, which sometimes enjoys us. Dr. Minoru Yoshida would introduce to us his 'products,' which were sometimes fascinating and relaxed us. I should thank Mr. Yousuke Sakao for his help in profiling an HPSG parser, and thank Dr. Yoshimasa Tsuruoka and Mr. Kazuhiro Yoshida for their advice on probabilistic modeling. I deeply appreciate Ms. Hiroko Nakanishi for her support in using the Enju parser. I would thank Mr. Kenta Oouchida for discussions on parsing algorithms for LTAG. Special thanks are also due to Dr. Takaki Makino, Mr. Yutaka Mitsuishi, Dr. Edson T. Miyamoto, Dr. Jin-Dong Kim, and Dr. Jun'ichi Kazama for their encouragement and corrections for my papers on which this dissertation based. Furthermore, the latter two are my good bicycle friends, and we shared a good time for cycling along with exploring various ramen shops.

Ms. Akane Yakushiji and Mr. Satoshi Simpuku are very good fellows with whom I studied, discussed our research, and enjoyed research life. I and Ms. Akane Yakushiji often had a rapid-fire two-people comedy act; I usually feed her when she talks to herself. Mr. Satoshi Shimpuku is one of my best friends in the laboratory with whom I talked various kinds of matters for the two years during the master course.

Among other laboratory members, I would like to thank Mr. Takashi Tsunakawa, Mr. Yuichiro Matsubayashi, and Mr. Tadayoshi Hara for chatting about several interesting topics together. Mr. Takashi Tsunakawa brought me to various restaurants and hot springs, driving his own car. On Sunday, we enjoyed grade-B gourmet tours, which focused on tasty cuisine at up to 1,000-yen. Mr. Yuichiro Matsubayashi is a real dedicated person, and inspired me to look for 'a new hobby' every-time, which involves magic, cocktail, poem and the like. Mr. Tadayoshi Hara kindly let me enjoy his belongings. I must remember to return them back to him before the graduation.

I thank secretaries in the laboratory. Especially, I owed many things to Ms. Mika Tarukawa and Ms. Minako Ito during my research stay at University of Sussex. Ms. Mika Tarukawa would kindly serve me a good green tea in the morning.

I wish to thank Prof. Aravind Joshi, Dr. Anoop Sarkar, Dr. Rashmi Prasad, Dr. Carlos Prolo, and Dr. Tonia Bleam for supporting and discussing on the LTAG framework during my visit to University of Pennsylvania, September 2000. Especially, I have Dr. Anoop Sarkar to thank for his help in applying his parser as part of our experiments. He accepted many requests for the facility of his parser, and spent his precious time. He was also a good guidance during my research stay at University of Pennsylvania, and kindly let me make copies of precious papers and dissertations.

During 2003 - 2004, I spent 10 months in Brighton, UK as a visiting researcher of University of Sussex, Informatics Department, under the guidance of Prof. John Carroll. The most of the latter part of this dissertation had been worked out during the days I spent in Brighton.

Among others, I would first express my best gratitude to Prof. John Carroll, who invited me to another approaches to lexicalized grammar-based natural language processing, which broadened my research perspectives. I am gratefully to Dr. Diana McCarthy, Dr. Alex Fang, Dr. Rob Koeling, Mr. Xinglong Wang, and other faculty members for helpful discussions, and especially thank Alex Fang for his help in using the acquired lexical resources. I wish to thank a Japanese colleague Prof. Kentaro Inui of Nara Institute of Science and Technology, for variable discussions and letting me use his laptop when my laptop crashed just one day before the deadline of ACL. I am grateful again to the other Japanese colleague Dr. Takaki Makino for enjoying a trip together to London, Seven Sisters Country Park, and southern west of UK by a car. I am also grateful to Dr. Anna Korhonen of University of Cambridge for variable discussions related to subcategorization acquisition.

I can never thank enough for Kate, a landlady of my room, and roommates Cheng and Duan for every-day care, support, and sharing a good living environment. Kate sometimes cooked various kinds of food for me; they are actually the *best* English food I experienced in UK. Cheng brought me to Seven Sisters Country Park, the best place I visited during my research stay. Duan is not only a good chemist but also a good cook of Vietnamese food, and would kindly serve some of the results.

I also acknowledge anonymous reviewers of the journals, conferences and workshops, who gave me helpful comments to refine ideas and experiments, and would like to thank all members of the laboratory for their patience that allows me to stay at the laboratory.

Finally, I thank my parents and my brother for supporting me in these years, and Ms. Chieko Akiyama for encouraging me all the time during the doctor course.

Contents

List of Figures

List of Tables

Introduction

Researchers in the field of Natural Language Processing (NLP) have often argued on the questions concerning whether in-depth syntactic and semantic information is critical to the performance of NLP applications (Yakushiji et al. 2001; Li and Roth 2001; Sebastiani 2002; Chen and Rambow 2003; Huang et al. 2004; Carreras and Màrques 2004). Although robust analysis of shallow syntax including part-of-speech (POS) tags (Cutting et al. 1992; Brill 1994; Ratnaparkhi 1996; Brants 2000), dependency structures (Kurohashi and Nagao 1994; Eisner 1996; Collins 1996; Kudo and Matsumoto 2002; Yamada and Matsumoto 2003), and phrase structures (Jelinek et al. 1994; Magerman 1995; Collins 1997; Charniak 1997; Bod 2001; Collins 2003) has been successfully employed in some systems that need to handle natural language texts (*e.g.*, information retrieval (Baeza-Yates and Ribeiro-Neto 1999) and text categorization (Sebastiani 2002)), intelligent NLP applications such as information extraction, question answering, and machine translation have been reported to require more information about what sentences involve (Copestake et al. 1995; Palmer et al. 1998; Harabagiu et al. 2001; Surdeanu et al. 2003). The research area that aims at acquiring deep syntactic and semantic structures of sentences has thus emerged from behind the success of shallow syntactic analysis (Gildea and Palmer 2002; Bouillon et al. 2003; Carreras and Màrques 2004; Schulte im Walde and Brew 2002).

Among several attempts to provide in-depth syntax and semantic analysis, '*lexicalization*' approaches to formalization of grammars have been extensively pursued in both syntactic (Kaplan and Bresnan 1982; Gazdar 1988; Steedman 1986; Schabes et al. 1988; Pollard and Sag 1994) and semantic theories (Pinker 1989; Jackendoff 1990; Dowty 1991; Levin 1993; Pustejovsky 1995). In the established lexicalized grammar formalisms, syntactic constraints such as dependency and constituency are abstracted away from grammar rules; those constraints are radically relocated into lexical entries, and take the form of *a subcategorization frame* (SCF), a set of selectional constraints on the types and the number of arguments of a predicate. Grammar rules therefore include only a small number of construction-independent general rules, which interact with a richer lexicon to capture syntactic generalization. Syntactic arguments of a predicate (*e.g.*, verb) have thus a close tie with semantic arguments of the predicates within their lexical entries. This integrated analysis of syntactic and semantic structures is expected to meet the demand of the intelligent NLP systems.

Although the lexicalized grammar formalisms have potential to realize sophisticated NLP applications that require deeper linguistic analysis, such grammars have been rarely adopted in practical systems, due to the difficulty in developing indispensable components including grammar resources, parsing technologies, and lexical resources[1] suitable for the target domain. Because lexicalized grammars are designed to handle both in-depth syntactic and semantic phenomena, the design of grammar theories tends to be complicated, which also prevents us from achieving efficient processing environments. Although several theoretical and statistical parsing technologies have been studied in individual formalisms, more efforts should be necessary for helping grammar engineering and achieving enough efficiency for practical applications. On the other hand, lexicalized grammars are inherently dependent on intricate lexicons, and developing the lexical resources sets another bottleneck. Manual-development of comprehensive subcategorization lexicons has been proved to be costly and thus impractical. This is because predicates change their behavior between sublanguages, domains and over time (Sekine 1998; Roland 2001). Thus we need to establish both i) *static* components such as grammar resources and parsing technologies and ii) *dynamic* lexical resources.

This dissertation proposes two methodologies for solving the above problems that set bottlenecks in applying lexicalized grammars to practical applications. The first methodology accelerates collaboration among lexicalized grammars in order to build static grammar resources and parsing technologies. The second methodology tackles the problem to develop intricate lexicon resources that are dynamic in nature.

Approach to Collaboration among Lexicalized Grammars

The first part of this dissertation describes a novel approach to collaboration among the lexicalized formalisms, towards constructing *static* components such as grammar resources and parsing technologies that are generic within the lexicalized framework.

To date, individual lexicalized formalisms such as Lexicalized Tree Adjoining Grammar (LTAG) (Schabes et al. 1988), Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag 1994), and Combinatory Categorial Grammar (CCG) (Steedman 2000) have implemented gram-

[1]In this dissertation, we refer to lexical resources as associations between words and their lexical entry templates, while we refer to grammar resources as grammar rules and lexical entry templates.

mars and their processing environments in each closed community. Some discussion of the correspondences between the two formalisms has accompanied their development; i.e., their linguistic relationships and differences have been investigated (Abeillé 1993; Kasper 1998), as has conversion between two grammars in the two formalisms (Kasper et al. 1995; Tateisi et al. 1998; Becker and Lopez 2000). These ongoing efforts towards *collaboration* have contributed to the development of the two formalisms, particularly to the development of individual linguistic theories.

As have the linguistic theories elaborated, a wide range of parsers have been developed for those grammars. Parsers that have been proposed independently of one another often share the same parsing techniques that are claimed to be independent of individual grammar formalisms. Examples of such generic techniques are dynamic programming (Kasami 1965; Younger 1967; Earley 1970; Vijay-Shanker and Joshi 1985; Haas 1987), left-to-right parsing (Tomita 1986; Briscoe and Carroll 1993; Schabes 1994; Nederhof 1998), reductions to Boolean matrix multiplication (Valiant 1975; Satta 1994; Rajasekaran and Yooseph 1998; Lee 2002), and two-phase (or guided) parsing (Maxwell III and Kaplan 1993; Torisawa and Tsujii 1995; Yoshida et al. 1999; Barthélemy et al. 2001; Kay 2000) including CFG filtering (Harbusch 1990; Poller 1994; Torisawa and Tsujii 1996; Poller and Becker 1998; Torisawa et al. 2000; Kiefer and Krieger 2000). However, as mentioned by Carroll (Carroll 1994) and other researchers, while these techniques are generic in the sense that they can be used for efficient implementation of parsers for any grammar formalism, their impact often varies from one formalism to another (Schabes and Waters 1995; Yoshida et al. 1999). It seems that generic techniques actually interact with the characteristics of individual grammar formalisms.

The proposed work is thus intended to provide a basis for accelerating collaboration between the communities. It is built around a method of LTAG-to-HPSG grammar conversion, which is expected to lower the technical barrier between the grammar formalisms. This differs from previous methods of conversion in that it guarantees *strong equivalence* between the original and the obtained grammars;[2] that is, the results of parsing (derivation trees) by an LTAG grammar can be derived from those of the obtained HPSG-style grammar and vice versa. Having strongly equivalent grammars based on two formalisms is valuable for both communities in the following way:

Sharing of grammar resources HPSG-based applications can make use of LTAG grammar resources such as large-scale English (Doran et al. 2000) and French (Abeillé and Candito 2000) grammars that have been extensively developed. Our method of conversion can there-

[2]Chomsky (Chomsky 1963) first introduced the notion of strong equivalence between grammars, where both grammars generate the same set of structural descriptions (*e.g.*, parse trees). Kornai and Pullum (Kornai and Pullum 1990) and Miller (Miller 1999) used the notion of isomorphism between sets of structural descriptions to provide the notion of strong equivalence across grammar formalisms, which we have adopted in this research.

fore reduce the considerable workload involved in developing large-scale resources from scratch. In this dissertation, we report on the conversion of a large-scale LTAG grammar.

Linguistic correspondence We are also able to explore the correspondences between linguistic accounts given by the two formalisms. In the lexicalized framework, lexicon resources have similar constraints such as subcategorization frames, which are considered to be independent of individual formalisms. However, the grammar theories provide analogous linguistic accounts on some of the other linguistic phenomena, as in the treatment of long-distance dependencies in CCG and HPSG. Since the HPSG-style grammar obtained by our method has both the computational architecture that underlies HPSG and the linguistic specifications that were given in the original LTAG, the difference between the LTAG and HPSG formalisms will be made apparent by comparing the obtained grammars with hand-crafted HPSG grammars (Kay et al. 1994; Flickinger 2002). Such comparison will facilitate the development of grammar theories.

Comparison of parsing technologies Grammar conversion can be used as a means of abstracting away the surface differences between grammar formalisms, which are obstacles to carry out a meaningful comparison among generic parsing techniques implemented for different grammar formalisms. That is, by measuring the performance of parsers based on the original grammar and the ones based on the obtained grammar, one can gain a deeper insight into the generic parsing techniques and share techniques developed for parsers for different formalisms. Strongly equivalent grammars are also very helpful for incorporating techniques that have been found to be efficient from the parsers based on one formalism to the parsers based on another, because the grammar conversion defines a clear correspondence between those grammars.

We theoretically validate our approach by providing a formal proof of strong equivalence for our grammar conversion from LTAG to HPSG-style grammar, and empirically demonstrate our methodology by utilizing the grammar conversion for sharing of grammar resources and parsing comparison. We used the conversion algorithm which we implemented to successfully convert the XTAG English grammar (XTAG Research Group 2001), which is a large-scale LTAG grammar, into an HPSG-style grammar. In this dissertation, we investigated the types of linguistic phenomena covered by the XTAG English grammar, and the correspondence to their analysis in the HPSG formalism.

We focus on two generic parsing techniques in this dissertation, namely dynamic programming (Sarkar 2000; Haas 1987) and CFG filtering (Harbusch 1990; Poller 1994; Torisawa and

Tsujii 1996; Poller and Becker 1998; Torisawa et al. 2000; Kiefer and Krieger 2000). We first see how these techniques have been employed in parsers for the two particular grammar formalisms, LTAG and HPSG. Since dynamic programming forms the basis of most contemporary parsing techniques, a comparison of parsers using it allows us to roughly grasp the difference between the performance of LTAG and HPSG parsers. Since the impact of CFG filtering for LTAG is quite different from that for HPSG, CFG filtering can be a good material for demonstrating our methodology that improves generic parsing techniques through parsing comparison. Next, we show that grammar conversion yielding an HPSG-style grammar from a given LTAG grammar reveals the true nature of these generic parsing techniques. It follows from the experimental results that we suggest parsing techniques for LTAG that can be more efficient than those implemented for the original LTAG grammar, even though they use the same generic techniques.

Approach to Acquiring Lexical Resources from Corpora

The second part of this dissertation concerns novel approaches to acquire reliable '*dynamic*' lexical resources for lexicalized grammars from raw and annotated corpora in the target domain. Although our collaborative approach within the lexicalized grammar formalisms facilitates the development of static grammar resources, lexicon resources cannot be comprehensive enough to handle real-world sentences. This is because predicates change their behavior between sublanguages, domains and over time (Sekine 1998; Roland 2001), and such changes are difficult to predict in advance. Thus we need to establish a method for dynamically acquiring appropriate lexical knowledge from corpora.

There are two types of lexical resource available for lexicalized grammars. One is a lexicon that includes only associations between words and SCF types, in other words, *SCF co-occurrence* for words. A typical example of such lexicon is hand-coded lexicons, which are part of hand-crafted lexicalized grammars (Doran et al. 2000; Abeillé and Candito 2000; Flickinger 2002). The other type of lexical resource is a lexicon that includes not only associations between words and SCF types but also co-occurrence frequency counts between words and SCF types in a particular corpus, in other words, *SCF distributions* for words. A typical example of such lexicon is one automatically acquired from annotated corpora (Xia 1999; Miyao et al. 2004). These resources are precise enough to be employed for practical use, since they are built by human lexicographers or acquired from annotated corpora by using elaborated heuristic rules. Their recall is, however, reported not satisfactory for practical purpose (Roland 2001; Briscoe 2001), that is, they lack necessary words (problem on unknown words) or lack necessary subcategorization frames (problem on unknown associations between known subcategorization frames and known words) (Briscoe 2001).[3] In this dissertation, we present two methods of augmenting the above two types of lexical resources, respectively.

We first focus on a task that enhances the lexicons that include only associations between words and SCF types, and propose a method of augmenting such lexicons by SCFs acquired from raw corpora. A variety of methods have been proposed for automatic acquisition of general-purpose SCFs from corpora (Brent 1993; Ushioda et al. 1993; Manning 1993; Ersan and Charniak 1996; Briscoe and Carroll 1997; Carroll and Rooth 1998; Gahl 1998; Lapata 1999; Kuhn et al. 1998; Sarkar 2000) (surveyed in (Korhonen 2002)). One interesting possibility is to use these techniques to improve the coverage of existing large-scale lexical resources. However, there has been little work on evaluating the impact of acquired SCFs with the exception of (Carroll and Fang 2004). The problem when we integrate acquired SCFs into the target lexicon is the lower quality of the acquired SCFs, since they are acquired in an unsupervised manner, rather than being manually coded. If we attempt to compensate for the lack of recall by being less strict in filtering out less likely SCFs, then we will end up with a larger number of *noisy* lexical entries, which will cause erroneous parsing results. We thus need a method of selecting the most reliable set of SCFs from the system output as demonstrated in (Korhonen 2002).

In this task, we make use of SCF co-occurrences for words in the target lexicon to guide filtering of noisy acquired SCF lexicon. In the linguistic literature, SCF types taken by a single word is known to correlate with each other, and their alternation relations called *diathesis alternation* have been intensively studied (surveyed in (Levin 1993; McCarthy 2001)). In order to take advantages of such alternation relations that are implicitly included in the target lexicon, we first obtain *SCF confidence vectors* for words whose elements express how strong the evidence is that the word has each SCF type. In order to capture SCF co-occurrence in the target lexicon, we next perform clustering of SCF confidence vectors of words in the acquired SCF lexicon and the target lexicon. Since each centroid value of the obtained clusters indicates whether the words in that cluster have the SCF type, we eliminate SCFs acquired in error and predict possible SCFs according to the centroids. We applied our clustering method to SCFs acquired from mobile phone corpus (Carroll and Fang 2004), using the lexicons of the XTAG English grammar (XTAG Research Group 2001) and the LINGO English Resource Grammar (ERG) (Copestake 2002), respectively. We then compared the SCFs selected by our filtering method with SCFs obtained by naive frequency filtering in order to investigate the effect of clustering.

[3]There is another problem for existing lexical resources, which is called *unknown category* problem, that is, a word is expected to have a new SCF type which is not included in lexicons.

We second focus on the other type of lexical resource that includes co-occurrence frequency counts between words and SCF types in a particular corpus, and propose a method of constructing a probabilistic lexicon with accurate estimates of co-occurrence probabilities between words and SCF types, by smoothing the co-occurrence probabilities estimated from the raw frequency counts.[4] In such probabilistic lexicons, accuracy of estimates of the co-occurrence probabilities is quite important, because those probabilities determine the set of lexical entries that are employed for parsing of a given sentence. When we acquire SCFs from a small amount of annotate corpora, the resulting probability distributions are completely sparse. This implies that a parser may not be able to employ necessary SCFs in parsing. On the other hand, when we acquire a probability from a large amount of raw corpora, the resulting probability distributions are comprehensive but less accurate. Such noisy distributions are quite problematic because they can deteriorate not only parsing accuracy but also parsing efficiency (Sarkar et al. 2000). We thus need a method of acquiring accurate and comprehensive SCF distributions not only to have better parsing accuracy but also to determine an appropriate set of lexical entries that are employed in parsing.

In this task, we acquire accurate estimates of co-occurrence probabilities between words and SCFs from a small amount of annotated corpora by using the Probabilistic Latent Semantic Analysis (PLSA), which is a variant of latent class models, to perform smoothing of the observed accurate but sparse co-occurrence probabilities. The PLSA captures co-occurrence events of observed variables (words and SCF types in this case) by assuming unobserved latent variables or classes. We applied our smoothing method to SCFs of an HPSG grammar that is acquired from the Penn Treebank (Marcus et al. 1993), and compared the test-set perplexity of the co-occurrence probabilities estimated by linear interpolation using our PLSA model and raw observed frequency, with the test-set perplexity of the co-occurrence probabilities estimated by a more simple model only using raw observed frequency.

Thesis Structure The rest of this dissertation is structured as follows. Chapters 1 through 4 concern an approach to collaboration among grammar formalisms within the lexicalized grammar framework, while Chapters 5 through 7 concern an approach to acquiring lexical resources from corpora.

We first propose a grammar conversion from LTAG to HPSG-style grammar, and establish a methodology for the collaboration between the LTAG and HPSG formalisms using the nature of strong equivalence between an LTAG grammar and the HPSG-style grammar converted from the

[4]We assume that these lexicons assign probabilities to all possible associations between word and SCF types. The lexicon thus included lexical entries whose probabilities are larger than 0. In this context, smoothing co-occurrence probabilities between words and SCF types can be interpreted as an addition of plausible SCFs to the lexicons.

LTAG by the method we propose. Chapter 1 introduces the lexicalized grammar paradigm. We focus on two instance formalisms, LTAG and HPSG, and describe their processing architectures and existing resources including grammar resources and parsing techniques. Chapter 2 proposes a grammar conversion from LTAG to HPSG-style grammar. A formal proof of the strong equivalence between LTAG and HPSG-style grammar is provided in Section 2.3. Chapter 3 demonstrates our methodology through sharing grammar resources and comparison between generic parsing technologies for the two formalisms. Chapter 4 mentions work related to collaboration among the grammar formalisms.

We second describe a corpus-based extension of lexicalized grammar resources, making use of reliable lexical resources that are manually-tailored or acquired from annotated corpora. Section 5 explains linguistic behavior of subcategorization frames (SCFs), the target of our lexical acquisition, and then reviews methods for automatically extracting SCFs from corpus data. Chapter 6 proposes a method of filtering out less plausible SCFs from the SCFs acquired from raw corpora, exploiting co-occurrence tendency among SCF types in the target lexicon that includes only associations between words and SCF types. Chapter 7 proposes a method of obtaining accurate estimates for co-occurrence probabilities between words and SCF types, for the target lexicon that includes associations between words and SCF types along with their co-occurrence frequency counts.

Part I

Approach to Collaboration among Lexicalized Grammars

Chapter 1

Background to Lexicalized Grammar Formalisms

In this chapter we describe the lexicalized grammar paradigm (Schabes et al. 1988) and its two instance formalisms (Schabes et al. 1988; Pollard and Sag 1994) we concern.

Most of current linguistic theories give lexical accounts of several linguistic phenomena that used to be considered purely syntactic. The lexicon thus includes more information while grammar rules are abstracted simply to express general grammatical construction. Following the description given in (Schabes et al. 1988), we say that a grammar formalism is 'lexicalized' when it comprises:

Lexical entries: A finite set of (elementary) structures that are associated with words, which are usually heads of these structures. These structures involve word-specific lexical/syntactic constraints.

Grammar rules: A finite set of operations that compose elementary structures and generated structures. These grammar rules represent general grammatical constructions.

The elementary structures define the *domain of locality* over which constraints are specified, and these are local with respect to the lexical heads. The core constraint in the domain of locality for the lexical head is *a subcategorization frame* which roughly represents a list of *arguments* for the predicates (lexical heads). Arguments are words or phrases to complement the meaning of the lexical head. We will mention detailed discussions for arguments and subcategorization frames in Section 5.

In what follows, we describe the LTAG and the HPSG formalisms. An objective of our grammar conversion is to provide a formal link between the two formalisms, and then to bridge these

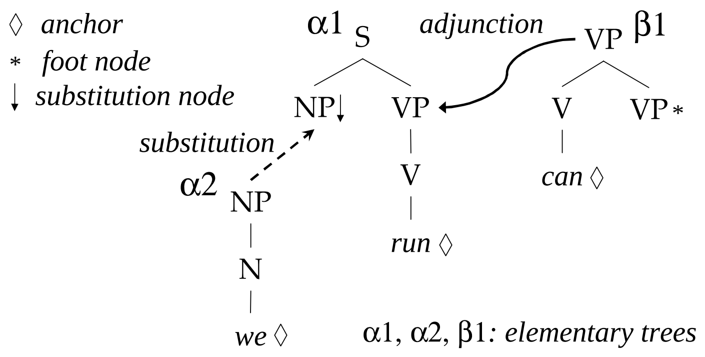


Figure 1.1: Lexicalized Tree Adjoining Grammar: basic structures (elementary trees) and composition operations (substitution and adjunction)

formalisms via strongly equivalent grammars obtained by the conversion. We therefore do not explain specific implementations of grammars in these formalisms, but describe the formal property of grammars in these formalisms. Because the formal property of HPSG is not clearly defined in the literature, we define *HPSG-style grammar*, the processing architecture that HPSG grammar defines over the typed feature structure (Carpenter 1992). Finally, we introduce grammar resources and parsing techniques developed for these formalisms.

1.1 Lexicalized Tree Adjoining Grammar

Lexicalized Tree Adjoining Grammar (LTAG) (Schabes et al. 1988) is a lexicalization approach to Tree Adjoining Grammar (TAG) (Joshi et al. 1975), and defined by a set of *elementary trees* that are composed by two operations called *substitution* and *adjunction*, as shown on Figure 1.1. An elementary tree has at least one leaf node that is labeled with a terminal symbol (i.e., word) called *an anchor* (marked with $\diamond$). Elementary trees are classified as either *initial trees* ($\alpha1$ and $\alpha2$) or *auxiliary trees* ($\beta1$). The label of one leaf node of an auxiliary tree is identical to that of its root node, and this is specially marked (here, with $*$) as *a foot node*. In an elementary tree, leaf nodes other than anchors and the foot node are called *substitution nodes* (marked with $\downarrow$).

In substitution, a leaf node (substitution node) is replaced by an initial tree, while in adjunction, an auxiliary tree with the root node and a foot node labeled x is grafted onto a node with the same symbol x. The results of analysis are described not only by *derived trees* (i.e., parse trees) but also by *derivation trees* (Figure 1.2). The derivation trees represent the history of combinations of trees, and are the deeper-level structural descriptions of LTAG. The left-hand side of Figure 1.2

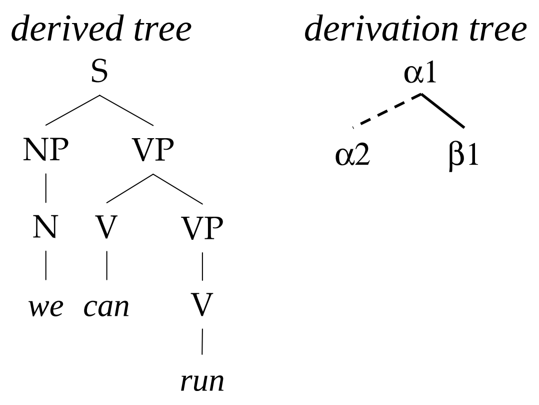


Figure 1.2: Derived tree and derivation tree

shows a derivation tree for the tree combination in Figure 1.1. In a derivation tree, each node expresses each elementary tree, and an elementary tree expressed by an internal node is substituted or adjoined by elementary trees expressed by its child nodes. Each branching thus expresses an application of substitution and adjunction. In the right-hand side of Figure 1.2, an elementary tree $\alpha 1$ is substituted by an elementary tree $\alpha 2$, and is adjoined by $\beta 2$.[1]

FB-LTAG (Vijay-Shanker 1987; Vijay-Shanker and Joshi 1988) is an extension of the LTAG formalism in which each node in the elementary trees has a feature structure, which contains a set of grammatical constraints on the node. The constraints are to be satisfied through unification during adjunction and substitution.

It is worth mentioning that the two key properties of LTAG elementary trees allow all dependencies involving a particular word to be local within the lexical entries of the word; 1) *extended domain of locality* (compared to CFG), and 2) factoring recursive structures from the domain of locality. Kroch and Joshi (Kroch and Joshi 1986; Kroch 1987; Kroch 1989) built linguistic foundation of the TAG formalisms by exploiting of these properties. The extended domain of locality allows us to express properties related to a word (such as subcategorization, agreement, certain types of word order variation) within the elementary tree anchored by the word. Figure 1.3 shows example elementary trees of English LTAG grammar. Elementary trees $\alpha 3$, and $\alpha 4$ in the figure represent subcategorization frames for the lexical anchors, "*love*" and "*give*." The argument category and their linear precedence constraints are explicitly expressed by the substitution nodes of an

[1]Strictly speaking, a derivation tree includes information on which nodes of an elementary tree take substitution or adjunction by augmenting each branching with the node address in which those operations take place.

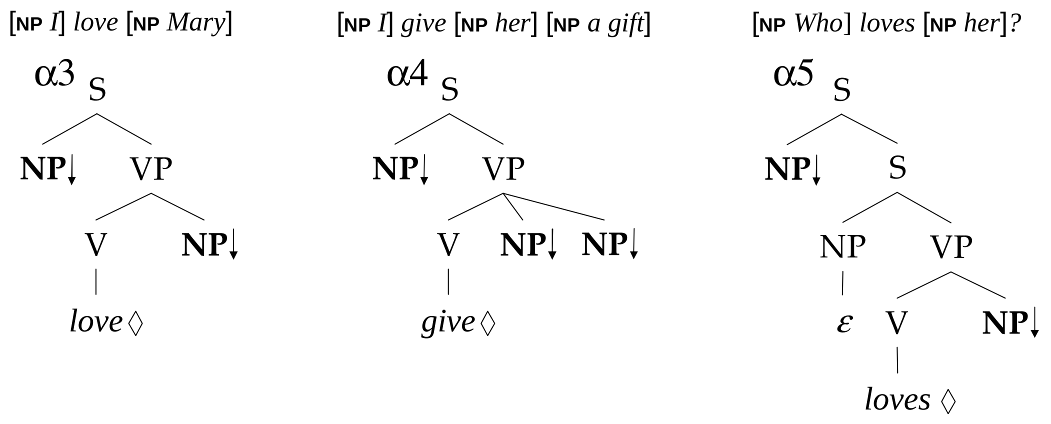


Figure 1.3: Elementary trees that represent subcategorization frames

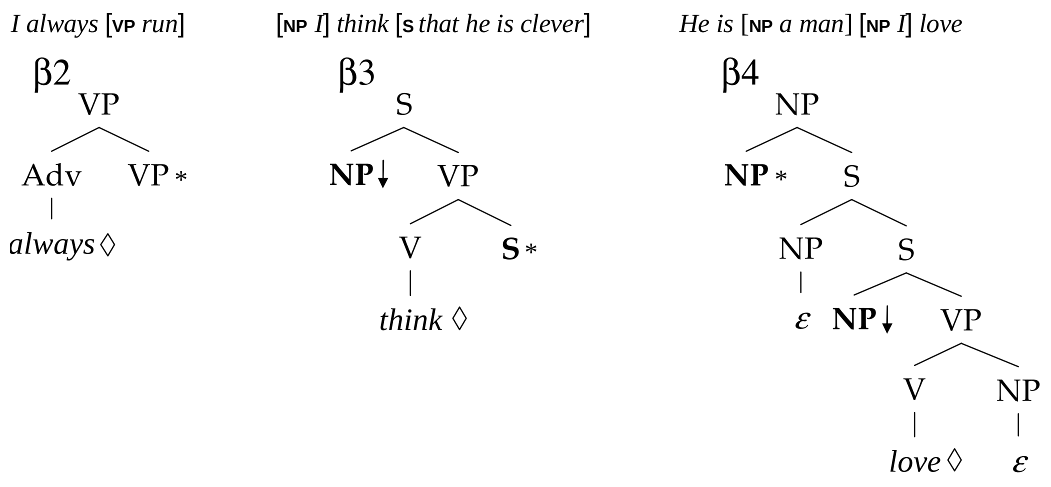


Figure 1.4: Elementary trees that represent recursive structures

elementary tree and their positions within the tree structure.[2] The elementary tree $\alpha5$ expresses a subject wh-extraction tree for $\alpha1$. While dependency and constituency among predicates and their arguments are roughly captured by putting them together within one elementary tree, recursive structures of natural language such as adjunction and modification are expressed by one auxiliary tree. Figure 1.4 shows example auxiliary trees. Auxiliary trees $\beta2$ and $\beta3$ respectively show modi-

[2]In the implemented LTAG grammars such as the XTAG English grammar (XTAG Research Group 2001), arguments in predicates' elementary trees are assigned subscripts according to their thematic roles.

fication of a verb phrase by an adverb "*always*" and an embedded clause of a verb "*think*." Because such modifiers and embedded clauses can repeatedly appear in a sentence, they are factored out as single recursive structures, i.e., auxiliary trees. Another example is object-extracted relative clause $\beta 4$. The relative clause introduces to a sentence *non-local* dependency between an antecedent and a predicate in the clause (the foot node and an anchor verb in the right-hand side of Figure 1.4. By extending domain of locality to include an antecedent in an elementary tree of an anchor verb, this non-local dependency is successfully expressed within the single elementary tree.

In short, the LTAG formalisms classify grammatical constructions into head-argument and head-adjunct relations according to their recursiveness. Syntactic properties related to a lexical anchor are explicitly described in the extended domain of locality given within each elementary tree.

1.2 Head-Driven Phrase Structure Grammar

HPSG is a linguistic theory based on the lexicalized grammar formalism, and is characterized by a modular specification of linguistic generalizations. It consists of *lexical entries* and *Immediate Dominance (ID) grammar rules* which are further broken down into *ID schemata* and *principles*.[3] All of them are described with *typed feature structures* (Carpenter 1992).

We should briefly introduce a formal property of the typed feature structures as a data structure. The typed feature structure is a rooted directed acyclic graph structure whose nodes and edges have an associated label. The label associated with a node is called *type* while a label associated with an edge is called *feature*. In the following typed feature structures, we express features with capitalized letters while we describe types with uncapitalized and italicized letters. Because the value of features can be either a type or a typed feature structure, the typed feature structures can represent a recursive structure like lists.[4] Figure 1.5 exemplifies a typed feature structure and its attribute-value matrix description. In the following, we describe typed feature structures by the attribute-value matrix description.

In the HPSG formalism, syntactic properties related to a lexical head are expressed in a more abstract way than in the LTAG formalisms. Figure 1.5 provides the definition of an HPSG *sign*,

[3]Strictly speaking, an HPSG grammar consists only of lexical entries and principles. One of the principles called *Immediate Dominance Principle* mention that immediate dominance constituency should be licensed by one of the rule schemata.

[4]The relations between types are represented by type hierarchy, and the unification is defined over the type hierarchy. We omit the definition of these notions in this dissertation because feature structures in the FB-LTAG formalism are not typed and we do not explicitly employ typing in the following feature structures except in the definition of the HPSG feature structure.

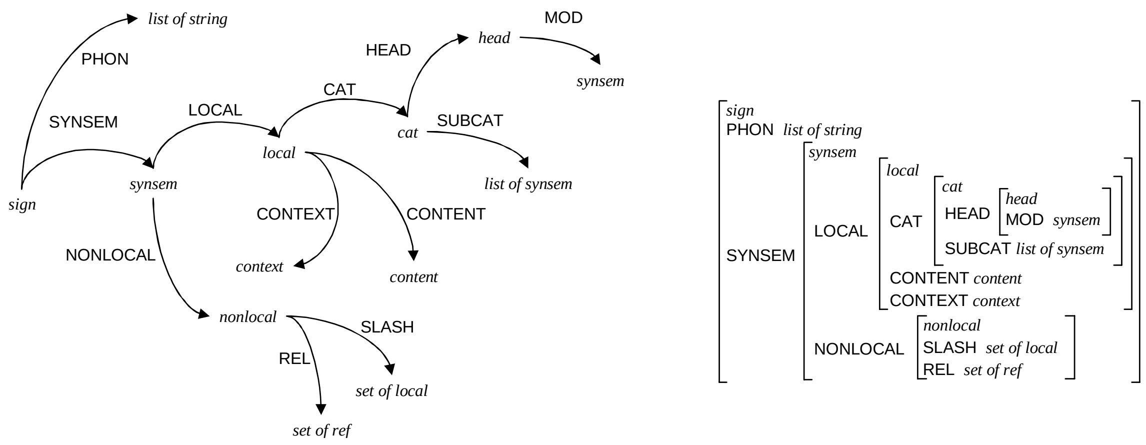


Figure 1.5: HPSG sign (left) and its AVM description (right)

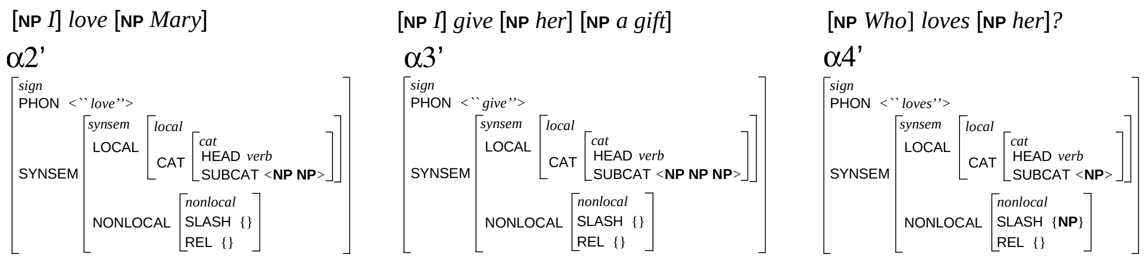


Figure 1.6: HPSG signs that represent subcategorization frames

which represents syntactic and semantic behavior of a word or a phrase. HEAD feature expresses the characteristics of the head word of the sign, such as syntactic category. SUBCAT feature represents a subcategorization frame, a list of selectional constraints on the arguments of the head word, while MOD feature represents a constraint on the modifiee of the head word (modifier). Long-distance dependencies are captured by the use of NONLOCAL feature structures that involve SLASH and REL features. Figure 1.6 exemplifies HPSG lexical entries for syntactic constructions provided in Figure 1.3. While in LTAG subcategorization frames are represented by leaf nodes of elementary trees, in HPSG they are represented by SUBCAT feature when they are local and by SLASH feature when they are nonlocal, both of which are included in a lexical entry for the same word. Head-adjunct relations, which are represented by relations between a foot node and anchors in LTAG, are represented by the MOD feature, as shown in the left-hand side of Figure 1.7.

Unlike the LTAG formalism, in HPSG, the constraints on possible syntactic structures taken

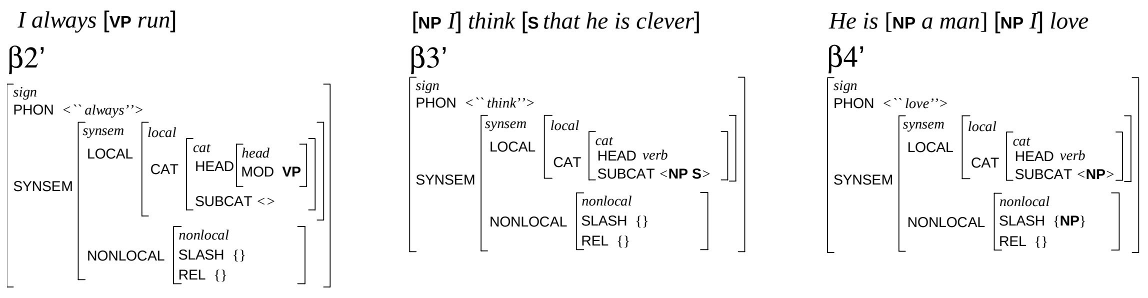


Figure 1.7: HPSG signs that represent recursive structures

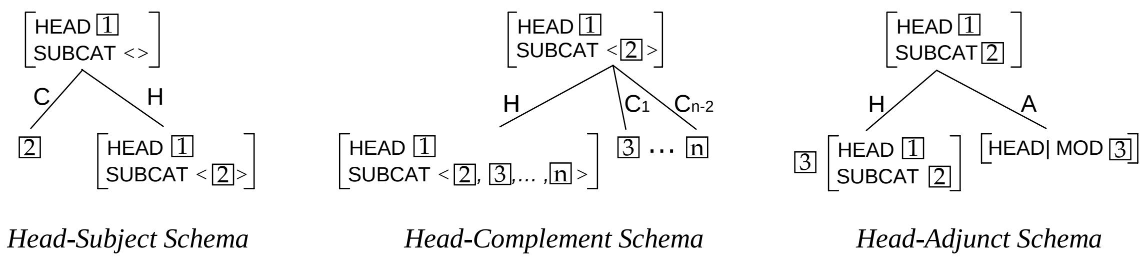


Figure 1.8: ID schemata that represent head-subject, head-complement, and head-adjunct relations

by a head word are not explicitly included in lexical entries but are modularized into a set of principles. There are three language-independent general principles that explicitly specify these constraints. *Immediate Dominance (ID) Principle* breaks down the grammatical constructions into the configurations of *immediate constituency*; local trees of depth one must be constructed by one of *Immediate Dominance (ID) Schemata*. The *ID schemata* provide an abstract definition of grammatical relations such as head-complement, head-subject, head-adjunct, head-marker, head-filler and the like. According to this notion of immediate constituency, other principles define the flow of information in a global structure. *The Head-Feature Principle* describes the identity of the HEAD Features between a phrase and its syntactic head. *The Subcategorization Principle* constrains the SUBCAT feature of the mother of a local tree, which expresses the fact that a head word subcategorizes their arguments. Other principles describe construction-specific information flow. Head-adjunct constraints, which LTAG formalized as adjunction operations, are provided not by principle but by the Head-Adjunct Schema; the schema represents modification of head daughter by non-head daughter, as show in the right-hand side of Figure 1.8.

Figure 1.8 shows examples of ID grammar rules for English provided in Pollard and

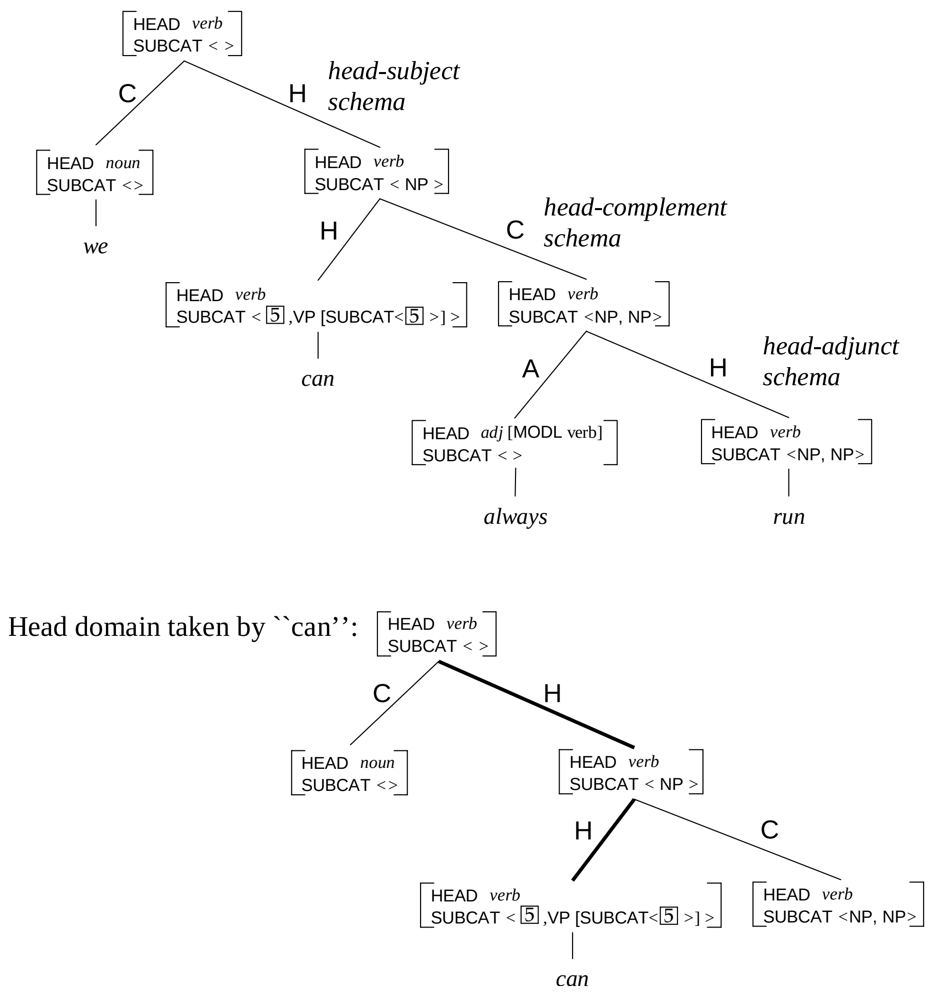


Figure 1.9: HPSG parse tree for a sentence "*we can always run*" and the head domain taken by "*can*"

Sag (Pollard and Sag 1994). The information flow between a mother and the daughters of a local tree is represented by the boxed numericals '$\boxed{n}$' called *tags* that express *structure-sharing*; the boxed numericals with the same number express the sharing of common values between the two (sub-)feature structures. The Head Feature Principle is represented by structure-sharing '$\boxed{1}$' between the HEAD feature structure, while the Subcategorization Principle is represented by structure-sharing '$\boxed{n}$' in the SUBCAT features. The three schemata represent linear precedence configuration of daughters of a local tree according to values of the SUBCAT features. The left-hand side and center of Figure 1.8 represent head-subject and head-complement relations, which the head takes arguments according to its SUBCAT value. The right-hand side of Figure 1.8 shows head-adjunct relation.

Using these definitions of principles and ID schemata, we define for each word *head domain*, which is a notion analogous to extended domain of locality for a word in LTAG. The head domain of a word is defined to be a syntactic tree structure that can be derived using values of features which are related to word's subcategorization frames (*e.g*, the SUBCAT, SLASH features).[5] Figure 1.9[6] is an example of a head domain of an auxiliary "*can*" in a sentence "*we can run*." Due to immediate constituency defined by the ID principle, the head domain is defined through a path from a head word to its maximal projection, along with the siblings around the path.

1.3 HPSG-*style* Grammar: HPSG's Computational Architecture

As seen in Sections 1.1 and 1.2, while LTAG developed the linguistic theory on the mathematically well-defined computational architecture, HPSG expresses a grammar with the typed feature structures, more powerful framework which can describe any directed acyclic graph, and the set of language-independent principles further restrict the form of grammars. We thus capture the difference between the LTAG and the HPSG grammars in terms of 1) the difference between the computational architecture that underlies the two formalisms and 2) the difference between the ways of locating grammatical constraints in lexical entries, grammar rules, and principles (especially in HPSG). Because existing studies attempted capturing these two differences at once, they obscure the formal property of the relation between the original and the obtained grammars. In this dissertation, we rather focus on the difference between the computational architecture that underlies the LTAG and HPSG formalisms, and then reveal the formal properties of it.

[5]In the work on conversion from HPSG to LTAG (Kasper et al. 1995), Kasper et al. refer these features as *selector feature*.

[6]The value of category is presented for simplicity, and the irrelevant parts of the sign have been omitted.

We now define *an HPSG-style grammar*, the computational architecture of HPSG (Pollard and Sag 1994), which are defined by the three general HPSG principles over the typed feature structures (Carpenter 1992). It consists of *lexical entries* and *Immediate Dominance (ID) grammar rules*, each of which is described with typed feature structures. The greater generative power of the underlying representation framework of HPSG allows us to obtain a trivial encoding of LTAG in the typed feature structure, as described by Keller (Keller 1994, pp. 144–151).[7] However, such a conversion cannot meet our needs because the resulting grammar is far from the one defined in (Pollard and Sag 1994), in that the resulting grammar does not satisfy the restriction on the processing architecture imposed by the principles in HPSG, such as immediate constituency. Hence, we restrict the form of an HPSG-style grammar to one that follows the HPSG formalism in the following ways. A lexical entry for a particular word must express the characteristics of the word, such as its subcategorization frame and grammatical category. ID grammar rules must represent the constraints on the configuration of immediate constituency and not be a construction-specific rule defined by lexical characteristics. These restrictions enable us not only to define a formal link between computational architectures that underlies LTAG and HPSG, but also to clarify the relationships between linguistic accounts given using LTAG and HPSG by comparing the HPSG-style grammar converted from LTAG with HPSG.

We should note that the HPSG-style grammar we have sketched above satisfies the requirement on the processing architecture that is assumed in implemented HPSG parsers. This means that we can apply any HPSG parsers to the obtained HPSG-style grammar. We should note that this property of HPSG-style grammar is also indispensable for parsing comparison between LTAG and HPSG parsers, because in parsing comparison we run LTAG and HPSG parsers using an LTAG grammar and the HPSG-style grammar obtained by the grammar conversion, respectively.

Note that Pollard and Sag (Pollard and Sag 1994) provide detailed linguistic specifications for the form of feature structures and adopt (language-specific) principles, as shown in Section 1.2. In our definition, we assume that principles are implicitly encoded in lexical entries and when we convert an LTAG grammar to an HPSG-style grammar we do not attempt to modularize linguistic specifications in the LTAG into the corresponding HPSG principles. In this manner, our study also investigates the utility of the HPSG processing architecture, which is defined by the HPSG principles over the typed feature structure, as a means of expressing different linguistic theories and intuitions.

Figure 1.10 illustrates an example of bottom-up parsing with an HPSG-style grammar. In the

[7] In their study, they simply represent a tree structure by nested lists like S-expressions in the programming language LISP.

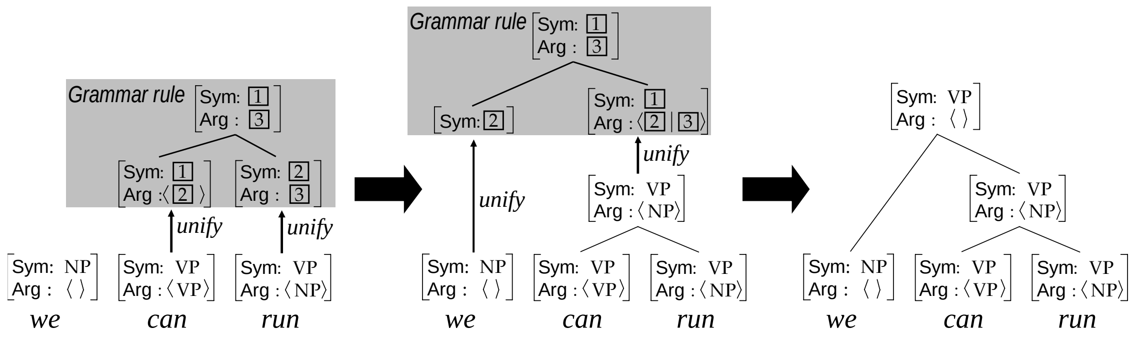


Figure 1.10: Example of bottom-up parsing with an HPSG-style grammar

HPSG framework, a parse tree is generated by incrementally applying ID grammar rules to lexical entries and constructing each of the branching structures one by one, while in LTAG it is done by composing elementary trees with the two operations. Thus, the key points in the conversion are 1) how to encode the tree structure of an elementary tree as an HPSG lexical entry, and 2) how to emulate substitution and adjunction by ID grammar rules. Note that there is no one-to-one correspondence between elementary trees and HPSG lexical entries. This is because the head domain defined by an HPSG lexical entry must take the form of the tree structure that can be decomposed into immediate constituency (Figure 1.9), while the extended domain of locality defined by an LTAG elementary tree can take arbitrary tree structure.

1.4 Grammar Resources and Parsing Techniques for LTAG and HPSG

1.4.1 Grammar Resources and Parsing Techniques for LTAG

The LTAG and its variant formalisms have been applied to various NLP/CL applications such as machine translation (Abeillé et al. 1990; Palmer et al. 1998), information retrieval (Chandrasekar and Srinivas 1997), generation (Joshi 1987; McCoy et al. 1992; Stone and Doran 1997), summarization (Baldwin et al. 1997), and psycholinguistic modeling (Joshi 1990; Kim et al. 1990; Kinyon 1999). In the following paragraphs, we introduce existing grammar resources and parsing technologies developed for the LTAG formalisms.

There are several grammars developed in the FB-LTAG formalism, including the XTAG English grammar, a large-scale English grammar (XTAG Research Group 2001) developed by the

XTAG Research group at the University of Pennsylvania. The XTAG Research Group has also developed Korean, Chinese, and Hindi grammars. Development of a large-scale French grammar (Abeillé and Candito 2000) has also started at the University of Pennsylvania, and is expanded at University of Paris 7. The XTAG group also provided a grammar development environment called the XTAG system, which is the most complete TAG workbench currently available (Doran et al. 2000). It includes a graphical interface (Paroubek et al. 1992), a parser (Schabes 1994), and a lexicon compiler.

Thanks to the mathematical foundation that underlies the LTAG framework, there are several theoretical studies on parsing algorithms for LTAG. Some parsing algorithms that are originally developed for CFG are re-interpreted in the LTAG context. Examples include a CKY-style bottom-up parser (Vijay-Shanker 1987), and Earley-style top-down parsers (Schabes and Joshi 1988; Schabes 1994; Nederhof 1999), and head-driven parsers (Lavelli and Satta 1991; van Noord 1994; Sarkar 2000), and parsers based on Boolean matrix multiplication (Satta 1994; Rajasekaran and Yooseph 1998). These parsers have at most the worst-case time complexity $O(n^6)$ for input length n with the exception of (Satta 1994; Rajasekaran and Yooseph 1998), which requires $O(M(n^2))$ where $M(k)$ is the time needed to multiply two $k \times k$ Boolean matrices. One of other parsing algorithms that are claimed to be empirically efficient is two-phased (or guided) parsing (Harbusch 1990; Poller 1994; Poller and Becker 1998; Yoshida et al. 1999; Barthélemy et al. 2001), which uses a CFG extracted from the original grammar to prune invalid parse trees before using the whole constraints of the original grammar.

1.4.2 Grammar Resources and Parsing Techniques for HPSG

The HPSG formalism has been applied to various NLP/CL applications such as dialogue translation (Kay et al. 1994), machine translation (Copestake et al. 1995), information extraction (Yakushiji et al. 2001; Tsujii 2001), appointment scheduling (Uszkoreit et al. 1994), and generation (Carroll et al. 1999). In the following paragraphs, we introduce existing grammar resources and parsing technologies developed for the HPSG formalism.

Stanford University has been developing the English Resource Grammar (ERG), an HPSG grammar for English, as part of the Linguistic Grammars Online (LinGO) project (Flickinger 2002). In practical context, German (BABEL) and Japanese (JACY) HPSG-based grammars are developed and used in the Verbmobil project (Kay et al. 1994). From the implemented English (ERG) and Japanese (JACY) HPSG grammars, these groups also attempted to extract the components that are common across these grammars and therefore may be useful in the development of new grammars, and then provided them as the Grammar Matrix (Bender et al. 2002), which is open-source starter-kit for the development of new HPSG grammars. The Grammar Matrix is demonstrated for developing Greek, Italian, Norwegian HPSG grammars. The University of Groningen has independently developed a wide-coverage HPSG-like grammar (Alpino) (Bouma et al. 2000). University of Tokyo has translated the XTAG English grammar into a large-scale HPSG-like grammar (XHPSG) (Tateisi et al. 1998), and in parallel developed a wide-coverage underspecified HPSG-like grammar for Japanese (SLUNG) (Mitsuishi et al. 1998), which is used in a high-accuracy Japanese dependency analyzer (Kanayama et al. 2000). They have established a methodology for developing large-scale grammars, the corpus-oriented grammar development (Miyao et al. 2004), and have applied the methodology to the acquisition of robust English (Miyao et al. 2004) and Japanese (Yoshida 2005) HPSG-like grammars. Stanford University has developed grammar development environment called LKB (Linguistic Knowledge Builder), while University of Tokyo uses grammar development environment called will (Imai et al. 1998) and its extension called willex (Yakushiji et al. 2003) and Moriv.[8]

There are a variety of works on efficient parsing with HPSG, which allow the use of HPSG-based processing in practical application contexts (Flickinger et al. 2000; Oepen et al. 2002). As in the LTAG formalism, some parsing algorithms originally designed for CFG are imported in to the HPSG framework. Examples include a CKY-style bottom-up parser (Haas 1987) and a left-corner parser (Tomuro and Lytinen 2001). However, due to complex encoding of linguistic features in the typed feature structures, cost of unification is the major obstacle to achieving an empirically efficient HPSG parser. In order to solve inefficiency of unification, several algorithms have been proposed (*e.g.*, two-phased parsing (Torisawa and Tsujii 1996; Torisawa et al. 2000; Kiefer and Krieger 2000) and a quick check (Malouf et al. 2003) method that immediately checks unifiability of feature structures).

[8]`http://www-tsujii.is.s.u-tokyo.ac.jp/moriv/`

Chapter 2

Grammar Conversion from LTAG to HPSG-style Grammar

This chapter describes in detail an algorithm for converting from LTAG to strongly equivalent HPSG-style grammar, and discusses the correspondence between HPSG-style grammar and HPSG. The formal proof of strong equivalence between an LTAG and the HPSG-style grammar converted from the LTAG by the following grammar conversion is provided in Session 2.3.

2.1 Algorithm

As noted in Section 1.3, the grammatical constraints expressed in LTAG elementary trees should be encoded in HPSG lexical entries, and substitution and adjunction should be emulated by ID grammar rules. Thus, we propose a conversion algorithm which consists of 1) the conversion of elementary trees into HPSG lexical entries and 2) the emulation of substitution and adjunction by pre-determined ID grammar rules.

In the following description, we start by defining *canonical elementary trees*, which have a one-to-one correspondence with HPSG lexical entries.[1]

Definition 2.1.1 (Canonical elementary tree) *Canonical elementary trees are elementary trees that satisfy the conditions below:*

[1]In this dissertation, we assume that elementary trees consist of binary branching structures. A unary branching can be regarded as a binary branching in which one daughter is the empty category, and n-ary ($n \geq 3$) branchings can similarly be converted into binary branchings. This conversion guarantees strong equivalence by virtue of being a one-to-one mapping.

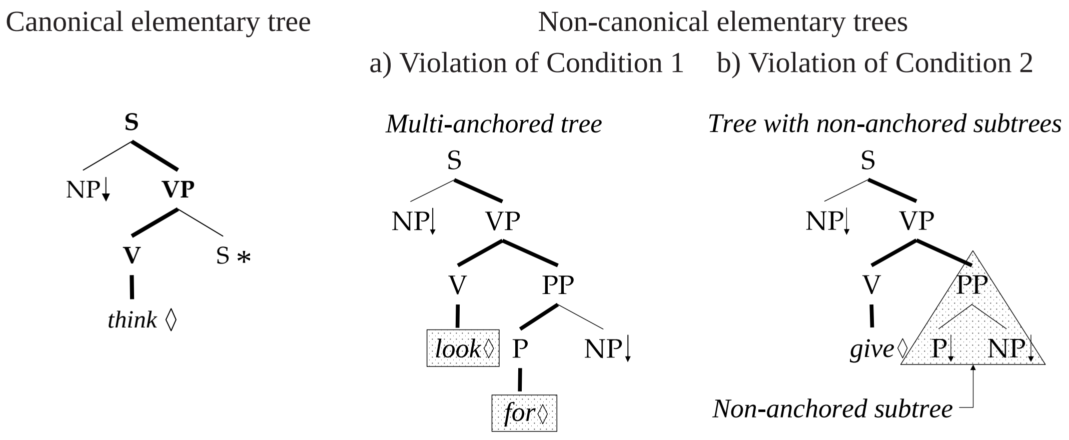

Canonical elementary tree
Non-canonical elementary trees
a) Violation of Condition 1
b) Violation of Condition 2
S
NP↓
VP
V
S ∗
think ◊
Multi-anchored tree
S
NP↓
VP
V
PP
look◊
P
NP↓
for◊
Tree with non-anchored subtrees
S
NP↓
VP
V
PP
give◊
P↓
NP↓
Non-anchored subtree

Figure 2.1: A canonical elementary tree and non-canonical elementary trees

Condition 1: A tree has only one anchor,
Condition 2: Every branching structure in a tree contains trunk nodes,

where *trunk nodes* (nodes with bold face in Figure 2.1) are nodes on *a trunk* which is a path from an anchor to the root node (the thick lines in Figure 2.1) other than the anchor (Kasper et al. 1995). Conditions 1 and 2 respectively guarantee that a canonical elementary tree has only one trunk and that each branching consists of a trunk node, a leaf node, and their mother which is also a trunk node, as seen in the example on the left-hand side of Figure 2.1. The center and the right-hand side of Figure 2.1 show non-canonical trees. We call a subtree of depth $n(\geq 1)$ that includes no anchor *a non-anchored subtree* (the right-hand side of Figure 2.1). Non-canonical elementary trees are converted to canonical trees before converting into HPSG lexical entries by the algorithm in the next section.

It should be noted that trunk nodes in canonical elementary trees are analogous to *heads* in the HPSG formalism in each immediate constituency where they select a category next to be combined. Thus a canonical elementary tree expresses the head domain defined by the lexical head. However, we avoid using the term head instead of the term trunk because the trunk nodes do not always have one-to-one correspondence to the notion of head in the HPSG formalism. We will discuss the notion of head in HPSG-style grammar in Section 2.2.

```
INPUT:   a canonical elementary tree $T$
OUTPUT:  a HPSG lexical entry $L$

procedure convert_tree_into_lexical_entry($T$)
begin
  $w$ := $n_{\text{depth}(T)}$
  $arg$ := []
  for $i$ := 1 to depth($T$)$-1$
    $n_{i-1}$ := trunk($T$, $i-1$)
    $l_i$ := leaf($T$, $i$)
    $d_i$ := direct($T$, $i$)
    $t_i$ := type($T$, $i$)
    $b_i$ := $\langle n_{i-1}, l_i, d_i, t_i \rangle$
    $arg$ := [$b_i$] $\oplus$ $arg$
  end for
  $L$ := $\langle arg, n_{\text{depth}(T)-1} \rangle$
  return $\langle w, L \rangle$
end
```

`depth:`	returns the integer of depth of the anchor.
`trunk:`	returns the non-terminal symbol of the trunk node.
`leaf:`	returns the non-terminal symbol of the leaf node at depth i
`direct:`	returns the side of the trunk node (left or right) for the leaf node at depth i
`type:`	returns $+$ when the leaf node of at depth i is a foot node or $-$ when a substitution node

Figure 2.2: An algorithm for converting a canonical elementary tree T to an HPSG lexical entry L

2.1.1 Conversion of Canonical Elementary Trees

As discussed in the previous section, when we assume the trunk nodes in a canonical elementary tree to be heads that select a category to be subcategorized next, a canonical elementary tree expresses the head domain of the lexical word. A canonical elementary tree can be therefore decomposed into a list of immediate constituency.

The procedure `convert_tree_into_lexical_entry` in Figure 2.2 presents an algorithm for converting a canonical elementary tree T into an HPSG lexical entry L. In the algorithm, arg is a stack of branchings b_i as described by a quadruplet $\langle n_{i-1}, l_i, d_i, t_i \rangle$ along the trunk. The parameter n_{i-1} represents the mother node of the trunk node n_i. The parameters l_i, d_i and t_i represent the

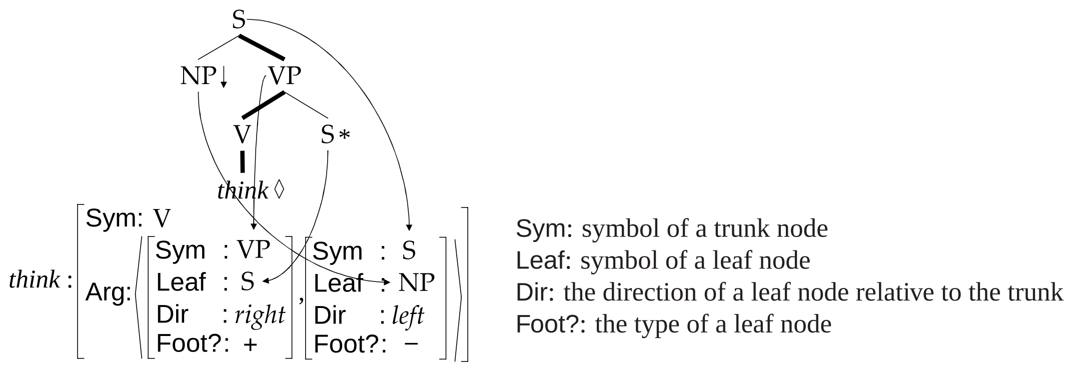

S
NP↓
VP
V
S∗
think ◊
think :
Sym: V
Arg:
Sym : VP
Leaf : S
Dir : right
Foot?: +
Sym : S
Leaf : NP
Dir : left
Foot?: −
Sym: symbol of a trunk node
Leaf: symbol of a leaf node
Dir: the direction of a leaf node relative to the trunk
Foot?: the type of a leaf node

Figure 2.3: Converting a canonical elementary tree for “think” to an HPSG lexical entry

leaf node at a depth i; respectively, they represent the nonterminal symbol, the direction (the side of the trunk node n_i on which the leaf node is), and the type (whether the node is a foot node or a substitution node). We call elements in this stack *arguments* of the word. Finally, the converted lexical entry L is the pair $\langle arg, n_{\mathtt{depth}(T)-1}\rangle$ described by the arguments arg and the mother of the anchor, namely, $n_{\mathtt{depth}(T)-1}$ where $\mathtt{depth}(T)$ is the depth of the tree T.

Figure 2.3 depicts an example of conversion of a canonical elementary tree for “*think*,” and shows the design of a feature structure that express an HPSG lexical entry. A canonical elementary tree is converted into an HPSG lexical entry by regarding leaf nodes as arguments of the anchor and storing them in a stack. The resulting feature structures include the Sym feature and the Arg feature that store the symbol $n_{depth(T)-1}$ and the arguments in arg, as a stack of feature structures with the four features Sym, Leaf, Dir and Foot?, which correspond to n_{i-1}, l_i, d_i and t_i, respectively. In parsing with the obtained HPSG-style grammar, the parser pops an element from the Arg feature to select a node that is unifiable with that element. It follows that the node with an empty stack as its Arg feature corresponds to the root node of the initial tree.

2.1.2 Tree Division: Division of Non-canonical Elementary Trees

Non-canonical elementary trees are initially divided into multiple subtrees, each of which has at most one anchor, by a procedure called *tree division*, as shown in Figures 2.4 and 2.5. Nodes that mark the separation of one tree into two are called *cut-off nodes*. A cut-off node is marked by *an identifier* to preserve the co-occurrence relation among the multiple anchors. The tree division converts multi-anchored trees, which only violate Condition 1, into canonical trees (Figure 2.4), while it converts trees with non-anchored subtrees into canonical trees and non-anchored subtrees

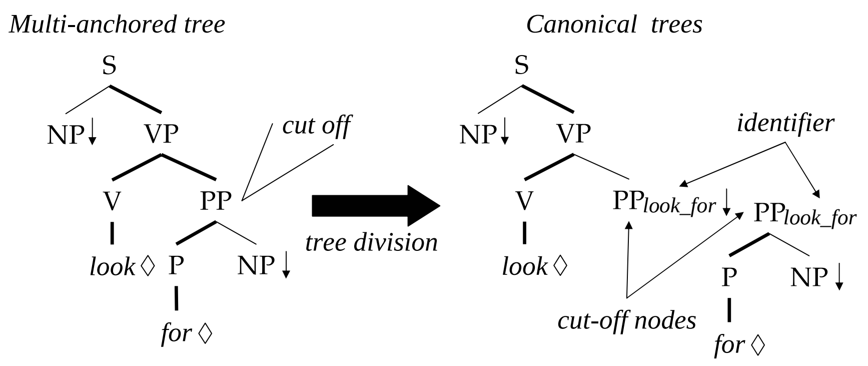


Figure 2.4: Dividing a multi-anchored elementary tree for "look for" into a set of subtrees, each of which has at most one anchor.

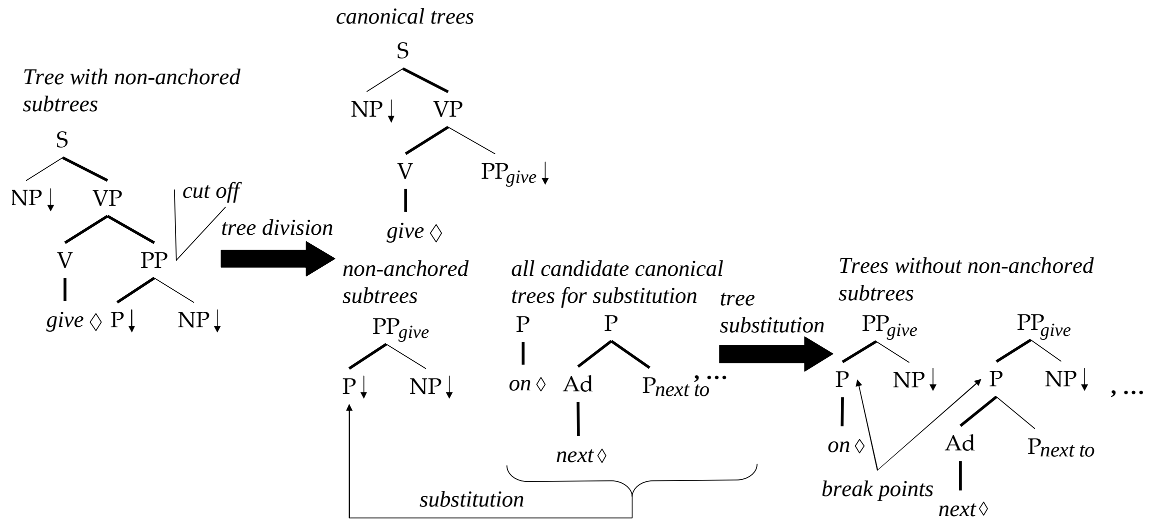


Figure 2.5: Converting a non-anchored subtree to a set of multi-anchored trees

(Figure 2.5).

The procedure `divide_tree_into_subtrees` in Figure 2.6 represents an algorithm for dividing a non-canonical elementary tree MT into a set of subtrees $\mathcal{ST}$. It starts by selecting one anchor A,[2] and the single-anchored tree CT of that anchor A, which consists of the trunk nodes and their sibling nodes, is then picked up (`(1)` in Figure 2.6). We traverse the path from the root node to the

[2]We describe how the function `select` in the algorithms in this section and the next section select an anchor/a leaf node respectively in Section 2.2.

```
INPUT:  a non-canonical elementary tree $MT$
OUTPUT: a set of trees with at most one anchor $\mathcal{ST}$

procedure divide_tree_into_subtrees($MT$)
begin
  if number($MT$) = 0
    return $\{MT\}$
  else
    $A$ := select($MT$)
    $\langle CT, \mathcal{T}\rangle$ := divtree($MT$, $A$)                $\cdots$ (1)
    foreach $T$ in $\mathcal{T}$
      $\mathcal{SST}$ := divide_tree_into_subtrees($T$)
      $\mathcal{ST}$ := $\mathcal{SST} \cup \mathcal{ST}$
    end foreach
    $\mathcal{ST}$ := $\mathcal{ST} \cup \{CT\}$
    return $\mathcal{ST}$
  end if
end

INPUT:  a non-canonical elementary tree $MT$
        one anchor of $MT$ $A$
OUTPUT: a pair of a canonical elementary tree $CT$
       and tree fragments $\mathcal{T}$

procedure divtree($MT$, $A$)
begin
  $\mathcal{T}$ := $\phi$
  for $i$ := 1 to depth($MT$, $A$)-1
    if nonleaf(arg(trunk($i$)))
      $\langle MT', T\rangle$ := cut($MT$, arg(trunk($i$)))     $\cdots$ (2)
      $Address$ := address(trunk($i$))
      mark($Address$, $MT'$, $T$)                       $\cdots$ (3)
      $\mathcal{T}$ = $\mathcal{T} \cup T$
      $MT$ := $MT'$
    end if
  end for
  $CT$ := $MT$
  return $\langle CT, \mathcal{T}\rangle$
end
```

`number:`	returns the number of anchors.
`select:`	returns one of anchors (default: the left-most one).
`depth:`	returns the integer of depth of the anchor.
`trunk:`	returns the trunk node at depth i
`arg:`	returns the sibling node of the trunk node.
`cut:`	cuts off the tree at the sibling node of the trunk node and returns a subtree whose root node is the sibling node.
`nonleaf:`	returns true if not a leaf node.
`address:`	returns address in the elementary tree.
`mark:`	marks an address for each cut-off node.

Figure 2.6: An algorithm for dividing a non-canonical elementary tree MT into a set of subtrees $\mathcal{ST}$, each of which has at most one anchor.

anchor A, cut off the sibling node[3] `arg(trunk(`i`))` if it is not a leaf node, and store *the address* of the elementary tree in the cut-off node as an identifier (`(2)` and `(3)` in Figure 2.6).

2.1.3 Tree Substitution: Substitution in Non-anchored Subtrees

The non-canonical elementary trees which violate Condition 2 have non-anchored subtrees. These non-anchored subtrees are first extracted by the algorithm in the previous section, and are then converted into multi-anchored trees by substituting a substitution node on the branching whose daughters do not consist of an anchor by every candidate tree for substitution, by a procedure called *tree substitution*, as shown in Figures 2.5. The candidate trees for the application of this process are selected from among all the canonical elementary trees and the ones obtained by the algorithm in the previous section. Substituted nodes are marked as *breaking points* to record the origination of these nodes. Note that non-anchored subtrees are not selected as candidates for substitution, because their root nodes originate from internal nodes of the elementary trees. This guarantees that the multi-anchored trees obtained by this process will satisfy Condition 2. These trees can be converted into single-anchored trees, to which we can apply the algorithm in Section 2.1.1, by the algorithm in the previous section.

The procedure `expand_tree_into_anchored_tree` in Figure 2.7 represents an algorithm for converting a non-anchored subtree NT into multi-anchored trees $\mathcal{MT}$. For each branching structure that consists of substitution nodes or foot nodes, one substitution node S is selected (`(1)` in Figure 2.7). The function `substitute` applies a substitution to the node *it* of every candidate tree which is substitutable for *S* (`(2)` in Figure 2.7).

Because the candidate trees for substitution include neither non-anchored subtrees nor auxiliary trees, the trees obtained by this process will satisfy Condition 2. When they take substitution at one substitution node, they also satisfy Condition 1 and are canonical trees; otherwise, they are multi-anchored trees and will be converted into canonical trees by the tree division. The resulting trees consist only of canonical trees because the tree substitution creates multi-anchored trees without non-anchored subtrees, which are divided into canonical trees.

2.1.4 Definition of ID Grammar Rules

In this section, we provide the definition of the grammar rules which emulate substitution and adjunction respectively and are thus called *substitution rule* and *adjunction rule* (in Figure 2.8). In

[3] It should be noted that the path from the foot node to the root node (*spine*: (Kasper et al. 1995)) in an auxiliary tree must not be cut because the spine represents the chain of head signs between the root node and the foot node, which are unified with the same internal node in the other trees.

```
INPUT:     a non-anchored tree $NT$
OUTPUT:    a set of multi-anchored trees $\mathcal{MT}$

procedure expand_tree_into_anchored_tree($NT$)
begin
  $\mathcal{BR}$ := na_br($NT$)
  $\mathcal{MT}$ := $\{NT\}$
  foreach $BR$ in $\mathcal{BR}$
    $S$ := select($BR$)                    $\cdots$ (1)
    $\mathcal{IT}$ := initial($S$)
    $\mathcal{MMT}$ := $\phi$
    foreach $MT$ in $\mathcal{MT}$
      $\mathcal{TMT}$ := substitute($MT$, $S$, $\mathcal{IT}$)$\cdots$ (2)
      mark($S$)
      $\mathcal{MMT}$ := $\mathcal{TMT} \cup \mathcal{MMT}$
    end foreach
    $\mathcal{MT}$ := $\mathcal{MMT}$
  end foreach
  return $\mathcal{MT}$
end
```

`na_br:`	returns the deepest branchings whose daughters do not consist of an anchor.
`initial:`	returns all candidate trees whose root node is the same as the leaf node.
`select:`	returns one of leaf nodes (default: the left-most one).
`substitute:`	causes substitution to the leaf node and returns a set of resulting multi-anchored trees.
`mark:`	marks the substituted node

Figure 2.7: An algorithm for converting a non-anchored subtree NT into a set of multi-anchored trees $\mathcal{MT}$

the figure, we give rules for the case where left-hand daughters correspond to the trunk nodes. Of course, there are symmetric rules for the right-hand case. These rules are independent of the input LTAG because they do not depend on any given characteristics for the LTAG.

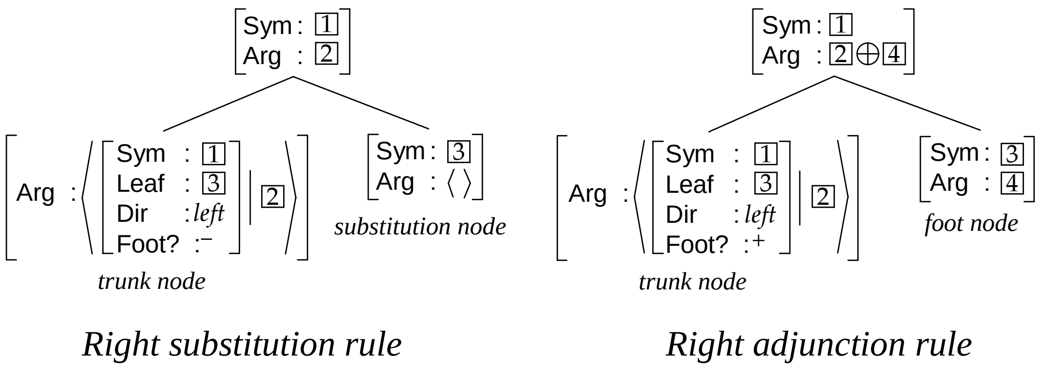


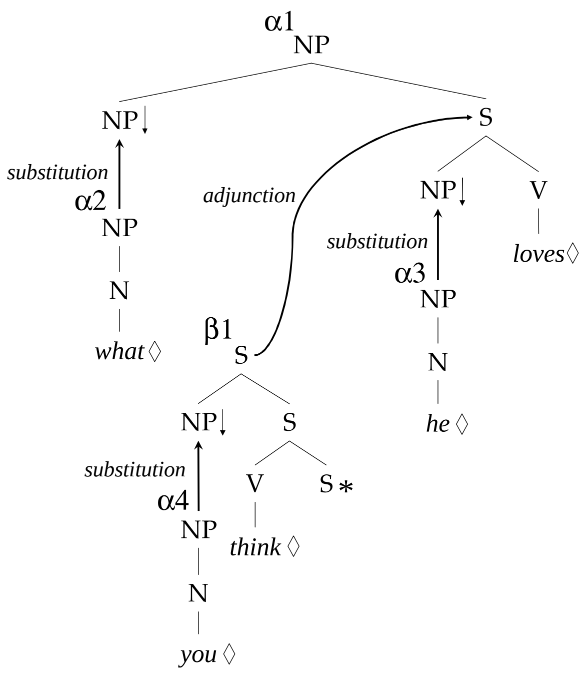
Figure 2.8: Grammar rules: the substitution rule and adjunction rule

Substitution rule: The Sym feature of the node to which substitution is applied must be identical to the Leaf feature ('$\boxed{3}$'[4] in the left-hand side of Figure 2.8) of the trunk node. The substitution rule percolates the tail elements ('$\boxed{2}$' in the left-hand side of Figure 2.8) of the Arg feature of the trunk node to the mother in order to continue constructing the tree. The value of the Arg feature of the node for substitution must be an empty stack $\langle\ \rangle$, because this node must be unified only with the node that corresponds to the root node of the initial tree. The value "$-$" or "$+$" of the Foot? feature explicitly determines whether the next rule to be applied is the substitution rule or the adjunction rule.

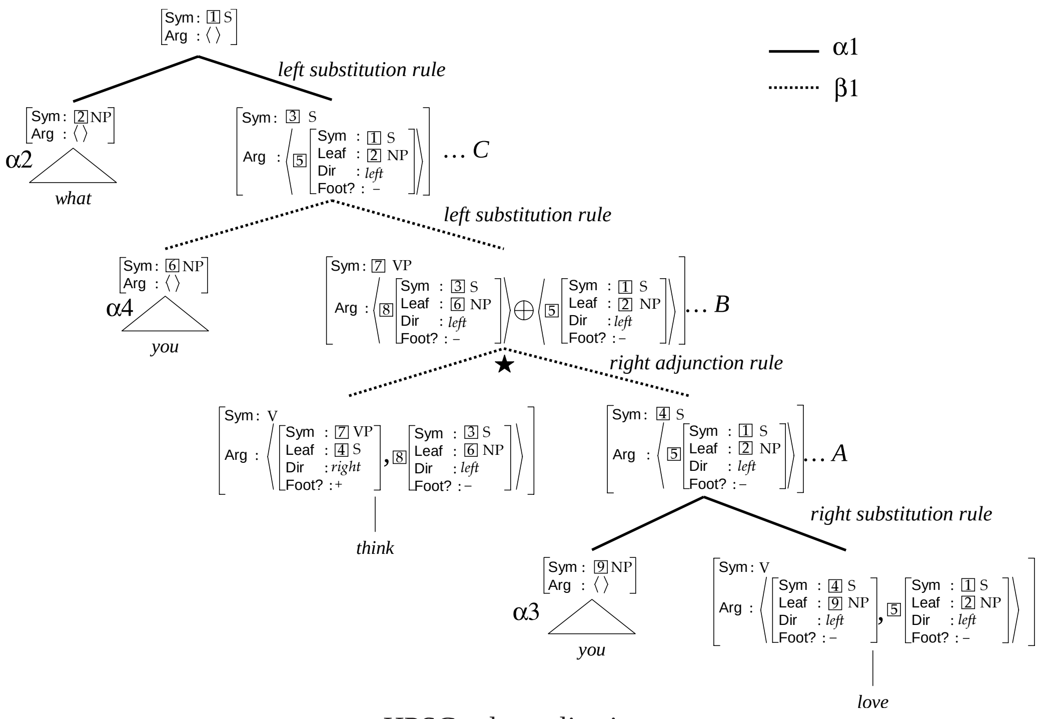
Adjunction rule: The Sym feature of a foot node must be identical to the Leaf feature of the trunk node ('$\boxed{3}$' in the right-hand side of Figure 2.8). The value of the Arg feature of the mother node is a concatenated stack of the Arg features of both of its daughters ('$\boxed{2}$' and '$\boxed{4}$' in the right-hand side of Figure 2.8). This allows the parser to construct the tree which corresponds to the adjoining tree and then to continue constructing the tree which corresponds to the adjoined tree.

Figure 2.9 shows examples of rule application. The solid lines indicate the adjoined tree ($\alpha 1$) and the dotted lines indicate the adjoining tree ($\beta 1$). The adjunction rule is applied in order to construct the branching marked with $\star$, where "*think*" takes as its argument the node having the Sym feature's value of S. By applying the adjunction rule, the Arg feature of the mother node B becomes a concatenated stack of the Arg features of both $\beta 1$, '$\boxed{8}$', and $\alpha 1$, '$\boxed{5}$.' Note that when the construction of $\beta 1$ has been completed, the Arg feature of the trunk node C will return to its former

[4]Recall that the numbers in tags express only the sharing of common values between the (sub-)feature structures, and hence the numbering does not convey any side effects.



LTAG derivation


Sym : 1 S
Arg : ⟨ ⟩
left substitution rule
α1
β1
Sym : 2 NP
Arg : ⟨ ⟩
α2
what
Sym : 3 S
Arg : ⟨ 5 [Sym : 1 S, Leaf : 2 NP, Dir : left, Foot? : −] ⟩
… C
left substitution rule
Sym : 6 NP
Arg : ⟨ ⟩
α4
you
Sym : 7 VP
Arg : ⟨ 8 [Sym : 3 S, Leaf : 6 NP, Dir : left, Foot? : −] ⟩ ⊕ ⟨ 5 [Sym : 1 S, Leaf : 2 NP, Dir : left, Foot? : −] ⟩
… B
★
right adjunction rule
Sym : V
Arg : ⟨ [Sym : 7 VP, Leaf : 4 S, Dir : right, Foot? : +], 8 [Sym : 3 S, Leaf : 6 NP, Dir : left, Foot? : −] ⟩
think
Sym : 4 S
Arg : ⟨ 5 [Sym : 1 S, Leaf : 2 NP, Dir : left, Foot? : −] ⟩
… A
right substitution rule
Sym : 9 NP
Arg : ⟨ ⟩
α3
you
Sym : V
Arg : ⟨ [Sym : 4 S, Leaf : 9 NP, Dir : left, Foot? : −], 5 [Sym : 1 S, Leaf : 2 NP, Dir : left, Foot? : −] ⟩
love

HPSG rule applications

Figure 2.9: LTAG and HPSG parsing of the phrase “*what you think he loves*”

state (A). We can continue constructing $\alpha 1$ in the same way as for the case where no adjunction rules have been applied.

2.1.5 Extension to FB-LTAG

The algorithms in Section 2.1 produce the conversion of an LTAG, and are easily extensible to handle an FB-LTAG grammar by merely storing a feature structure for each node, together with the symbol, in the Sym feature and Leaf feature. The grammar rules are also extended to execute the feature structure unification done in FB-LTAG.

2.2 Correspondence between HPSG-style Grammar and HPSG

The above algorithms provide a formal link between LTAG and HPSG-style grammar, which we defined as the computational architecture assumed in HPSG. In this section, we discuss the linguistic correspondence between an HPSG-style grammar and HPSG (Pollard and Sag 1994) according to the syntactic *head*, which is a central notion in HPSG. We will discuss the difference between an HPSG-style grammar converted from an implemented LTAG grammar and HPSG in Section 3.1

Elaboration on HPSG signs In order to derive HPSG from an HPSG-style grammar that we have obtained, we must elaborate on the *sign* of the HPSG-style grammar. As we have seen in 1.2, HPSG provides a modular specification of linguistic generalization by using *principles* and *ID schemata* in the context of the lexicalist framework.[5] On the other hand, our HPSG-style grammar implicitly captures some of the principles and ID schemata of the definition in Section 1.2 in the following way. *The Immediate Dominance Principle* is satisfied by the use of the ID grammar rules. In the ID grammar rules, *the Subcategorization Principle* is expressed by the structure-sharing of the Sym and Leaf features which correspond to the HEAD feature in HPSG. We should note that a non-empty value for the Arg feature of the foot node in the adjunction rules roughly corresponds to the SLASH feature in HPSG, which supports *the Nonlocal Feature Principle*. The Arg feature thus corresponds to concatenation of the SUBCAT and SLASH features of HPSG. Other principles, such as *the Head Feature Principle*, are implicitly encoded in a trunk of the tree structure of the LTAG elementary trees. We will extract such principles in the LTAG context and subdivide the ID grammar rules into HPSG rule schemata by analyzing feature percolation (Tateisi et al. 1998). There are the following two issues in order to further elaborate an HPSG-style grammar.

[5]HPSG-specific linguistic theories such as binding theory must be implemented in the obtained HPSG-style grammar by defining additional features or special mechanisms.

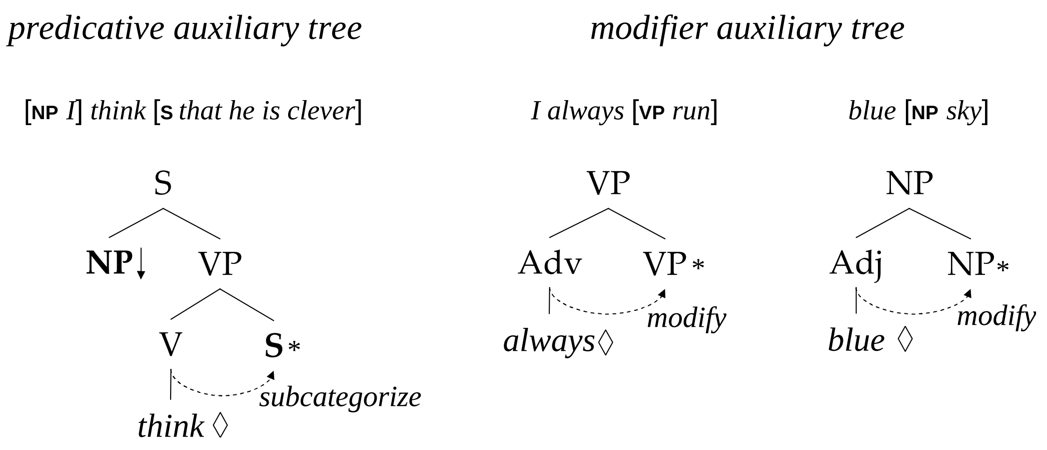


Figure 2.10: Predicative auxiliary tree for "*think*" and modifier auxiliary trees for "*always*" and "*blue*"

The distinction between predicative and modifier auxiliary trees Auxiliary trees in LTAG are of a predicative or a modifier type (Kroch 1989; Schabes and Shieber 1994), which introduces head-complement (or head-filler) relation and head-adjunct relation, respectively. More precisely, the former introduces a predicate that subcategorizes for a phrase of the category of its foot node, while the latter introduces a modifying, dislocated phrase, or a complement. This distinction is, in rough terms, made by determining which daughter is the head, a foot node or a trunk node. Tateisi et al. (Tateisi et al. 1998) distinguished these trees by manually analyzing feature percolation in auxiliary trees and by assigning HPSG rule schemata separately to each auxiliary tree.[6] In this dissertation, we consider that all trunk nodes are heads, that is, treat all auxiliary trees as predicative trees, but we can manually or semi-automatically determine the above categories by providing some linguistic cues or by analyzing feature percolation.

Head selection How we should implement the function `select` in Figure 2.6 that selects the anchor A is not entirely clear. Since most multi-anchored trees represent compound expressions or idioms, such as "*look up*" and "*kick the bucket*," this problem can be replaced with the problem of which word of a phrase is the syntactic/semantic head. In the HPSG framework, some of such compound expressions are handled as simplex entries as words with spaces (*e.g.*, "*look up*") (Copestake et al. 2002). However, in some idioms, their meanings are sometimes metaphori-

[6]The author wishes to thank Yuka Tateisi for her comments on distinction for predicative and modifier trees in her translation of the XTAG English grammar into an HPSG grammar

cal and thus non-decomposable into the parts of the expressions (*e.g.*, “*kick the bucket*,” compared with “*take advantage of*”). Riehemann (Riehemann 2001) proposed an approach to such idiosyncratic constructions, which licenses such constructions by having phrasal entries used when the phrase are constructed by a parser. We must also consider a similar issue to do with how we should implement the function `select` in Figure 2.7 that selects the leaf node S to be substituted. Since elementary trees with non-anchored subtrees represent constructions that require a specification beyond immediate dominance, such as *it-clefts* and *equative be*, this problem may be rephrased as one of finding which leaf node takes the dominant syntactic role and should be substituted in carrying out HPSG analysis. We currently simply select an anchor or a substitution node from the left-most node, though we can solve these problems by using linguistic ideas such as projection.

2.3 Proof of Strong Equivalence for Grammar Conversion from LTAG to HPSG-style Grammar

This section provides a formal proof for a strong equivalence between LTAG and an HPSG-style grammar converted from LTAG by our grammar conversion. In what follows, we first mention an informal sketch on how the strong equivalence is guaranteed for LTAG and the obtained HPSG-style grammar. We then proceed the formal proof, which comprises two parts. Part one proves that strong equivalence is guaranteed for the conversion from LTAG G to canonical LTAG G' by the tree division and the tree substitution. Part two proves that strong equivalence is guaranteed for the conversion from canonical LTAG G' to an HPSG-style grammar G''.

2.3.1 Informal Sketch on the Proof of the Strong Equivalence

In this section, we discuss how our algorithm guarantees strong equivalence between the grammar it obtains and the original grammar. In the obtained grammar, the grammar rules are applied only to those feature structures which correspond to nodes which are substitutable for/adjoinable with the canonical elementary trees of the original LTAG because the branchings encoded in the respective values of Arg specify the nodes to be subcategorized next. Strong equivalence also holds for the conversion of non-canonical elementary trees. For trees that violate Condition 1, we can distinguish the cut-off nodes from substitution nodes thanks to the identifiers, which allow recovery of the co-occurrence relation between the divided trees. For trees that violate Condition 2, we can identify those nodes to which substitution is to be applied in a combined tree because they are marked as breaking points, and thus consider the combined tree as two trees in the LTAG derivation.

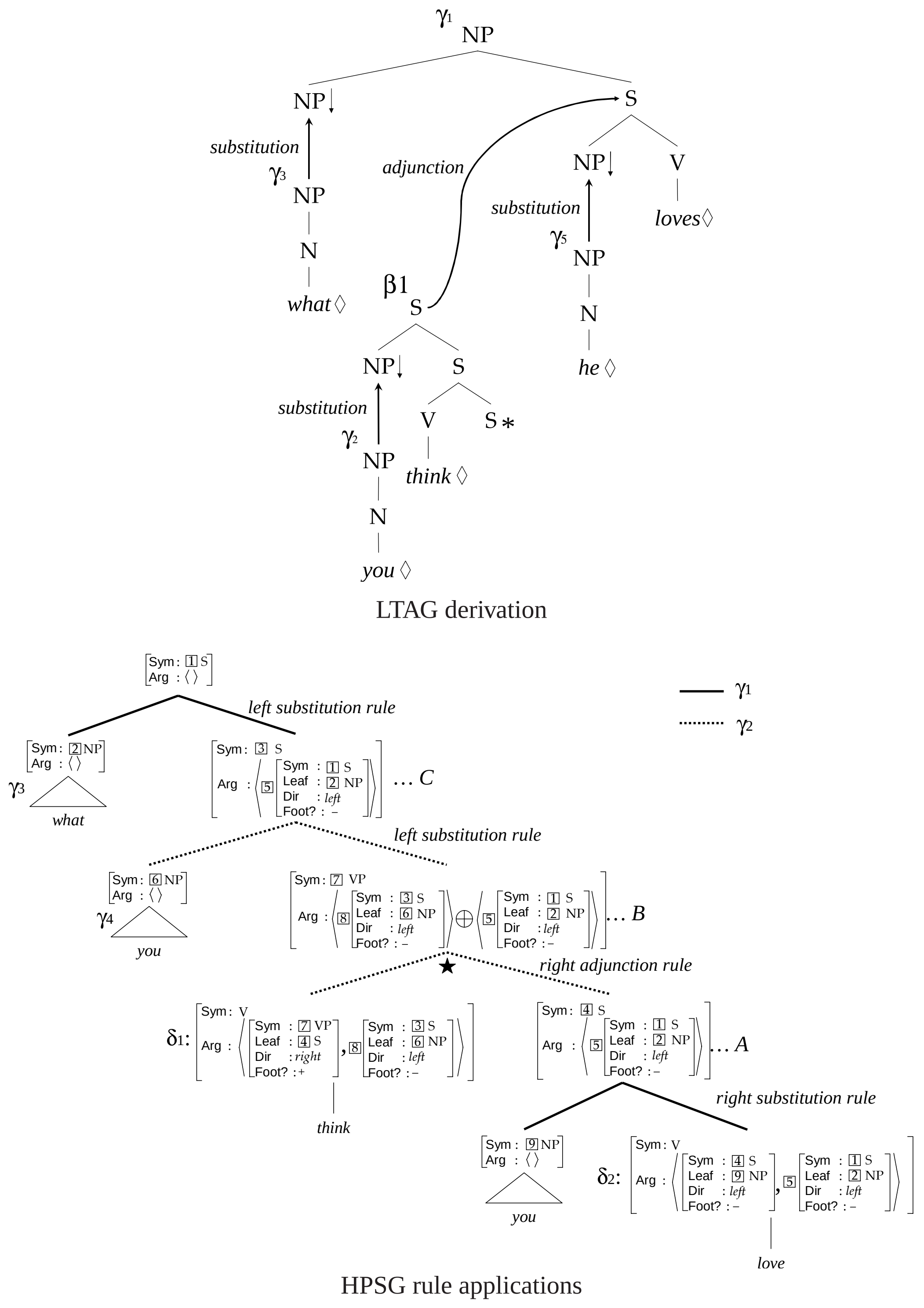


Figure 2.11: LTAG and HPSG parsing of the phrase “*what you think he loves*” (revisited)

We can thus avoid overgeneration by having the identifiers checked in the substitution rules, and avoid undergeneration by substituting *all* candidate trees for substitution nodes in the algorithm in Section 2.1.3.

Strong equivalence enables us to recover an LTAG derivation tree from an HPSG parse tree by following the history of rule applications and mapping each of them to substitution or adjunction. Let us take the case of Figure 2.11 as an example. We start by following the trunk node when the substitution rule was applied, or the foot node when the adjunction rule was applied. We then reach "*love*," and recognize it as the anchor of an elementary tree whose root node is identical to that of the parse tree. We then follow the path from the anchor to the root node to recognize γ^1 and combinations between γ^1 and other elementary trees. Since we start by finding an application of the substitution rule, we can map it to the substitution of γ^5 to γ^1 by recognizing the sibling node of the trunk node as the root node of γ^3 and by recursively recovering the partial derivation from the sibling subtree. Then, the next rule is the adjunction rule (marked with $\star$), and we find that the node A takes adjunction. We thus remember the length of the value of the Arg feature of the node A, and follow the trunk with handling rule applications as ones for the adjoining tree γ^2 until the length of the Arg feature is equal to that for the node A. This is the case at the node C. This implies that the construction of the adjoining tree γ^2 is completed at the node C. We restart the recognition of γ^1. After handling another application of the substitution rule, we reach the root node S, and complete the recognition of $\gamma 1$ and thus the whole derivation tree.

2.3.2 Definitions

We first define LTAG, according to the definition of TAG given by Vijay-Shanker (Vijay-Shanker 1987). We then define *a derivation tree*, which is a structural description of LTAG, and introduce the notion of strong equivalence.

We hereafter denote a tree as a set of pairs $\langle p, X \rangle$ where $p \in \mathcal{N}^*$, which is a free monoid of the set of natural numbers, and $X \in V$, which is a finite set of alphabets (Gorn 1962). For example, a tree in the left-hand side of Figure 2.3 is denoted as $\{(\epsilon, S), (\epsilon \cdot 1, NP)(\epsilon \cdot 2, VP), (\epsilon \cdot 2 \cdot 1, V), (\epsilon \cdot 2 \cdot 2, S), (\epsilon \cdot 2 \cdot 1 \cdot 1, think)\}$. An inequality $p \leq q$ is satisfied if and only if there is a $r \in \mathcal{N}^*$ such that $q = p \cdot r$. Another inequality $p < q$ is satisfied if and only if $p \leq q$ and $p \neq q$.

Definition 2.3.1 (Lexicalized Tree Adjoining Grammar (LTAG)) *A lexicalized tree adjoining grammar G is a quintuplet (Σ, NT, S, I, A) where Σ and NT are a finite set of terminal symbols and a finite set of nonterminal symbols, respectively, S is a distinguished nonterminal symbol called the start symbol, and I and A are a finite set of initial trees and a finite set of auxiliary trees,* *respectively.*[7]

Here, an elementary tree $\gamma \in A \cup I$ is a tree whose leaf nodes are labeled by $X \in NT \cup S$ or $x \in \Sigma$, and whose internal nodes are labeled by $X \in NT \cup S$. The symbol of one leaf node in an auxiliary tree $\beta \in A$ is identical to that of its root node, and is specially marked as a foot node. Note that at least one leaf node, called anchor, in an elementary tree γ is labeled with $x \in \Sigma$, and leaf nodes other than anchors and the foot node are marked as substitution nodes.

We hereafter use the notion of *an address* of a node in a tree. An address of a tree is a symbol that indicates a unique node in the tree.

Next, we define *a derivation* for an elementary tree γ. Let us denote a tree that is derived from an elementary tree γ by having substitution and adjunction into γ as γ'. When we produce γ' from an elementary tree γ by applying substitutions and adjunctions of several trees $\gamma'_1, \gamma'_2, \ldots, \gamma'_k$ to γ at k distinct addresses $a_1, a_2, \ldots, a_k$, the production is denoted by $\gamma' \to \gamma[a_1, \gamma'_1][a_2, \gamma'_2] \ldots [a_k, \gamma'_k]$ where $k \geq 1$, and $[a_i, \gamma'_i]$ indicates substitution at a_i of γ'_i if a_i is a substitution node, or indicates adjunction at a_i of γ'_i if a_i is an internal node. This production is called *a derivation* for γ if $a_1, a_2, \ldots, a_k$ include all addresses of the substitution nodes in γ. A derivation for γ without substitution and adjunction is denoted as $\gamma' \to \epsilon$. The set of all possible derivations D_G for LTAG G = (Σ, NT, S, I, A) is then denoted as follows:

$$\begin{aligned} D_G = \quad & \{\gamma'_i \to \epsilon \mid 1 \leq i \leq m, \gamma_i \in A \cup I, \gamma_i \text{ includes no substitution node.}\} \\ & \cup \{\gamma'_i \to \gamma_i[a_1, \gamma'_{i_1}][a_2, \gamma'_{i_2}] \ldots [a_k, \gamma'_{i_k}] \mid k \geq j \geq 1, i > m; \gamma_i, \gamma_{i_j} \in A \cup I; \\ & \qquad a_1, a_2, \ldots, a_k \text{ include all addresses of the substitution nodes in } \gamma_i\} \end{aligned}$$

We use the above notations to define *a derivation tree*, which represents the history of combinations of trees and is a structural description of LTAG.

Definition 2.3.2 (derivation tree) *A derivation tree for LTAG G = (Σ, NT, S, I, A), Υ_G, is formed from any subset of the set of all derivations D_G by uniquely relabeling identical elementary trees in the derivations of the subset. A derivation tree Υ_G must satisfy the following conditions:*

- *Because γ_i can be adjoined or substituted once, γ'_i can appear once respectively in the left-hand side and the right-hand side of derivations in Υ_G except for the one distinguished elementary tree γ_S, which is the root of the derivation tree Υ_G. The condition implies that trees cannot be substituted or adjoined to more than one node.*

[7]For simplicity, we omit the notion of adjoining constraints and the proof considering the adjoining constraints in this dissertation, and then assume all internal nodes take selective adjoining constraints.

- *γ'_S can appear once in the left-hand side of the derivation.*
- *The inequality $i > i_j \geq 1$ is necessary to avoid cyclic applications of substitution and adjunction among elementary trees.*

Next, we give the definition of strong equivalence between two grammars G_1 and G_2. Strong equivalence is intuitively that the two grammars generate equivalent *structural descriptions*, *structural description* which are the most informative data structures given by G; examples of structural descriptions are parse trees by CFG and derivation trees by LTAG. The following definition follows from the one by Miller (Miller 1999, p. 7).

Definition 2.3.3 (strong equivalence) *Let the set of all possible structural descriptions given by two given grammars G_1 and G_2 be $T_D(G_1)$ and $T_D(G_2)$. The two given grammars G_1 and G_2 are strongly equivalent if and only if there is a bijective (i.e., one-to-one and onto) mapping from a structural description of G_1, $\Upsilon_{G_1} \in T_D(G_1)$, to a structural description of G_2, $\Upsilon_{G_2} \in T_D(G_2)$.*

In what follows, we assume that structural descriptions of LTAG are derivation trees in which the root node of γ_S is labeled by the start symbol *S* in the definition 2.3.2.

2.3.3 Proof for Tree Division and Tree Substitution

In this section we give a proof that strong equivalence is guaranteed for grammars before and after the two tree transformations.

We omit the proof of the substitution procedure, because the tree substitution depicted in Figure 2.12 is exactly the same as the one that Schabes and Waters (Schabes and Waters 1995, pp. 494–495) defined and proved in their strong lexicalization procedure of CFG into Lexicalized Tree Insertion Grammar.

The tree division procedure is formalized in the following lemma.

Lemma 2.3.1 (The tree division) *Let $G = (\Sigma, NT, S, I, A)$ be an LTAG. Let $\gamma \in A \cup I$ be an elementary tree and let μ be an internal node not on the spine with address p of γ that is labeled by X. We divide γ at μ and obtain two trees γ^u, γ^v as follows. Let γ^u be a subtree except that a node labeled by $Y \notin NT \cup S$ is added to its root node, and let γ^v be a supertree, except that the symbol of μ is relabeled by the symbol $Y \notin NT$ and by marking it for substitution as shown in Figure 2.13. Define $G' = (\Sigma, NT \cup \{Y\}, S, I', A')$ where I' and A' are created as follows:*

If $\gamma \in I$ then $I' = (I - \{\gamma\}) \cup \{\gamma^u, \gamma^v\}$ and $A' = A$
If $\gamma \in A$ then $I' = I \cup \{\gamma^v\}$ and $A' = (A - \{\gamma\}) \cup \{\gamma^u\}$

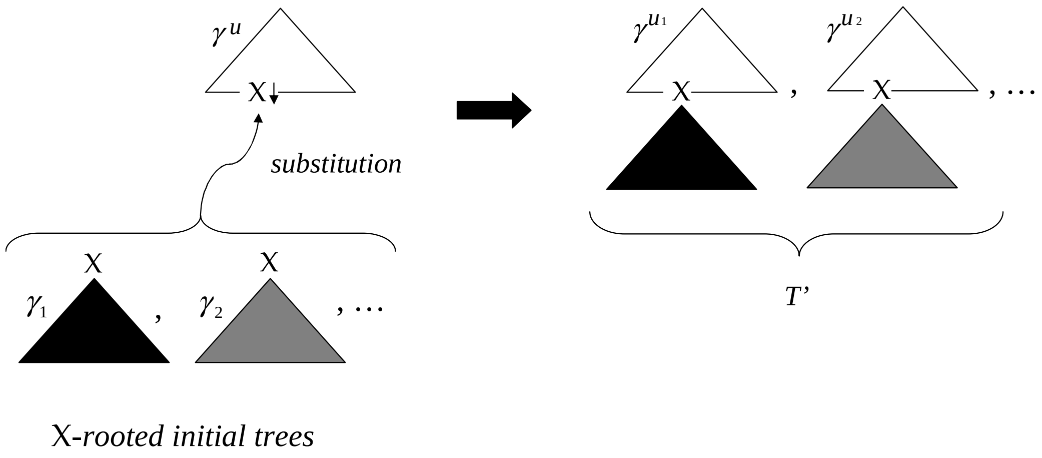


Figure 2.12: Sketch for the tree substitution

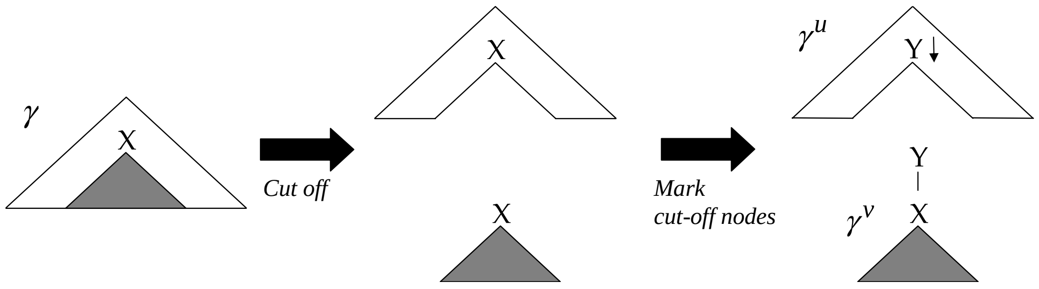


Figure 2.13: Sketch for the tree division

Then, G' is strongly equivalent to G; that is, there is a one-to-one onto mapping from the set of derivation trees $T_D(G')$ generated by G' to the set of derivation trees $T_D(G)$ generated by G for the same sentence.

Proof We show that there is a one-to-one mapping from a derivation tree $\Upsilon_{G'} \in T_D(G')$ to a derivation tree $\Upsilon_G \in T_D(G)$.

Assume each derivation tree $\Upsilon_{G'}$ consists of elementary trees $\{\gamma_1, \ldots, \gamma_n\}$, $\gamma_j \in A \cup I$ for $1 \leq j \leq n$. Then, we can represent the derivation tree $\Upsilon_{G'}$ by the set of derivations as shown in the definition 2.3.2.

Because we assume that a derivation tree is rooted by an elementary tree whose symbol of the root node is S, γ^v cannot appear as a root of derivation trees. Every occurrence of γ^v in $\Upsilon_{G'}$ must thereby accompany with γ^u and vice versa. In the following procedure, we construct a one-to-one mapping from $\Upsilon_{G'}$ to Υ_G by replacing every occurrence of γ^u which takes a substitution of γ^v with γ in derivations in $\Upsilon_{G'}$.

1. When $\gamma^u \notin \{\gamma_1, \ldots, \gamma_n\}$ or $\gamma^v \notin \{\gamma_1, \ldots, \gamma_n\}$, $\Upsilon_{G'}$ includes neither γ^u nor γ^v. $\Upsilon_{G'}$ therefore consists of $\gamma_i \in (A \cup I - \{\gamma\}) \subset A \cup I$, there is exactly the same derivation tree Υ_G in $T_D(G)$.

2. When $\gamma^u \in \{\gamma_1, \ldots, \gamma_n\}$, we can construct one derivation tree Υ_G from Υ'_G as follows.

   (a) We first replace every occurrence of γ'^u in the right-hand side of derivations with γ'.

   (b) We next replace every derivation whose left-hand side is either γ'^u or γ'^v.

      i. When a root node with address ϵ of γ^v takes substitution or adjunction, a pair of two derivations whose left-hand side is γ'^u and γ'^v is denoted as $\gamma'^u \rightarrow \gamma^u[a_1, \gamma'_1] \ldots [a_{h-1}, \gamma'_{h-1}][p, \gamma'^v]$ and $\gamma'^v \rightarrow \gamma^v[\epsilon, \gamma'_h][b_{h+1}, \gamma'_{h+1}] \ldots [b_k, \gamma'_k]$, where $k > h \geq 1$. Here we assume $a_i \ngeq p$ for $1 \leq i < h$ without loss of generality. We replace these two derivations with the following derivation:

$$\gamma' \rightarrow \gamma[a_1, \gamma'_1] \ldots [a_{h-1}, \gamma'_{h-1}][p, \gamma'_h][p \cdot 1 \cdot b_{h+1}, \gamma'_{h+1}] \ldots [p \cdot 1 \cdot b_k, \gamma'_k]$$

      ii. If a root node with address ϵ in γ^v takes neither adjunction nor substitution, we can also replace a pair of two derivations whose left-hand side are respectively γ'^u and γ'^v with one derivation whose left-hand side is γ' in a similar way as above.

   (c) By repeating the above replacements at most the number of pairs of two derivations for γ^u and γ^v, we can obtain a set of derivations Υ_G without γ'^u and γ'^v. The replacement in (a) is valid since γ^u includes both root node and foot node of γ, and thus γ can substitute or adjoin every node at which γ^u does. In the procedure (b), we replace exactly the same number of γ'^u as the procedure (a). The resulting derivations including γ' is valid in G because γ' appear only once in the right-hand side and the left-hand side of the derivations, respectively.

The resulting derivation tree Υ_G is the same as $\Upsilon_{G'}$ except that every occurrences of γ^u which takes a substitution of γ^v with γ. Since γ^u which takes a substitution of γ^v is the same as γ except that one internal node is added, this does not cause effect on the frontier string. Also, when $\Upsilon^1_{G'}$ and $\Upsilon^2_{G'}$ are mapped to Υ^1_G and Υ^2_G that are equivalent with each other, $\Upsilon^1_{G'}$ and $\Upsilon^2_{G'}$ are also equivalent owing to the formulation of the above mapping.

On the other side, we can also construct a one-to-one onto mapping from Υ_G to $\Upsilon_{G'}$ by replacing every occurrence of γ in Υ_G by γ^u which takes a substitution of γ^v.

In this way, we can construct a one-to-one onto mapping from a derivation tree $\Upsilon_{G'} \in T_D(G')$ to a derivation tree $\Upsilon_G \in T_D(G)$ for the same sentence. This indicates that G is strongly equivalent to G'. $\square$

2.3.4 Proof for Conversion from Canonical LTAG to HPSG-style Grammar

We prove that strong equivalence is guaranteed for a conversion from canonical LTAG G to an HPSG-style grammar G'. We first define *an HPSG parse*, which is a structural description of an HPSG-style grammar. We then prove strong equivalence by giving a bijective mapping from a derivation tree by G to an HPSG parse by G'.

Definition 2.3.4 (HPSG-style grammar converted from LTAG) *Given a canonical LTAG G = (Σ, NT, S, I, A), an HPSG-style grammar G' converted from G is denoted by a sextuplet (Σ, NT, S, Δ_I, Δ_A, R) where $\delta_i \in \Delta_I$ and $\delta_j \in \Delta_A$ are lexical entries converted from $\gamma_i \in I$ and $\gamma_j \in A$, respectively, and R denotes the substitution and adjunction rules. δ_i is denoted as follows: δ_i = (s_0, (s_1, l_1, d_1, t_1), $\ldots$, (s_k, l_k, d_k, t_k)) where $k \geq 1$, $s_0 \in \Sigma \cup NT$ is the symbol of the mother node of the anchor in γ_i, and $s_j \in \Sigma \cup NT$, $l_j \in \Sigma \cup NT$, $d_j \in \{right, left\}$, $t_j \in \{+, -\}$ are values of Sym, Leaf, Dir, and Foot? features in the j-th element of the Arg feature in δ_i. When the length of the Arg feature of δ_i is 0, δ_i is denoted as δ_i = (s_0, ϕ).*

First, we introduce the notion of *origination* for the Sym and Leaf features in HPSG lexical entries in order to define *an HPSG parse*, which represents the histories of rule applications to lexical entries and is a structural description of an HPSG-style grammar. We define the origination of the feature in δ_i as $\langle p, \gamma_i \rangle$, which indicates that the value of the feature originates from the symbol of a node with address p in γ_i. Figure 2.14 shows examples of the originations for HPSG lexical entries δ_1 and δ_2 converted from LTAG elementary trees γ_1 and γ_2 for "*think*" and "*love*." In the figure, a subscript attached to an internal node in the elementary trees indicates the address of the node.

Next, we define *a rule history* for δ_i, which is a history of rule applications to a lexical entry δ_i. We take each rule application to δ_i and its ancestor nodes as an element of the sequence of rule applications for δ_i if and only if the applied rule pops an element that originates from an element of the Arg feature in δ_i. For example in Figure 2.11, a lexical entry for "*think*" and its ancestor nodes A and B take the right adjunction rule, the left substitution rule, and the left substitution rule in this order. Among these applied grammar rules, the right adjunction rule and the first left substitution

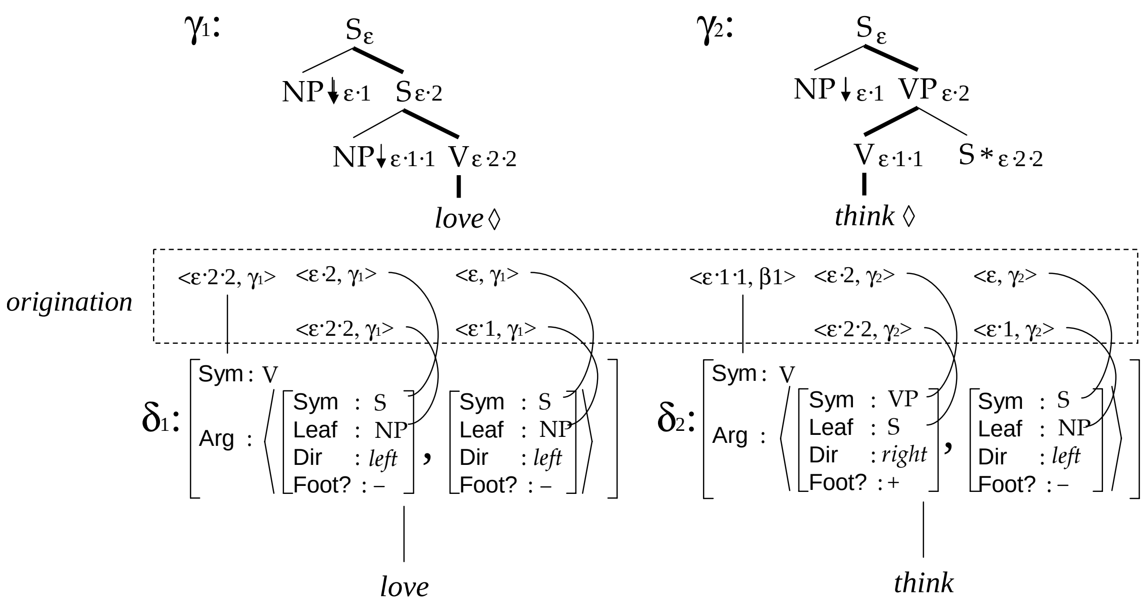


Figure 2.14: The origination for the Sym and Leaf features in the HPSG lexical entries converted from elementary trees

rules pop an element that originates from an element of the Arg feature in the lexical entry. Thus, applications of these grammar rules are chosen as an element of the sequence of rule applications for the lexical entry of "*think*."

Assuming that δ_i is denoted as the one given in definition 2.3.4, when $\langle a_j, \gamma_i \rangle$ is the origination of l_j and $\langle b, \gamma_{i_j} \rangle$ is the origination of s_{i_j} unified with l_j in the grammar rule, a sequence of rule applications for δ_i is denoted as follows:

$$\delta'_i \rightarrow \delta_i[x_{i_1}, y_{i_1}][x_{i_2}, y_{i_2}] \ldots [x_{i_k}, y_{i_k}],$$

where $k \geq j \geq 1$, (x_{i_j}, y_{i_j}) is (a_i, δ'_{i_j}) if $t_j = -$, or (b, δ_{i_j}) if $t_j = +$. Each $[x_{i_j}, y_{i_j}]$ expresses an application of one grammar rule to the lexical entry δ_i and its ancestors, and assume that these are sorted according to the order of the rule applications in the bottom-up manner. When $x_{i_1}, x_{i_2}, \ldots x_{i_k}$ include b where $k \geq h \geq 1$ and $t_h = +$ or a_j where $k \geq j \geq 1$, and $t_j = -$ in the sequence of rule applications for δ_i, we call the sequence of rule applications *a rule history* for δ_i. When the length of the Arg feature of δ_i is 0, a rule history for δ_i is denoted by $\delta'_i \rightarrow \epsilon$. For example in Figure 2.11 and the tree naming given in Figure 2.14, rule histories for a lexical entry

δ_1, δ_2 for "*love*" "*think*" in the HPSG parse tree can be denoted as follows:

$$\delta'_1 \rightarrow \delta_1[\epsilon \cdot 2 \cdot 2, \delta'_5][\epsilon \cdot 1, \delta'_3].$$

$$\delta'_2 \rightarrow \delta_2[\epsilon \cdot 2, \delta_1][\epsilon \cdot 1, \delta'_4]$$

where $delta_3$, $delta_4$, and $delta_5$, are lexical entries for "*what*", "*you*", "*love*." Note that the application of the right adjunction rule in this example is denoted by $[\epsilon \cdot 2, \delta_1]$, which includes the information in the adjoined tree γ_1.

Lemma 2.3.2 *Given an HPSG-style grammar G' = (Σ, NT, S, Δ_I, Δ_A, R), a rule history for $\delta_i \in \Delta_I \cup \Delta_A$ must be the following form.*

- *When the length of the* Arg *feature of δ_i is 0, $\delta'_i \rightarrow \epsilon$*
- *When the length of the* Arg *feature of δ_i is not 0, and*
  - *When $\delta_i \in \Delta_I$, $\delta'_i \rightarrow \delta_i[a_1, \delta'_{i_1}][a_2, \delta'_{i_2}] \ldots [a_k, \delta'_{i_k}]$.*
  - *When $\delta_i \in \Delta_A$, $\delta'_i \rightarrow \delta_i[a_1, \delta'_{i_1}] \ldots [a_{h-1}, \delta'_{i_{h-1}}][b, \delta_{i_h}][a_{h+1}, \delta'_{i_{h+1}}] \ldots [a_k, \delta'_{i_k}]$ where t_h = $+$.*

Proof When the length of the Arg feature of δ_i is 0, no rule application is assigned as a rule application for δ_i because it is defined according to elements in the Arg feature. The rule history for δ_i is thus denoted as $\delta'_i \rightarrow \epsilon$.

When $\delta_i \in \Delta_I$, the elements in the Arg feature of δ_i keep their order until the grammar rules consume all the elements. This is because both substitution and adjunction rules do not change the order of the Arg feature, and also do not remove an element of the Arg feature without unifying it with another node. The rule history for δ_i is thus denoted as $\delta_i[a_1, \delta'_{i_1}][a_2, \delta'_{i_2}] \ldots [a_k, \delta'_{i_k}]$.

When $\delta_i \in \Delta_A$, the elements in the Arg feature of δ_i keep their order during rule applications until the grammar rules consume all the elements as in the case where $\delta_i \in \Delta_I$. One difference is that there exists h so that $1 \leq h \leq k$ and $t_h = +$, in other words, it includes exactly one element l_h where t_h = $+$ that originates a foot node. The rule history for δ is then denoted by $\delta'_i \rightarrow \delta_i[a_1, \delta'_{i_1}] \ldots [a_{h-1}, \delta'_{i_{h-1}}][b, \delta_{i_h}][a_{h+1}, \delta'_{i_{h+1}}] \ldots [a_k, \delta'_{i_k}]$ where t_h = $+$. □

By using lemma 2.3.2, we can define the set of rule histories by G' = (Σ, NT, S, Δ_I, Δ_A, R) as follows:

$$\begin{aligned} D_{G'} = \quad & \{\delta'_i \rightarrow \epsilon \mid 1 \leq i \leq m, \gamma_i \in I, \text{the length of the Arg feature of } \delta_i \text{ is } 0\ \} \\ & \cup \{\delta'_i \rightarrow \delta_i[a_1, \delta'_{i_1}] \ldots [a_k, \delta'_{i_k}] \mid m < i \leq n, k \geq j \geq 1, \delta_i, \delta_{i_j} \in \Delta_I\} \\ & \cup \{\delta'_i \rightarrow \delta_i[a_1, \delta'_{i_1}] \ldots [a_{h-1}, \delta'_{i_{h-1}}][b, \delta_{i_h}][a_{h+1}, \delta'_{i_{h+1}}] \ldots [a_k, \delta'_{i_k}] \\ & \qquad \mid n < i, k \geq h \geq 1, k \geq j \geq 1, t_h = +, \delta_i \in \Delta_A, \delta_{i_j} \in \Delta_I\} \end{aligned}$$

We use the above notations to define *an HPSG parse*,[8] which represents the history of rule applications and is a structural description of an HPSG-style grammar.

Definition 2.3.5 (HPSG parse) *Given an HPSG-style grammar G' = (Σ, NT, S, Δ_I, Δ_A, R) converted from G, an HPSG parse $\Psi_{G'}$ is formed from a subset of the set of all rule histories $D_{G'}$ by renaming identical lexical entries in the rule histories of the subset uniquely. An HPSG parse $\Psi_{G'}$ must satisfy the following conditions:*

- *δ'_i where $\delta_i \in \Delta_I$ can appear once respectively in the left-hand side and the right-hand side of rule histories except for the one distinguished lexical entry δ_S where δ'_S appears once in the left-hand side of the rule history for δ_S.*
- *δ'_i where $\delta_i \in \Delta_A$ must appear only once in the left-hand side of the rule history for δ_i.*
- *$1 \leq i_j < i$ for the rule history for $\delta_i \in \Delta_I$.*
- *$1 \leq i_j < i$ where $j \neq h$, and $i_h > i$, for the rule history for $\delta_i \in \Delta_A$.*

The third and fourth conditions are necessary to avoid cyclic applications of grammar rules to lexical entries.

Lemma 2.3.3 *Let G = (Σ, NT, S, I, A) be a canonical LTAG and G' = (Σ, NT, S, Δ_I, Δ_A, R) be an HPSG-style grammar converted from G. Then, we can map a derivation tree Υ_G by G one-to-one onto to an HPSG parse $\Psi_{G'}$ by G'.*

Proof We first show a mapping from $\Psi_{G'}$ to a set of derivations $\Upsilon_{G'}$, and then show that $\Upsilon_{G'}$ is a valid derivation by G. Suppose an HPSG parse satisfying definition 2.3.5. We can map it one-to-one to a set of derivations $\Upsilon_{G'}$ with the following procedure. For each δ_i where $\delta_i \in \Delta_A$, we eliminate $[b, \delta_{i_h}]$, which corresponds to an application of the adjunction rule, and insert the element $[b, \delta'_i]$ to the right-hand side of the rule history for δ_{i_h} at the position immediately after $[b \cdot x, \delta'_{i-1}]$ where x = 1 or 2. Then, we obtain a set of derivations $\Upsilon_{G'}$ by replacing δ_{i_j} and δ'_{i_j} with γ_{i_j} and γ'_{i_j} in the rule history for δ_i and by regarding it as the derivation for γ_i in $\Upsilon_{G'}$. This mapping is one-to-one because the operation pair of eliminating $[b, \delta_{i_h}]$ and adding $[b, \delta'_i]$ is a one-to-one mapping.

Following the definition 2.3.2, we show that $\Upsilon_{G'}$ is a valid derivation tree by G. First, every substitution and adjunction in the derivations in $\Upsilon_{G'}$ must be valid in G. Since the substitution

[8]We omit the proof showing that an HPSG parse by G corresponds to a unique parse tree derived by G.

and adjunction rules preserve the order of the elements in the Arg feature of δ_i, substitution rules always unify the symbol of the substitution node with the symbol of the root node of γ_{i_j}. This unification represents the same constraint as the one imposed by substitution. We can give an analogous argument for an adjunction rule. The substitution and adjunction in the derivations in $\Upsilon_{G'}$ are then valid in G. Second, all addresses in the substitution nodes of γ_i must be included in the derivation for γ_i. This is apparently guaranteed by definition of the rule history for δ_i. Third, γ'_i can appear only once respectively in the right-hand side and the left-hand side of the derivations. This is apparently guaranteed for γ'_i where $\gamma_i \in I$ by definition 2.3.5, and is guaranteed for γ'_i where $\gamma_i \in A$ because δ'_i does not appear in the right-hand side of rule histories, $[b, \delta_{i_h}]$ appears only once in the rule history for δ_i, and the elimination of $[b, \delta_{i_h}]$ accompanies the addition of $[b, \gamma'_i]$ once to the right-hand side of the derivation for γ_{i_h}. Fourth, the elements in the right-hand side of the derivation for γ_i must be $[a_j, \gamma'_{i_j}]$ where $i_j < i$. This is apparently guaranteed for γ'_i where $\gamma_i \in I$ by definition 2.3.5, and is guaranteed for γ'_i where $\gamma_i \in A$ because the addition of $[b, \gamma'_i]$ for the derivation for γ'_{i_h} satisfies $i_h > i$ from definition 2.3.5.

The frontier string is preserved before and after this mapping from $\Psi_{G'}$ to $\Upsilon_{G'}$, because δ_i stores the same linear precedence constraints between δ_i and δ_j for $i \neq j$ as the constraints between γ_i and γ_j. Thus, an HPSG parse $\Psi_{G'}$ by G' is mapped one-to-one onto a derivation tree $\Upsilon_{G'}$ that is valid in G.

We can construct a mapping from Υ_G onto an HPSG parse Ψ_G by inverting the procedure for the above mapping from $\Psi_{G'}$ onto $\Upsilon_{G'}$. The obtained Ψ_G is a valid HPSG parse by G' because we can give an analogous argument for the validity of the rule histories in Ψ_G.

2.4 Chapter Summary

We proposed an algorithm for the conversion of grammars from an arbitrary FB-LTAG grammar into a strongly equivalent HPSG-style grammar. Our algorithm first convert LTAG elementary trees into a set of tree structures that have only one word and can be decomposed into immediate constituency. We then convert the tree structures into HPSG feature structures by encoding the tree structures in stacks. A set of pre-determined rules manipulates the stack to emulate substitution and adjunction. The nature of strong equivalence guaranteed by the grammar conversion enables us to obtain parsing results of an LTAG grammar from parsing results of the HPSG-style grammar obtained by conversion. The definition of strong equivalence and a formal proof on the strong equivalence between the original and obtained grammars are also provided. The obtained grammar successfully abstracted away surface differences on computation devices that underlie the two formalisms. Our method thus enables the sharing of LTAG resources with the HPSG community, the application of HPSG technologies to LTAG grammars, and the clarification of the differences between linguistic analysis according to the two grammar formalisms.

Chapter 3

Experiments on Collaboration between the LTAG and HPSG formalisms

This chapter demonstrates applications of grammar conversion to collaboration between the LTAG and HPSG formalisms. Section 3.1 shows experiments on grammar resource sharing, which converts a sizable LTAG grammar into an HPSG-style grammar, and shows specification of the obtained HPSG-style grammar. Section 3.2 conducts comparison between parsers for the LTAG and HPSG formalisms, and clarifies their algorithmic differences that cause performance difference.

3.1 Experiments on Grammar Resource Sharing

We applied our conversion algorithm to the latest version of the XTAG English grammar (XTAG Research Group 2001),[1] a large-scale LTAG grammar for English. We successfully converted all elementary trees[2] in the XTAG English grammar to HPSG lexical entries. Table 3.1 shows the classification of the elementary trees of the XTAG English grammar according to the conditions we introduced in Section 2.1. In the table, $\mathcal{A}$ shows the number of canonical elementary trees, while $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$ respectively show the number of trees that violate only Condition 1, only Condition 2, and both conditions. The second row shows the number of the obtained HPSG lexical entries converted from the LTAG elementary trees.

[1] We used the grammar attached to the latest distribution of the LTAG parser which we used in the parsing experiment. This parser is available at: `ftp://ftp.cis.upenn.edu/pub/xtag/lem/lem-0.14.0.tgz`

[2] These elementary trees should more strictly be called *elementary tree templates*. That is, elementary trees are abstracted from lexicalized trees, and one elementary tree template is defined for one syntactic construction, which is assigned to a number of words.

Table 3.1: Classification of elementary trees in the XTAG English grammar (LTAG) and lexical entries converted from them (HPSG)

Grammar	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\mathcal{D}$	**Total**
LTAG	326	763	54	50	1,193
HPSG	326	1,989	1,083	2,474	5,872

Table 3.2: The classification of the non-canonical elementary trees in Table 3.1: multi-anchored trees (corresponding to $\mathcal{B}$)

Construction	**# of trees**
Compound expressions	413
Verb with PP	194
Idioms	140
Others	16
Total	**763**

Table 3.2 shows how multi-anchored elementary trees are employed in the XTAG English grammar. The table shows that they are mainly used for compound expressions or idioms. Although such multi-word expressions are reported to be an important issue in the HPSG formalism (Riehemann 2001; Copestake et al. 2002), the obtained grammar is able to handle them when multi-anchored trees that represent multi-word expressions are converted into multiple lexical entries. Another case of multi-anchored trees is multi-anchored trees for verbs that take a prepositional phrase as their complement, in which a preposition is specified as another anchor. Such a case is shown in Figure 3.1.[3] The obtained grammar expresses this construction by cut-off nodes to require specified subtrees. In linguistic specifications in HPSG, on the other hand, such constraints are expressed by having a PFORM feature, which takes the values that represent the type of the corresponding prepositional phrase. This HPSG account seems to be consistent with the obtained grammar, that is, the LTAG analysis.

Table 3.3 shows the grammatical phenomena represented by elementary trees with non-

[3]We borrow examples in this section from elementary trees included in the current version of the XTAG English grammar (XTAG Research Group 2001). For simplicity, we omit some leaf node anchored by an empty category and an internal node that has such a leaf node.

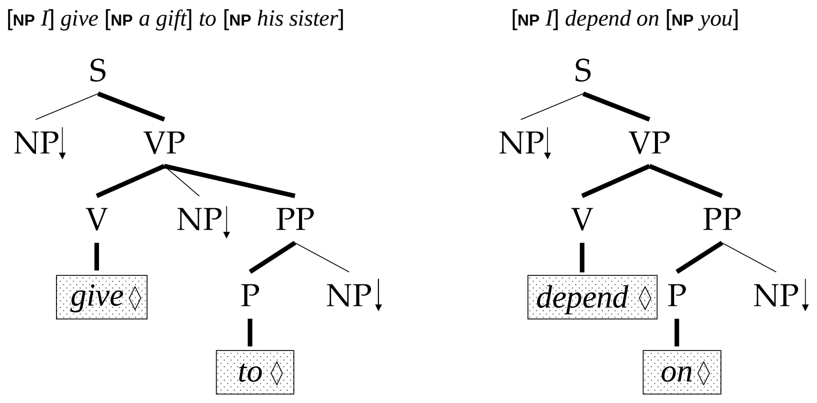

[NP I] give [NP a gift] to [NP his sister]
S
NP↓
VP
V
NP↓
PP
give ◊
P
NP↓
to ◊
[NP I] depend on [NP you]
S
NP↓
VP
V
PP
depend ◊
P
NP↓
on ◊

Figure 3.1: Non-canonical elementary trees for verbs that take a PP complement (the preposition is specified as another anchor)

Table 3.3: The classification of the non-canonical elementary trees in Table 3.1: trees with non-anchored subtrees (corresponding to $\mathcal{C} \cup \mathcal{D}$)

Construction	**# of trees**
Verb with PP	85
It-cleft	12
Others	7
Total	**104**

anchored subtrees. These elementary trees express constructions requiring specifications beyond immediate dominance. As we can see in Table 3.1 ($\mathcal{C} \cup \mathcal{D}$), these trees are expanded to include quite a large number of lexical entries. This result leads us to expect that these constructions might be difficult to handle in the standard HPSG formalism. The most common case of such constructions is single-anchored elementary trees for verbs that take a prepositional phrase as their complements. These elementary trees have non-anchored subtrees that express the expanded PP structure (Figure 3.2) (see also discussion on PP complement verbs (XTAG Research Group 2001, pp. 117–121)). This construction allows the extraction of the object of the preposition, and expresses the verb taking the object of the preposition as its argument rather than taking the PP as shown in Figure 3.2. In the HPSG formalism, on the other hand, this NP extraction is explained by using the SLASH feature, and the predicate-argument relation between the verb and the object

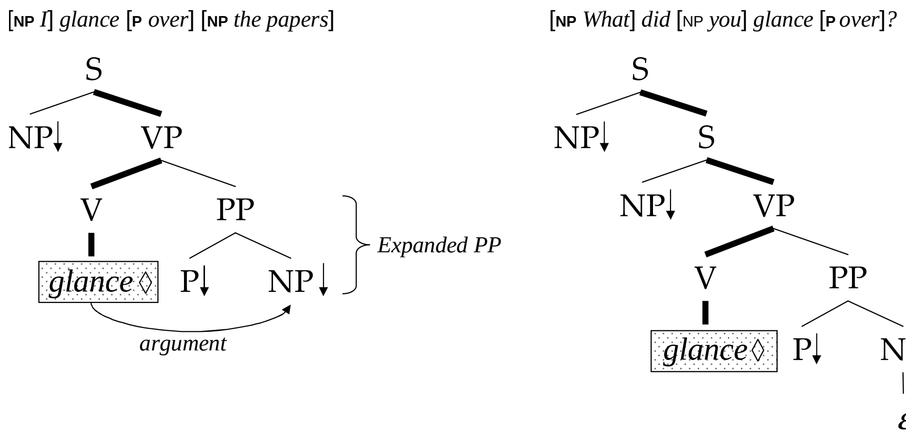

[NP I] glance [P over] [NP the papers]
[NP What] did [NP you] glance [P over]?
S
NP↓
VP
V
PP
glance◊
P↓
NP↓
Expanded PP
argument
S
NP↓
S
NP↓
VP
V
PP
glance◊
P↓
NP
ε

Figure 3.2: A non-canonical elementary tree for a verb that takes PP complement and its syntactic alternation of wh-moved object of a P

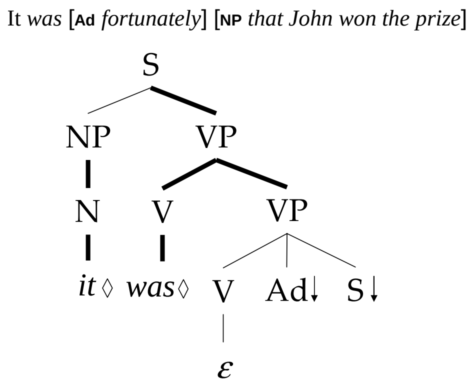


Figure 3.3: A non-canonical elementary tree that expresses it-cleft without gap

of the preposition is expressed by linking the argument in the predicative-argument structure (the CONTENT feature) with the object of the preposition. This construction is hence explained differently in LTAG and HPSG. Further case is a kind of *it-clefts* without gaps (Figure 3.3) (see also discussion on it-clefts (XTAG Research Group 2001, pp. 122–123)). These structures are handled as adjunct extraction in HPSG (Pollard and Sag 1994, pp. 384–388), which allow interaction between extracted adjuncts (*e.g.*, prepositional phrases and adverbs) and their modifiees through the SLASH feature of the modifiees.

As exemplified by these cases, the treatment of some linguistic phenomena in the obtained

HPSG-style grammar is analogous to the one employed in the HPSG formalism, especially in the way to specify a syntactic structure taken by a node subcategorized by a trunk node (or a head in HPSG). However, these grammars differ in the treatment of some type of unbounded dependency, which conveys dependency beyond the the locality normally assumed in LTAG. Further elaboration would be necessary for the conversion of elementary trees that include such convoluted unbounded dependencies.

3.2 Experiments on Parsing Comparison

In this section, we perform a comparison between LTAG and HPSG parsers based on generic parsing techniques. As described in Introduction, we focus on two generic parsing techniques, namely, dynamic programming (Sarkar 2000; Haas 1987) and CFG filtering (Harbusch 1990; Poller 1994; Torisawa and Tsujii 1996; Poller and Becker 1998; Torisawa et al. 2000; Kiefer and Krieger 2000).

In what follows, we introduce dynamic programming and CFG filtering in Sections 3.2.1 and 3.2.2, respectively, and also review how these techniques have been employed in parsers for LTAG and HPSG. We then compare LTAG and HPSG parsers in Section 3.2.3 and 3.2.4, respectively. Since dynamic programming forms the basis of most contemporary parsing techniques, a comparison of parsers using it allows us to roughly grasp the difference between the performance of LTAG and HPSG parsers. Since the impact of CFG filtering for LTAG is quite different from that for HPSG, CFG filtering can be a good material for demonstrating our methodology towards improving generic parsing techniques through parsing comparison. After showing each set of experiments, we also identify the algorithmic difference between the LTAG and HPSG parsers that causes performance difference, and then suggest the way of improving generic parsing techniques.

3.2.1 Dynamic Programming Techniques

The LTAG and HPSG parsers with dynamic programming used in our experiments (van Noord 1994; Haas 1987) perform *factoring*, a common-sense parsing technique that avoids generating duplicate equivalent partial parse trees. In the following sections, we briefly describe how factoring is accomplished in parsers for the two formalisms.

Head-corner parser for LTAG

One of the LTAG parsers used in our experiments is a head-corner LTAG parser (Sarkar 2000). Its parsing algorithm is a chart-based variant of van Noord's (van Noord 1994). The parser uses a

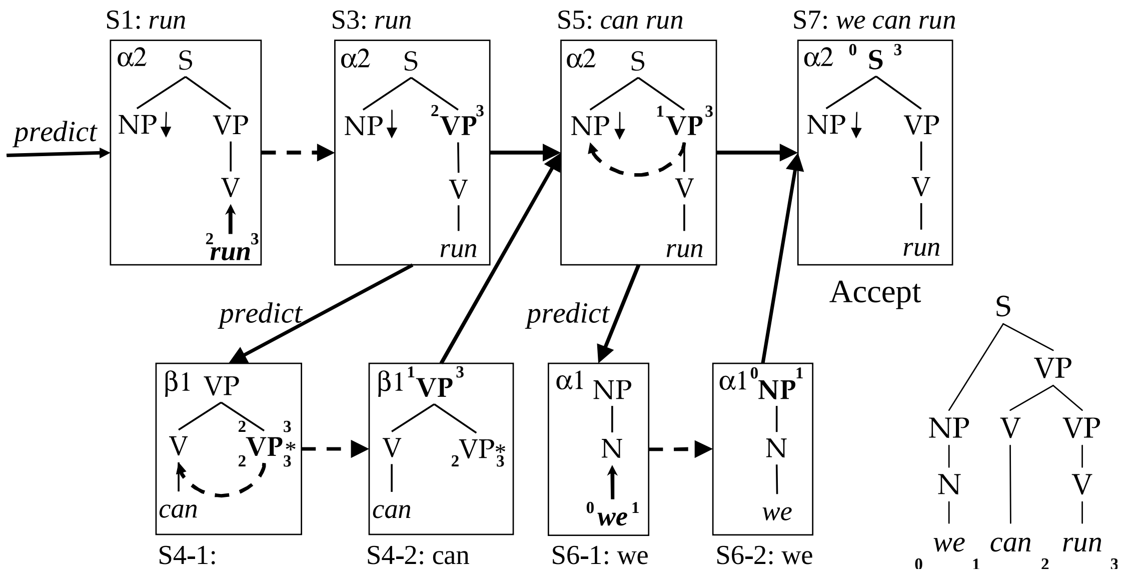


Figure 3.4: Example of head-corner parsing for an LTAG grammar

data structure called *an agenda*. The agenda stores *states* to be processed. A state is represented by a quintuplet consisting of an elementary tree, its root node, its processing node, a span of the foot node over the input, and a span of the tree over the input, which denote the processed and unprocessed parts of the tree.

Figure 3.4 depicts the process of head-corner parsing for the sentence "*we can run*." In the figure, nodes in bold face, arrows in states, arrows between states,[4] and superscripts (subscripts) of processing nodes (foot nodes) respectively indicate processing nodes, directions of processing, relations between the states, and spans of the trees (the foot nodes) over the input string. In head-corner parsing, the parser traverses a tree from one leaf node called *the head-corner* to the root node. This tree traversal is called *head-corner traversal*. During head-corner traversal, the parser recognizes the siblings of the processing node and possible adjunctions at this node. In Figure 3.4, the parser first performs prediction, which pushes into the agenda a new state of an initial tree $\alpha 1$ whose root node matches the symbol S corresponding to the sentence (state S1 in Figure 3.4). The parser proceeds in a bottom-up manner from the head-corner "*run*" to S by pushing into the agenda new states which are generated by moving the processing nodes of states in the agenda, until the

[4]Dotted arrows between states imply that there is at least one intermediate states between them.

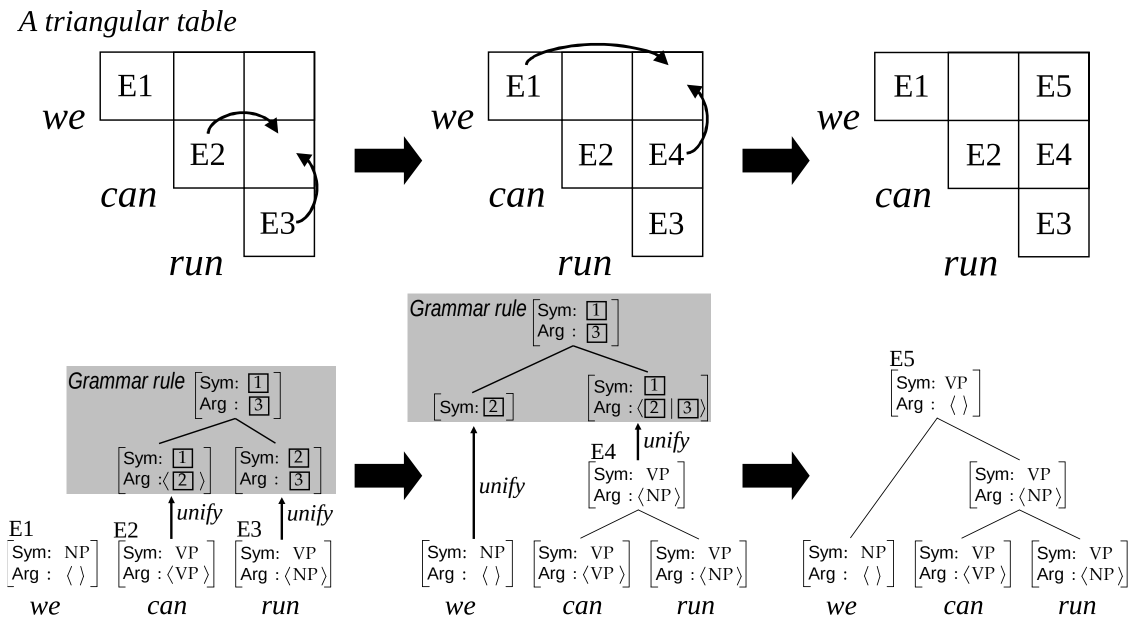


Figure 3.5: Example of CKY-style parsing for an HPSG grammar

processing nodes reach to the root node and recognizes the elementary tree. When moving up to the node VP in $\alpha 2$ (S3), the parser executes adjunction at the processing node VP and predicts a new state S4-1 for the adjoining tree $\beta 1$. After recognizing $\beta 1$ (S4-2), the parser tries to execute substitution at the sibling of VP (S5). To cause substitution at the sibling NP, the parser predicts a new state S6-1 for $\alpha 1$. Then, the parser proceeds to the root S of $\alpha 2$ (S8). Since there is no state to be processed in the agenda, parsing of "*we can run*" ends.

The parser performs factoring when it generates a new state. That is, it pushes a state in the agenda only when an equivalent state does not exist in the agenda. Note that equivalent states are those which have the same quintuplet.

CKY-style HPSG parser

One of the HPSG parsers used in our experiments is a CKY-style HPSG parser (Haas 1987). The parser uses a data structure called *a triangular table*. The triangular table stores *edges*, which correspond to partial parse trees. An edge is described with a tuple consisting of a feature structure that represents the root node of a partial parse tree and a span over the input.

Figure 3.5 illustrates an example of the CKY-style parsing for an HPSG grammar. First, lexical entries for "*we*," "*can*," and "*run*" are stored as edges E1, E2, and E3 in the triangular table. Next, E2 and E3 are each unified with the daughter feature structures of an ID grammar rule. The feature structure of the mother node is determined as a result of this unification and is stored in the triangular table as a new edge E4. An ID grammar rule is then applied to E1 and E4, and a new edge E5 is generated. Since the parse tree spans the whole input string, parsing of "*we can run*" ends.

The parser performs factoring when it generates a new edge. That is, it stores an edge in the triangular table unless an equivalent edge exists in the cell. Note that equivalent edges are those which have the same tuple.

3.2.2 CFG Filtering Techniques

CFG filtering is a parsing scheme that predicts possible parse trees by using a CFG extracted from a given grammar. An initial offline step of CFG filtering is performed to *approximate* a given grammar with a CFG, in other words, to extract a CFG backbone from a given grammar (*Context-Free (CF) approximation*). The resulting CFG is used as an efficient device for computing the necessary conditions for parse trees.

After the initial step, CFG filtering generally comprises two phases. In phase 1, the parser first constructs possible parse trees by using the CFG obtained in the initial off-line step, and then filters out CFG edges unreachable by top-down traversal starting from roots of successful context-free derivations. In phase 2, the parser applies full constraints in the given grammar to the remaining CFG edges, and eliminates parse trees that are not licensed by those constraints. The number of the remaining CFG edges therefore indicates the degree of approximation, and we call those edges *essential edges*.

The parsers with CFG filtering for LTAG and HPSG follow the above parsing strategy, but differ in their ways of approximating a grammar with a CFG and eliminating impossible parse trees in phase 2. In the following sections, we briefly describe the CF approximation and the elimination of impossible parse trees for each formalism.

CF approximation of LTAG

CFG filtering techniques for LTAG were first proposed by Harbusch (Harbusch 1990) and subsequently studied by Poller (Poller 1994) and Poller and Becker (Poller and Becker 1998). In the CFG filtering techniques for LTAG (Harbusch 1990; Poller and Becker 1998), every branching of elementary trees in a given grammar is extracted as a CFG rule as shown in Figure 3.6. The

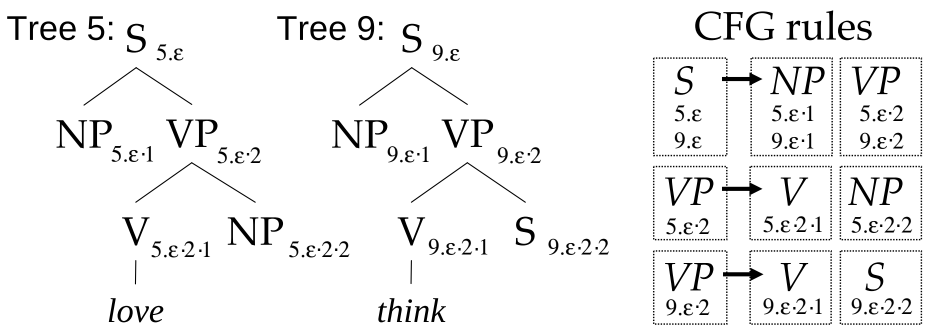


Figure 3.6: Extraction of CFG from LTAG

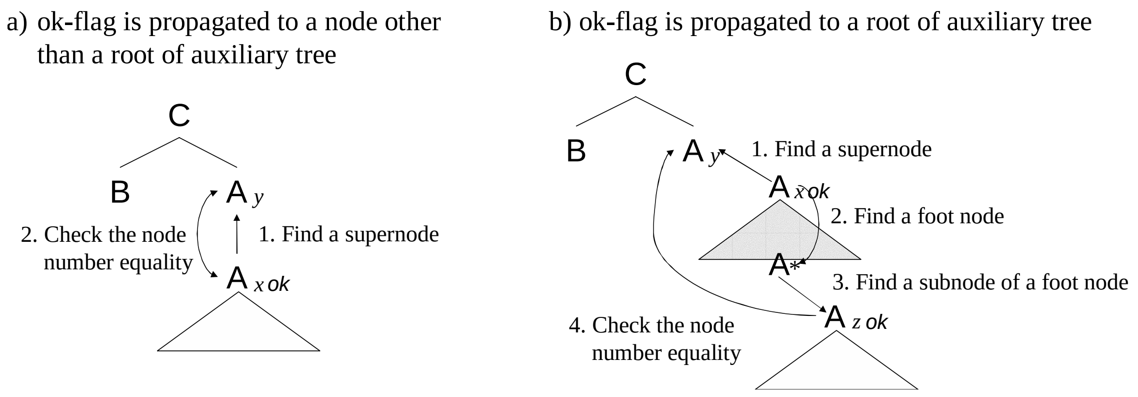

a) ok-flag is propagated to a node other than a root of auxiliary tree
C
B
A y
A x ok
1. Find a supernode
2. Check the node number equality
b) ok-flag is propagated to a root of auxiliary tree
C
B
A y
A x ok
A*
A z ok
1. Find a supernode
2. Find a foot node
3. Find a subnode of a foot node
4. Check the node number equality

Figure 3.7: Ok-propagation from an essential edge to another

nonterminal symbol of the left-hand side of each CFG rule is associated with *a node number* that records a unique node address (a subscript attached to each node in Figure 3.6). In the phase 1, they perform CFG parsing using CFG rules without the node numbers.

Because the obtained CFG can reflect only local constraints given in each local structure of the elementary trees (e.g., a verb (V) and a noun phrase (NP) constitute a verb phrase (VP) in the lower branching of the tree 5 as in Figure 3.6), it generates invalid parse trees that connect local trees extracted from different elementary trees. To eliminate such illegal parse trees, a link between branchings is preserved as a node number. As depicted in Figure 3.7, we can eliminate such parse trees by traversing essential edges in a bottom-up manner and recursively propagating *an ok-flag* from node number x to node number y when a connection between x and y is allowed in the LTAG grammar. We call this propagation *ok-propagation* (Poller 1994). The left-hand side of Figure 3.7

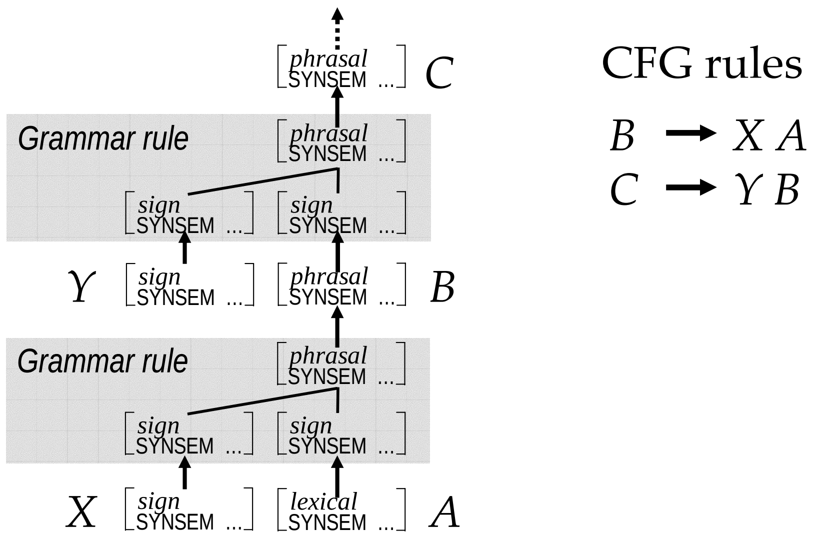


Figure 3.8: Extraction of CFG from HPSG

includes ok-propagation between the split internal nodes in the same elementary tree, and from a root node of an elementary tree to a substitution node of another tree. Since these constructions concern only a local structure of a tree, we simply examine the equality between the node numbers. The right-hand side of Figure 3.7 corresponds to the recognition of adjunction. When the ok-flag is propagated to a root of an auxiliary tree, we first find the foot node of the auxiliary tree, and next identify an internal node of another tree that takes adjunction. We then examine the equality between the node numbers of the adjoined (and split) internal nodes.

CF approximation of HPSG

CFG filtering techniques for HPSG were proposed by Torisawa and Tsujii (Torisawa and Tsujii 1996) and subsequently studied by Kiefer and Krieger (Kiefer and Krieger 2000). In the CFG filtering techniques for HPSG (Torisawa and Tsujii 1996; Torisawa et al. 2000; Kiefer and Krieger 2000), a CFG is extracted from a given HPSG grammar by recursively instantiating daughters of a grammar rule with lexical entries and generated feature structures, as shown in Figure 3.8. This procedure stops when new feature structures are not generated. We must impose restrictions (Shieber 1985) on the features (i.e., ignore them) or on the number of rule instantiations or both in order to guarantee termination of the rule instantiation. A CFG is obtained by regarding the initial and the generated feature structures as nonterminals and bottom-up derivation relationships as CFG rules.

Table 3.4: Parsing performance with the XTAG English grammar for the ATIS corpus

Parser	**Parse time (sec.)**
Naive	1.54
lem	20.76

The resulting CFG reflects the local and global constraints of the whole structure in lexical entries, because generated HPSG signs by rule applications preserve the whole constraints that are necessarily to construct syntactic structured determined by the constraints (recall the internal node in the Figure 1.9). However, it still generates invalid parse trees that do not reflect the constraints given by the features that were ignored in the CFG. These parse trees are eliminated in phase 2 by applying HPSG grammar rules that correspond to the applied CFG rules. We call this rule application *rule-application*.

3.2.3 Comparison of Dynamic Programming Techniques

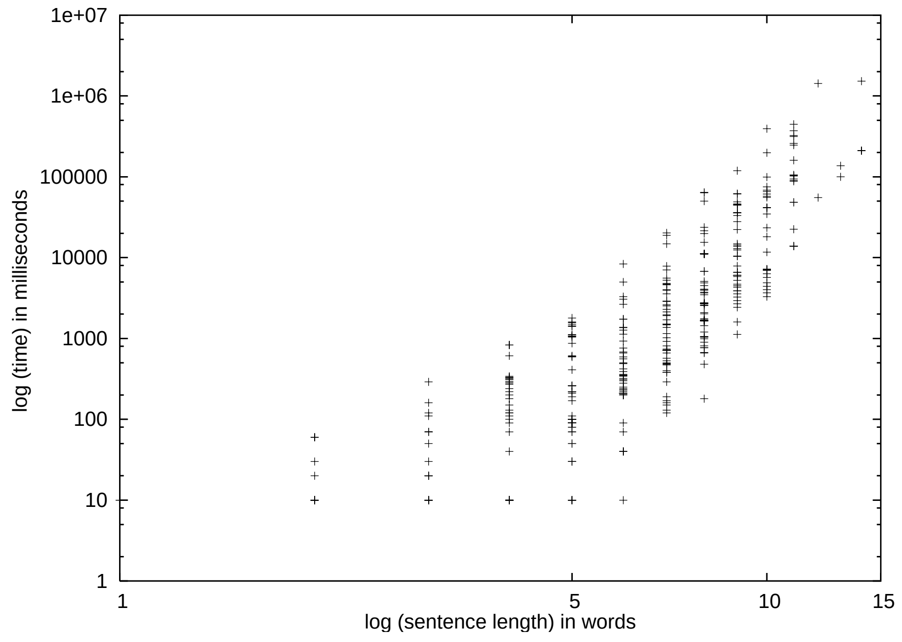
We compared a pair of dynamic programming techniques for LTAG (van Noord 1994) and HPSG (Haas 1987) described in Section 3.2.1. Henceforth, *lem* refers to the LTAG parser (Sarkar 2000), ANSI C implementation of the head-corner parsing.[5] *Naive* refers to C++ implementation of the CKY-style HPSG parser.

Table 3.4 shows the parsing speed results for 452 sentences from the ATIS corpus (Marcus et al. 1993)[6] (average sentence length: 6.32 words). The machine used in the following experiments was a 1.26 GHz Pentium III with 4 GB memory. The results showed that the HPSG parser achieved a speed-up of a factor of 13 in terms of the average parse time. Figure 3.9 shows parse time plotted against sentence length, where both axes use logarithmic scales. Since the increase in parse time versus sentence length plotted on logarithmic scales is equal to the degree of polynomial order of the empirical time complexity, the graphs show that the order of the empirical time complexity of *lem* is higher than that of *Naive*.

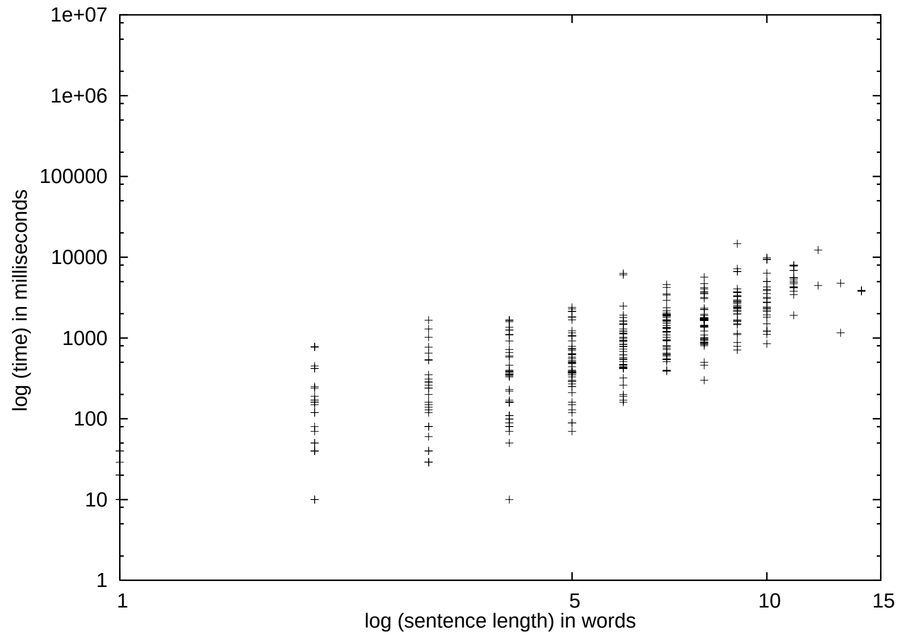
As noted in Section 3.2.1, both parsers have an architecture that supports factoring, but the ways in which they perform factoring differ. Remember that a state in the LTAG parser is a quintuplet consisting of an elementary tree, a root node, a processing node, a span of the foot node over an

[5]The LTAG parser is available at: `ftp://ftp.cis.upenn.edu/pub/xtag/lem/lem-0.14.0.tgz`

[6]We eliminated 56 sentences because of parser time-outs, and 69 sentences because the LTAG parser had bugs in its preprocessor preventing it from producing correct derivation trees.



lem



Naive

Figure 3.9: Parsing performance with the XTAG English grammar for the ATIS corpus

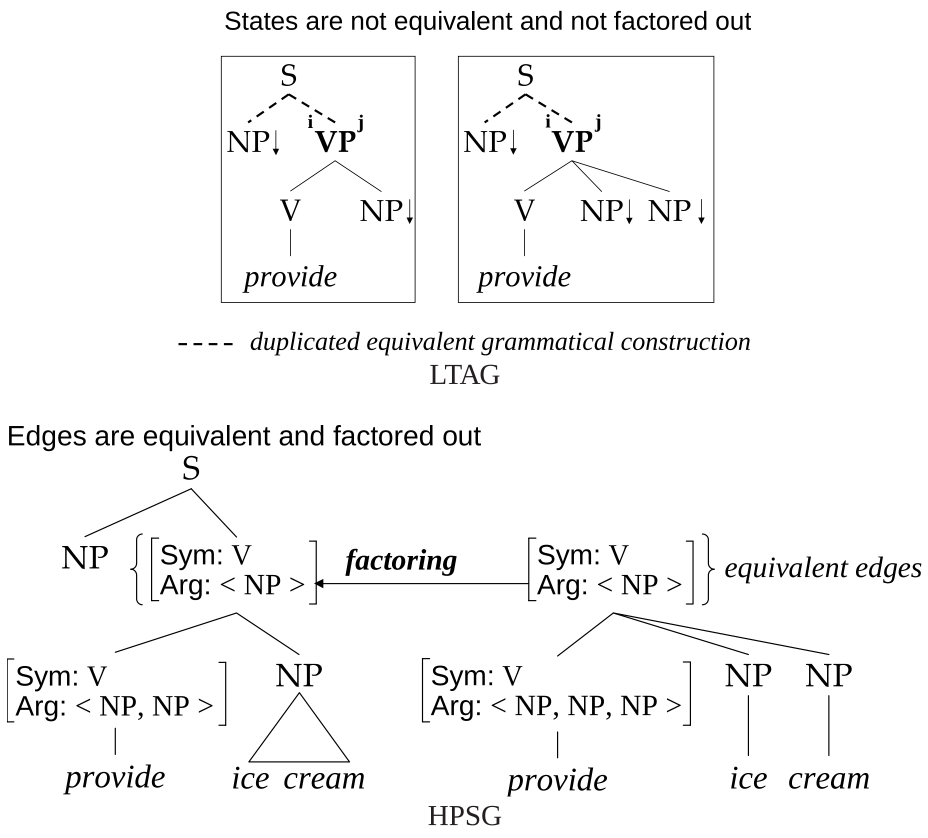


Figure 3.10: Difference between factoring schemes in LTAG and HPSG: ambiguity between NP and NP-NP constructions

input, and a span of the tree over an input, while an edge in the HPSG parser is a tuple consisting of a feature structure and a span over an input. By considering how the HPSG parser handles the HPSG-style grammar converted from the LTAG, we see that the HPSG parser performs factoring of edges when the edges' feature structures are equivalent (the right-hand side of Figure 3.10 and 3.11); this means the factoring can be performed every internal node.[7] On the other hand, the LTAG parser performs factoring of states only when the elementary trees of the states have root nodes labeled the same nonterminal (Figure 3.10 and 3.11).[8] Since the root node corresponds to a feature structure

[7]The author wishes to thank Mika Tarukawa for suggesting this example.

[8]Strictly speaking, the factoring is performed when the states with equivalent root nodes substitute or adjoin to another tree x. When the tree x take substitution or adjunction of the elementary trees, the generated states are equivalent because

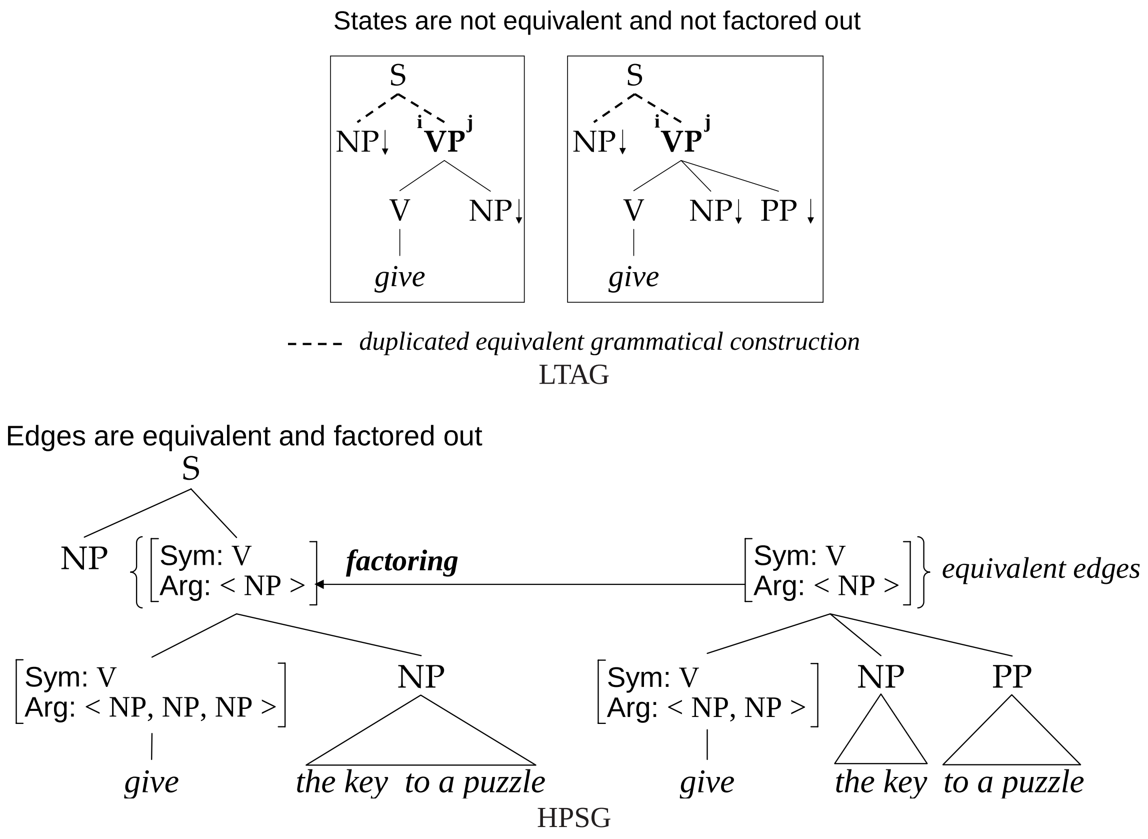
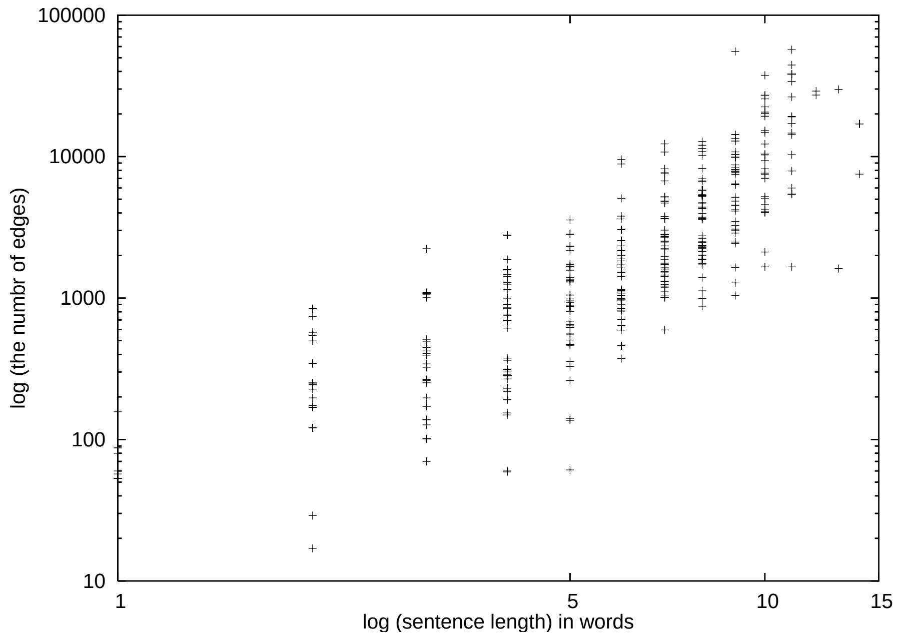


Figure 3.11: Difference between factoring schemes in LTAG and HPSG: ambiguity between NP and NP-PP constructions

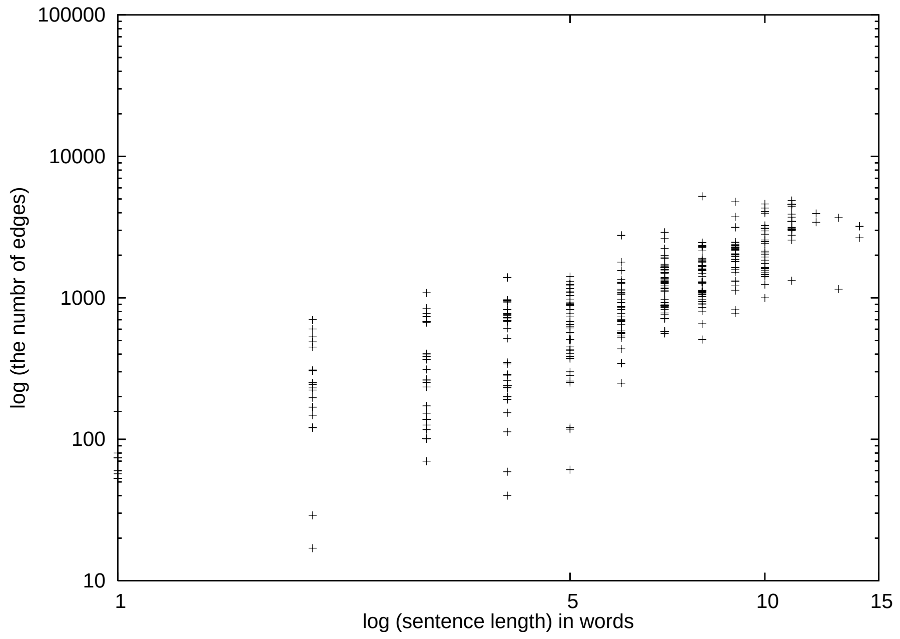
whose Arg feature is an empty stack in HPSG, this difference means that the HPSG parser factors out more partial parse trees than does the LTAG parser. As illustrated in Figure 3.10 and 3.11, the LTAG parser cannot avoid duplicating equivalent grammatical constructions corresponding to fragments of elementary trees. It is also noteworthy that some of these ambiguities in elementary trees are closely related to PP attachment ambiguity, which are expected to appear quite frequently in real-world sentences. We hypothesize that this difference in factoring schemes leads to the difference in the empirical time complexity.

To verify the above argument, we conducted a parsing experiment on the same corpus by using

they do include no information on the substituted or adjoined trees.



Naive$_{rf}$



Naive

Figure 3.12: Number of edges of a variant of *Naive* (*Naive*$_{rf}$) which performs factoring only when the factoring can be performed in *lem* (above) and *Naive* (below)

a variant of *Naive* (hereafter *Naive*$_{rf}$) which performs factoring only when the factoring could be performed by the LTAG parser.[9] Since edges and states generated by the LTAG and HPSG parsers represent partial parse trees, their numbers are reliable indicators of empirical time complexity. Figure 3.12 shows the number of edges plotted against sentence length, where both axes use logarithmic scales.[10] The increase in the number of edges of *Naive*$_{rf}$ was higher than that of *Naive*. Since *Naive*$_{rf}$ mimics the factoring scheme of *lem*, the difference in parse time between *lem* and *Naive* and the difference in the number of edges between *Naive*$_{rf}$ and *Naive* confirm that the difference in the factoring scheme is the major cause of difference in the empirical time complexity.[11]

At first glance, these results are inconsistent with the fact that the theoretical bound of worst time complexity for HPSG parsing is exponential, while LTAG parsing requires $\mathcal{O}(n^6)$ for an input of length n. However, Carroll (Carroll 1994) demonstrated that theoretical bounds of time complexity with respect to grammar size and input length have little impact on performance for some unification-based parsing algorithms, and attributed the reason to the specification of grammars (i.e., variations in grammar rules, etc.). Sarkar et al. (Sarkar et al. 2000) studied LTAG grammars extracted from the Penn Treebank and reported that the theoretical bound of computational complexity does not significantly affect parsing performance and that the most dominant factor is syntactic lexical ambiguity, i.e., ambiguity of lexical entries for the same words. Our results are therefore convincing because factoring handles ambiguity in partial parse trees, which is mostly caused by the syntactic lexical ambiguity.

It should be noted that the above investigation also suggests another way of factoring in LTAG. We can merge two states which have equivalent unprocessed parts, as depicted in Figure 3.10, into a single state when they cover the same span of input. This kind of factoring that merges edges with equivalent unprocessed parts has been proposed for CFG by Leermakers (Leermakers 1992). In his parser, the edges in Earley parsing are merged if their rules have a common unprocessed suffix. As exemplified by the application of *Naive* to an HPSG-style grammar converted from LTAG, this kind of factoring is applicable to LTAG parsing. Our study empirically attested to its effectiveness in LTAG parsing, not by implementing complex parsing algorithms, but by simply applying the HPSG parser potentially equipped with such functionality to the grammar converted from LTAG.

[9] We did not compare the states of *lem* with the edges of *Naive* because an edge of *Naive* does not have a one-to-one correspondence with a state of *lem*.

[10] *Naive* and *Naive*$_{t}extitrf$ in fact duplicate parts of auxiliary trees when they adjoin to different trees. Although *lem* can avoid this duplication, it has no serious effect on our conclusion.

[11] *lem* and *Naive* also differ in their ways of handling linguistic features. However, we suppose that their impact on parsing performance is mild compared to one by the difference in factoring.

Table 3.5: Size of extracted LTAGs and CFGs approximated from them (above: the number of nonterminals; below: the number of rules)

Grammar	**G**$_2$	**G**$_{2\text{-}4}$	**G**$_{2\text{-}6}$	**G**$_{2\text{-}8}$	**G**$_{2\text{-}10}$	**G**$_{2\text{-}21}$	**G**$_{xtag}$
LTAG	65	66	66	66	67	67	21
	1,488	2,412	3,139	3,536	3,999	6,085	1,226
CFG$_{PB}$	65	66	66	66	67	67	21
	716	954	1,090	1,158	1,229	1,552	202
CFG$_{TNT}$	1,989	3,118	4,009	4,468	5,034	7,454	9,034
	18,323	35,541	50,115	58,356	68,239	118,464	85,454

3.2.4 Comparison of CFG filtering techniques

Following on from the comparison of dynamic programming techniques, we compared a pair of CFG filtering techniques for LTAG (Harbusch 1990; Poller and Becker 1998) and HPSG (Torisawa et al. 2000; Kiefer and Krieger 2000) described in Section 3.2.2. We chose the filtering technique of Poller and Becker (Poller and Becker 1998) because it is the most sophisticated algorithm for CFG filtering for LTAG. Hereafter, we refer to its C++ implementation as *PB*. The other filtering technique for HPSG was *TNT* (Torisawa et al. 2000). We modified the CF approximation of the original *TNT* by instantiating both daughters and restricting the number of rule instantiations, as shown in (Kiefer and Krieger 2000), to approximate the obtained HPSG-style grammar with a CFG. In phase 1, *PB* and *TNT* performed Earley (Earley 1970) and CKY (Younger 1967) parsing, respectively. Note that the CFG filtering techniques for LTAG used the same CF approximation, as did the CFG filtering techniques for HPSG, as described in Sections 3.2.2. Comparison of *PB* and *TNT* thus suffices to investigate the effect of the CF approximations for LTAG and HPSG.

We used the XTAG English grammar without linguistic features and LTAGs extracted by the method proposed in (Miyao et al. 2003) from Sections 2-21 of the Wall Street Journal (WSJ) portion in the Penn Treebank (Marcus et al. 1993) and their subsets.[12] We converted them into strongly equivalent HPSG-style grammars by using the grammar conversion described in Section 2.1.1. Table 3.5 shows the size of CFGs approximated from the strongly equivalent grammars. G$_{xtag}$, G$_x$, CFG$_{PB}$, and CFG$_{TNT}$ henceforth refer to the XTAG English grammar, the LTAG extracted from Section x of WSJ, the CFGs approximated from G$_{xtag}$ and G$_x$ by *PB* and *TNT*, respectively.

[12]The elementary trees in the LTAGs are binarized.

Table 3.6: Parsing performance (sec.) for Section 2 of WSJ

Parser	$\mathbf{G}_2$	$\mathbf{G}_{2\text{-}4}$	$\mathbf{G}_{2\text{-}6}$	$\mathbf{G}_{2\text{-}8}$	$\mathbf{G}_{2\text{-}10}$	$\mathbf{G}_{2\text{-}21}$	$\mathbf{G}_{xtag}$
PB	1.4	9.1	17.4	24.0	34.2	124.3	15.3
TNT	0.044	0.097	0.144	0.182	0.224	0.542	0.606
ratio (PB/TNT)	31.8	93.8	120.8	131.9	152.7	229.3	25.2

Table 3.7: Number of essential edges generated in parsing of Section 02 of WSJ

Parser	$\mathbf{G}_2$	$\mathbf{G}_{2\text{-}4}$	$\mathbf{G}_{2\text{-}6}$	$\mathbf{G}_{2\text{-}8}$	$\mathbf{G}_{2\text{-}10}$	$\mathbf{G}_{2\text{-}21}$	$\mathbf{G}_{xtag}$
PB	791	1,435	1,924	2,192	2,566	3,976	871
TNT	63	121	174	218	265	536	548
ratio (PB/TNT)	12.6	11.9	11.1	10.1	9.7	7.4	1.6

CFG$_{TNT}$ is much larger than CFG$_{PB}$. By investigating parsing performance using these CFGs as filters, we conclude that larger size of CFG$_{TNT}$ resulted in the better parsing performance.

Table 3.6 shows the parse time for 254 sentences of length $n\,(n \leq 10)$ from Section 2 of WSJ (average sentence length: 6.72 words).[13] This result shows not only that *TNT* was much faster than *PB*, but also that the performance difference between them increased when the larger grammars were used.

To estimate the degree of CF approximation, we measured the number of essential (inactive) edges of phase 1. Table 3.7 shows the number of essential edges. *PB* produces a much greater number of essential edges than *TNT*. We then investigated the effect of different numbers of essential edges on phase 2. Table 3.8 shows the success rates of ok-propagation and rule-application. The success rate of rule-application (for *TNT*) is 100% for LTAGs extracted from WSJ and relatively high for the XTAG English grammar, while that of ok-propagation (for *PB*) is quite low.[14] These results indicate that CFG$_{TNT}$ is superior to CFG$_{PB}$ with respect to the degree of CF approximation.

[13]We used a subset of the training corpus to avoid using default lexical entries for unknown words, because there are various ways to assign default entries for automatically extracted grammars and this would have an uncontrolled effect on parsing performance.

[14]This means that the extracted LTAGs must be in a subclass of LTAG which is compatible with CFG and were completely approximated with CFGs. Note that *TNT* does not guarantee the success ratio of 100% for the phase 2 operations.

Table 3.8: Success rate (%) of phase 2 operations

Operations	**G**$_2$	**G**$_{2\text{-}4}$	**G**$_{2\text{-}6}$	**G**$_{2\text{-}8}$	**G**$_{2\text{-}10}$	**G**$_{2\text{-}21}$	**G**$_{xtag}$
ok-propagation (*PB*)	38.5	34.3	33.1	32.3	31.7	31.0	30.6
rule-application (*TNT*)	100	100	100	100	100	100	71.2

It is noteworthy in Table 3.6 that the performance difference between *PB* and *TNT* with the XTAG English grammar G$_{xtag}$ is smaller than the performance difference between the parsers with automatically extracted grammars G$_x$. In order to approximate the XTAG English grammar, we have not only restricted the number of rule instantiations to 5 but also have ignored the identifier which we introduced in 2.1.2 to convert non-canonical elementary trees into HPSG lexical entries. Due to these restrictions, the degree of approximation of G_{xtag} by *TNT* is worse than the degree of approximation of G_x by *TNT*. Another factor would be the specifications of the original grammar. As shown in Table 3.5, the variation of the branching structures in the elementary tree templates of G$_{xtag}$ (equal to the number of grammar rules in CFG$_{PB}$) is smaller than that of G$_x$. This means that G$_{xtag}$ contains a number of structural ambiguities in the elementary trees, which are efficiently packed in the CFG approximated by *PB*.

We can work out the reason for the performance difference between *PB* and *TNT* by investigating how the CF approximation of HPSG approximates HPSG-style grammars converted from LTAGs. As described in Section 2.1.1, the grammar conversion preserves the whole structure of each elementary tree (precisely, a canonical elementary tree) in a stack, and grammar rules manipulate the top element of the stack. A generated feature structure in the approximation process thus corresponds to the whole unprocessed parts of a canonical elementary tree, as shown in Figure 3.13. This implies that successful context-free derivations obtained by CFG$_{TNT}$ basically involve elementary trees in which all substitution and adjunction have succeeded. However, as mentioned in Section 3.2.2, CFG$_{PB}$ (as well as a CFG produced by another work (Harbusch 1990)) cannot avoid generating invalid parse trees that connect two local structures where adjunction takes place between them. We used G$_{2\text{-}21}$ to calculate the percentage of ok-propagations that were between two node numbers that take adjunction (the right-hand side of Figure 3.7) and the success rate for this percentage. 87% of the total number of trials of ok-propagations was of this type but their success rate was only 22%. These results suggest that the global constraints in a given grammar are essential to obtaining an effective CFG filter. This would decrease the number of essential edges generated by *PB*.

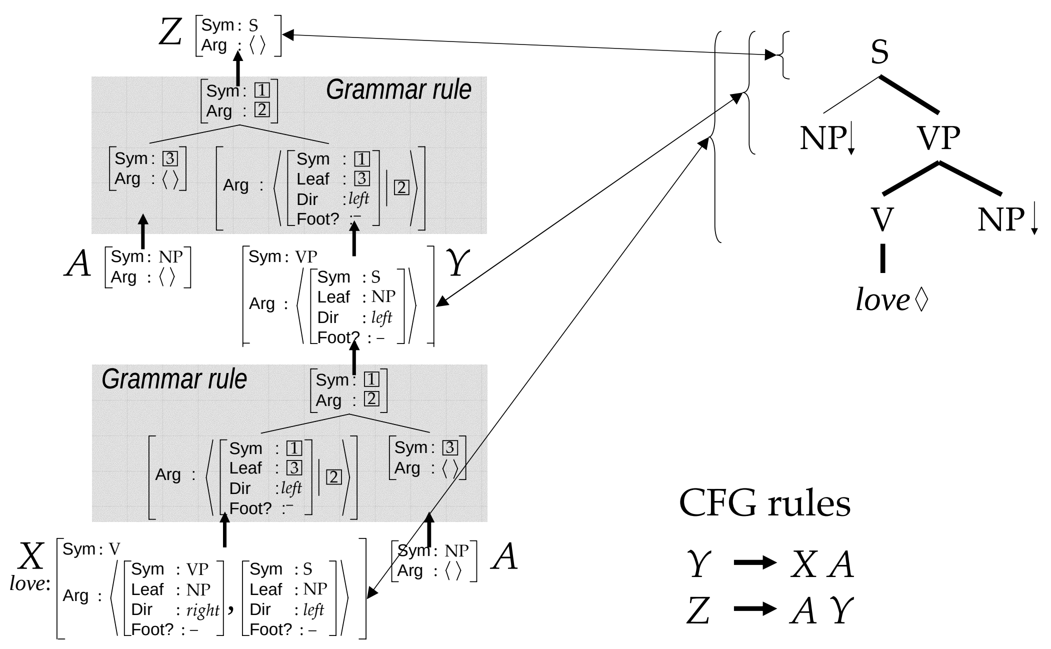

Z
Sym : S
Arg : ⟨ ⟩
Grammar rule
Sym : 1
Arg : 2
Sym : 3
Arg : ⟨ ⟩
Arg : ⟨ Sym : 1, Leaf : 3, Dir : left, Foot? : – | 2 ⟩
A
Sym : NP
Arg : ⟨ ⟩
Sym : VP
Arg : ⟨ Sym : S, Leaf : NP, Dir : left, Foot? : – ⟩
Y
Grammar rule
X
love:
Sym : V
Arg : ⟨ Sym : VP, Leaf : NP, Dir : right, Foot? : –, Sym : S, Leaf : NP, Dir : left, Foot? : – ⟩
S
NP↓
VP
V
NP↓
love◊
CFG rules
Y → X A
Z → A Y

Figure 3.13: CF approximation of an HPSG-style grammar converted from LTAG

It should be noted that the above investigation also suggests another way of making a CF approximation for LTAG. We first define a unique mode of tree traversal, such as head-corner traversal (van Noord 1994) described in Section 3.2.1, on which we can sequentially apply substitution and adjunction. We then recursively apply substitution and adjunction on that traversal to elementary trees and generated tree structures. Because the processed parts of the generated tree structures are not used again, we regard the unprocessed parts of the tree structures as nonterminals of a CFG. We can thereby perform another type of CFG filtering for LTAG by combining this CFG filter with a head-corner LTAG parsing algorithm (van Noord 1994) that uses the same tree traversal.

3.3 Chapter Summary

We illustrated the utility of our grammar conversion to bridge the LTAG and HPSG formalisms. We first showed that we can obtain a large-scale HPSG-style grammar that is compatible with HPSG systems, by converting from a large-scale LTAG grammar. The XTAG English grammar, a large-scale LTAG grammar, was successfully converted into HPSG-style grammar. We next investigated the specification of the obtained HPSG-style grammar, and then discussed different ways of encodings of grammatical constructions in the both framework by comparing accounts of several linguistic phenomena by the obtained HPSG-style grammar and HPSG. We then empirically compared two pairs of LTAG and HPSG parsers based on dynamic programming and CFG filtering. Experiments comparing parsers using dynamic programming showed that the different implementations of the factoring scheme caused a difference in the empirical time complexity of the parsers. This result suggests that for LTAG parsing we can achieve a drastic speed-up by merging two states whose elementary trees have the same unprocessed parts. Another experiment comparing parsers with CFG filtering showed that the CF approximation of HPSG produced a more effective filter than that of LTAG. This result also suggests that we can obtain an effective CFG filter for LTAG by approximating the LTAG with a CFG by applying substitution and adjunction along tree traversal and regarding unprocessed parts of generated tree structures as nonterminals of the CFG.

Chapter 4

Related Work to Collaboration among Lexicalized Grammar Formalisms

In this chapter, we discuss the work related to collaboration among lexicalized grammar formalisms. Section 4.1 mentions grammar conversion between LTAG and other grammar formalisms, especially HPSG and those of mildly context-sensitive grammar formalisms, along with a comparison of their results with our research. Section 4.2 describes studies on parsing algorithms of lexicalized grammars that are related to the two parsing techniques focused in our parsing comparison. Section 4.3 discusses work on comparison between LTAG and HPSG parsers, and compares their approaches with ours. Section 4.4 introduces further collaboration between the LTAG and HPSG formalisms that exploits results of our grammar conversion.

4.1 Grammar Conversions between LTAG and Other Formalisms

Study on conversion among LTAG and other formalisms There are several studies on the grammar conversion among different grammar formalisms including LTAG. Weir (Weir 1988), Joshi et al. (Joshi et al. 1991), and Vijay-Shanker and Weir (Vijay-Shanker and Weir 1994) investigated the relation among Tree Adjoining Grammar (TAG) (Joshi et al. 1975), Head Grammar (HG) (Pollard 1984), Combinatory Categorial Grammar (CCG) (Steedman 1986; Steedman 1985), and Linear Indexed Grammar (LIG) (Gazdar 1988), which are in the class called *mildly context-sensitive formalism*. In their research, these four grammar formalisms have been proven to be weakly equivalent with each other in the sense that they generate the same set of strings. Vijay-Shanker and Weir (Vijay-Shanker and Weir 1994) designed parsing algorithms for them by fully exploiting this nature of weak equivalence. Doran and Srinivas (Doran and Srinivas 2000) converted the XTAG English grammar (The XTAG Research Group 1995) into a CCG. They have succeeded in building a wide-coverage CCG with a relatively small workload. Since these studies made use of weak equivalence between the grammar theories towards the collaboration among the four grammar formalisms, further collaboration can be possible when we can obtain strongly equivalent grammars as we show in this dissertation.

Manual translation from the XTAG English grammar to HPSG Tateisi et al. (Tateisi et al. 1998) have manually translated a subset of the XTAG English grammar (The XTAG Research Group 1995), a large-scale English LTAG grammar, into an HPSG grammar. Their work differs from our work in that their main objective was not to pursue the collaboration between the HPSG and LTAG communities, but to construct a wide-coverage HPSG grammar by translating a large-scale LTAG grammar into the HPSG framework. They manually investigated feature percolation in the XTAG English grammar by assuming one of the pre-defined ID schemata to construct each branching in the elementary trees. They have successfully achieved a large-scale HPSG grammar with a relatively small workload, and also revealed linguistic correspondence between feature percolation in the XTAG English grammar and those in HPSG grammar.

Their method, however, depended on a translator's intuitive analysis of the original grammar, and the translation was thus manual and dependent on the input LTAG, i.e., the XTAG English grammar. The manual translation demanded considerable efforts on the part of the translator, and obscured the existence or non-existence of strong equivalence between the original and obtained grammars. Our method of grammar conversion differs from Tateisi's method in that it is fully automatic, and can apply to any grammars in the LTAG formalism. We believe that these properties are essential to catch up with the continuous update of the LTAG resources. This nature of strong equivalence can much contribute to the LTAG communities, by supplying established HPSG technologies such as grammar debugging environments to them. It is nevertheless noteworthy that when we attempt to acquire an HPSG grammar from an HPSG-style grammar converted from LTAG, we will have to analyze feature percolation in an FB-LTAG grammar in order to modularize principles from lexical entries, as demonstrated by Tateisi et al. through their translation.

Conversion from HPSG to LTAG Kasper et al. (Kasper et al. 1995) proposed a grammar conversion from HPSG to LTAG, which is the direction opposite to our conversion, and a subsequent study (Becker and Lopez 2000) discussed the relation between an original HPSG and the obtained LTAG grammar. Their work aims at clarification of the linguistic relevance between the two grammar theories which have developed in parallel. In their conversion, the head domain of an HPSG lexical entry for each word is determined by applying possible grammar rules to the lexical entry and by consuming the list-value of *a selector feature* (SF), which is basically SUBCAT feature in the HPSG formalism. They then regard the generated tree structure as LTAG elementary trees. The rule applications to an HPSG lexical entry terminate when a feature structure of the root of the generated tree structure satisfies a certain condition. When the feature structure of the root of a generated tree structure has empty SF values, the tree is recognized as an initial tree; or when it shares some non-empty SF value in common with one of the frontier nodes in the tree structure, the tree structure is recognized as an auxiliary tree. This conversion is analogous to part of our conversion from a canonical elementary tree to an HPSG lexical entry. However, given the greater generative power of HPSG, the conversion required that some restrictions be placed on the input HPSG to suppress its generative capacity. Moreover, Becker and Lopez (Becker and Lopez 2000) pointed out that there was overgeneration and undergeneration of the LTAG grammars. Thus the results of conversion do not guarantee strong equivalence. Furthermore, although Becker and Lopez asserted that conversion from HPSG to LTAG is also beneficial to obtain speed efficiency from a theoretical viewpoint, our results in this dissertation have definitively proved that their assertion does not hold from a viewpoint of empirical time complexity; that is, we achieved a drastic speed-up by converting LTAG to HPSG.

The above conversions do not have the advantages we have addressed, which are only attainable when strong equivalence is preserved. We should finally address a study that compile a TAG grammar into a strongly equivalent grammar based on a different grammar formalism.

Conversion from TAG to Range Concatenation Grammar Boullier (Boullier 1998) and his subsequent study (Boullier 1999) have proposed Range Concatenation Grammar (RCG) as an alternative representation framework for TAGs and other grammar theories. They have established an algorithm that converts TAGs into strongly equivalent RCGs. They claimed that they distinguished grammar theories and their representation framework, and proposed a parsing algorithm for RCGs converted from grammars in various linguistic theories (Barthélemy et al. 2001).

In this dissertation, our claim is that the choice of representation framework in each grammar theory is *not* independent from the grammar theories, and thus we do not aim at having a generic representation framework that is suitable for any lexicalized grammar formalisms.[1] We have instead proposed a conversion algorithm between lexicalized grammars in order to clarify

[1]Note that RCG is an abstract representation framework that is even independent from lexicalization approaches in the lexicalized grammar formalisms.

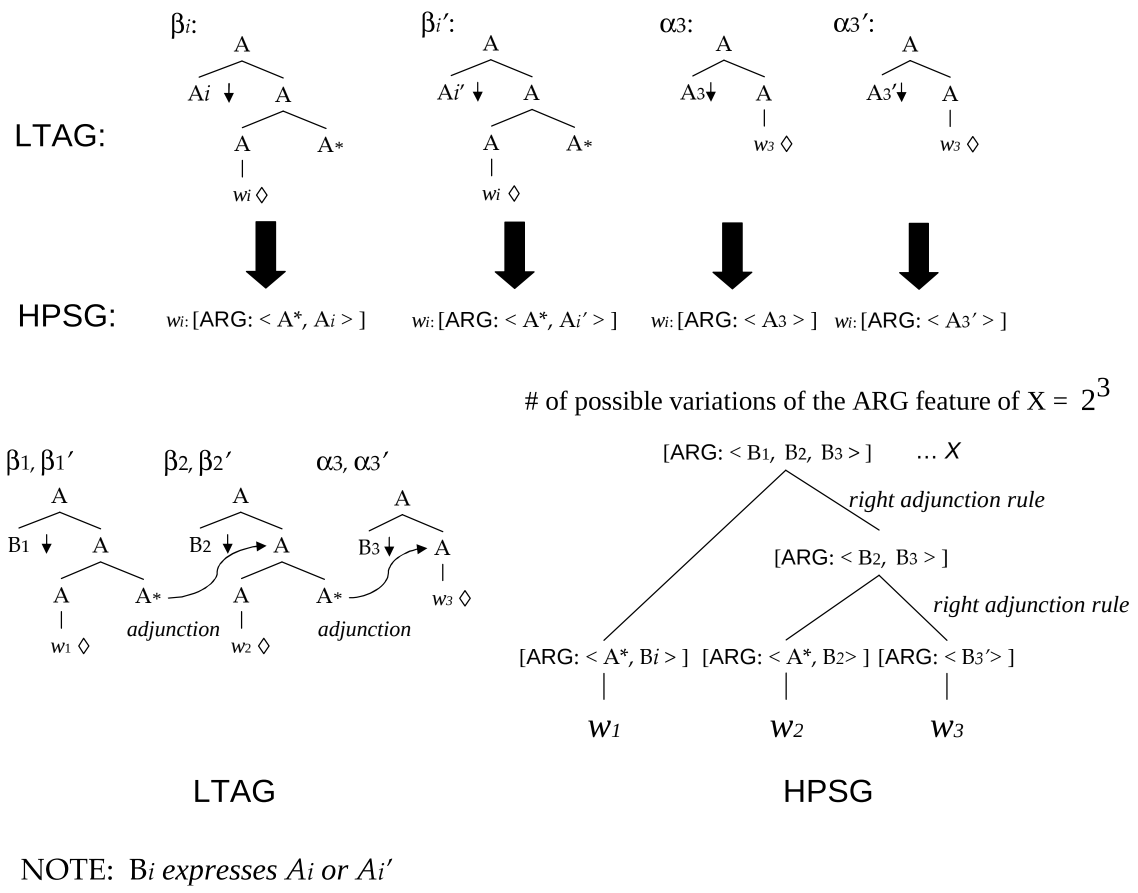


Figure 4.1: Exponential variations in the Arg feature

the differences among processing architectures of the grammar theories. For example, as we have experimented in Section 3.2.3, although the HPSG parser showed the lower empirical time complexity than the LTAG parser, its theoretical bounds of time complexity is beyond polynomial. This is because the number of the variations of the values of the Arg features in edges generated by an HPSG parser could be exponential since they include concatenation of parts of the Arg features that originate from different elementary trees, as shown in a synthetic example in Figure 4.1.[2] On the

[2]This explosion of the variations of the Arg features is due to an auxiliary tree with the foot node of depth $n \geq 2$ (*e.g.* β_i, β_i' in Figure 4.1.). In the LTAG grammars that describe natural language, these auxiliary trees are used to express embedded structure (*e.g.*, $\beta 3$ in Figure 1.4). Because in practice these embedded structures can recursively appear at the limited times (Sarkar et al. 2000), the poor computational treatment of HPSG parsers do not deteriorate the parsing efficiency.

other hand in the LTAG parser, because states do not involve combinations of parts of different elementary trees, the number of states are bounded to $O(n^4)$ for a sentence of length n.[3] Furthermore, as experimented in Section 3.2.4, when we approximate an LTAG grammar with a CFG via the HPSG-style grammar, we needed to ignore the identifier in the HPSG-style grammar, which was introduced as a result of tree division. This turned out to deteriorate the degree of CF approximation in the experiments.[4] Our claim is that parsers for a particular lexicalized grammar formalism can make use not only of parsing techniques that are claimed to be generic within the lexicalized grammar formalisms but also of parsing optimizations that are possible only under that grammar formalism. Thus, instead of developing parsing techniques for a generic representational framework, we assert to share generic parsing techniques within the lexicalized grammar formalisms by a method of grammar conversion, and further to optimize those parsing techniques to be suitable for each formalism in order to make use of individual specifications for the processing architecture of the formalism.

4.2 Previous Studies on Parsing of Lexicalized Grammars

In this section, we mention studies on parsing of lexicalized grammars that are related to dynamic programming and CFG filtering, respectively. These two parsing techniques focus on two major difficulties in parsing of lexicalized grammars. Dynamic programming focuses on duplication of equivalent partial results, which stem from ambiguous partial parse trees in parsing of natural languages. CFG filtering concerns the order to solve grammatical constraints in complex feature-based grammars including lexicalized grammars.

4.2.1 Related Work on Dynamic Programming

There are other studies that aim at avoiding the ambiguity caused by syntactic lexical ambiguity in LTAG. Evans and Weir (Evans and Weir 1998) have asserted that the compaction of substructures in elementary trees has a great impact on parsing performance. In their research, several elementary trees for each word were converted into finite-state automata, and merged into a single finite-state

[3]Let us recall the fact that the states of the LTAG parser was quintuplet consisting of an elementary tree, its root node, its processing node, a span of the foot node over the input, a span of the tree over the input. The last two parameters relate to the order of the number of the states, which could be expressed by a quadruplet whose elements range from 0 to n.

[4]For the test sentences in the experiment, the limited number of rule instantiations did not cause deterioration of the success ratio of *TNT*'s phase 2 operations. Even when we increase the limit from 5 to 10, the ratio was unchanged for the test sentences, which implies that the deterioration of the success ratio was completely due to ignoring the identifier.

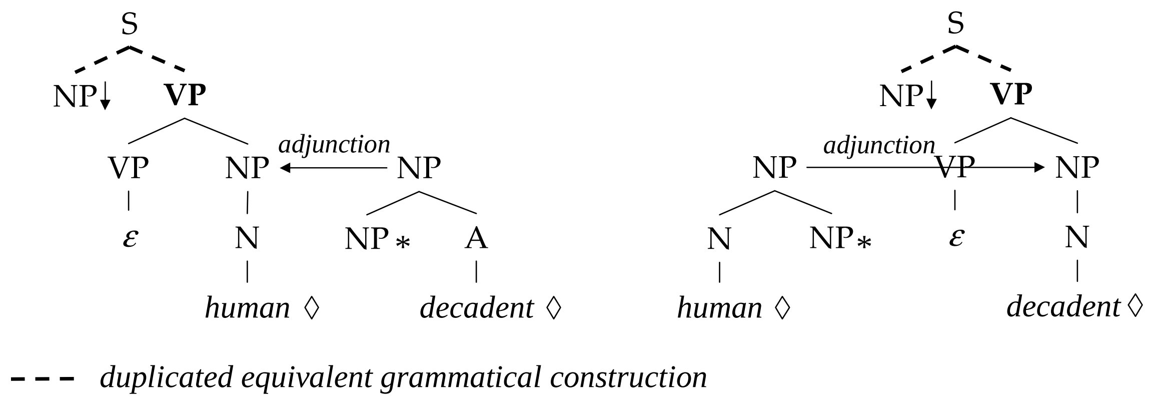


Figure 4.2: An example of syntactic phrasal ambiguity for a phrase "*human decadent*"

automaton. Chen and Vijay-Shanker (Chen and Vijay-Shanker 1997) used underspecified tree descriptions to allow ambiguous node labels in elementary trees. In the HPSG parser employed in our experiments, some of this compaction is dynamically executed by factoring. Shaumyan et al. (Shaumyan et al. 2002) evaluated an automaton-based parser with an LTAG grammar extracted by a method proposed by Xia (Xia 1999), and showed results similar to ours. However, the grammar they used had far less syntactic lexical ambiguity than the XTAG English grammar. Our results with the XTAG English grammar are a strong indication of the importance of compaction of substructures in elementary trees.

We should discuss in detail the differences between the ways to handle structural ambiguities in the above LTAG parsers and the HPSG parser employed in our experiments. The structural ambiguities in parse trees of lexicalized grammars can be classified into the ones which can be packed prior to parsing (syntactic lexical ambiguity) and the ones which can be packed only during parsing (syntactic phrasal ambiguity). The above LTAG parsers handle only the syntactic lexical ambiguity, which appears as structural ambiguities in several elementary trees assigned for each word, while the HPSG parser handle parts of syntactic lexical and phrasal ambiguities. Figure 4.2 shows an example of the syntactic phrasal ambiguity. Although both phrasal structures headed by VP require the same NP subject, their head words are different and thus these ambiguous structures are not factored out even in the LTAG parsers that can handle syntactic lexical ambiguity. Figure 4.3 shows an example of the syntactic lexical ambiguity which the HPSG parser cannot factor out. In the HPSG parser, the edges for the V nodes in Figure 4.3 are not factored out because only parts of the edges are equivalent. In order to handle such partially equivalent edges, modularization (Griffith 1995; Griffith 1996) of feature structures and packing of disjunctive feature structures (Miyao

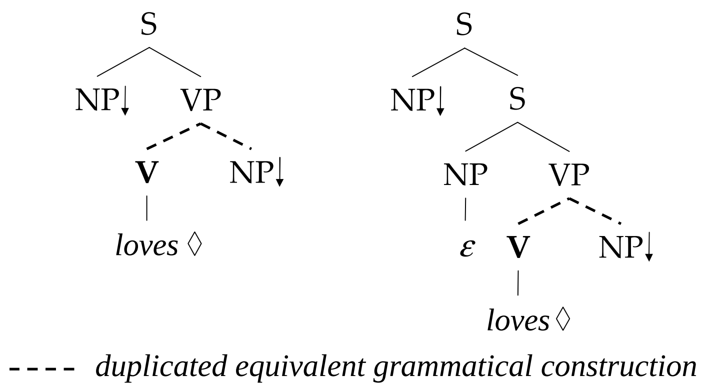


Figure 4.3: An example of syntactic lexical ambiguity which the HPSG parser cannot factor out

1999) have been studied in the unification-based formalisms. The packing of disjunctive feature structures is analogous to automaton-based compaction of lexical entries when it is applied to feature structures of lexical entries, and it further factors out the syntactic phrasal ambiguity when it is applied to feature structures of (partially) equivalent edges. If we investigate the effect of these different factoring schemes on the parsing performance, we can not only identify which type of structural ambiguities is dominant in the lexicalized grammars but also determine the better parsing architecture for the lexicalized grammars. However, it would require exhaustive sets of experiments because structural ambiguities depend entirely on the specification of the grammars; *e.g.*, most of the (syntactic) structural ambiguities can be resolved when semantic constraints are incorporated into grammars.

4.2.2 Related Work on CFG filtering

CFG filtering is first employed for parsing of the Lexical Functional Grammar (LFG) (Kaplan and Bresnan 1982) by Maxwell and Kaplan (Maxwell III and Kaplan 1993). Because an LFG grammar consists of an explicit CFG backbone and functional constraints, they incorporated some functional constraints of a given LFG grammar into the CFG backbone and used the augmented CFG backbone to guide parsing. They achieved the better parsing performance by moving several functional constraints including subcategorization frames of verbs into a CFG backbone. As also demonstrated by Shieber (Shieber 1985), the order to solve grammatical constraints is quite important in parsing of complex feature-based grammar formalisms including lexicalized grammars. In our experiments, we compared two different ways of incorporating constraints of an LTAG grammar into a CFG filter. We demonstrated for both hand-crafted and automatically extracted LTAG grammars that global constraints related to subcategorization frames and non-local dependencies are important to prune ambiguities of partial parse trees.

The effect of two-phased (or guiding) approaches to parsing also depends on the specification of grammars. When intensively restricted grammars such as the LINGO English Resource Grammar (Flickinger 2002) are used, the efficient processing of feature constraints (Nishida et al. 1999; Malouf et al. 2003) are obligatory even when we use CFG filtering (Torisawa et al. 2000; Callmeier 2000).

4.3 Comparison between Parsers for Different Grammar Formalisms

There is only one work on a comparison between LTAG and HPSG parsers. Yoshida et al. (Yoshida et al. 1999) reported a comparison between their LTAG parser and an HPSG parser (Nishida et al. 1999) on the same platform (LiLFeS programming language: (Makino et al. 1998)). They compared the parsing efficiency of two given HPSG (Torisawa and Tsujii 1996) and LTAG (Yoshida et al. 1999) parsers in terms of the parsing time and the number of the states/edges generated in parsing by the parsers.[5] The experimental results showed that the HPSG parser was more efficient than the LTAG parser under their experimental settings.

Although their research showed empirical efficiency of the HPSG parser against the LTAG parser, it was not a fair comparison not only because the LTAG and HPSG parsers they used do not share the same generic parsing techniques but also because the HPSG grammar they used was obtained by the manual translation by Tateisi et al. (Tateisi et al. 1998) and their translation did not guarantee the equivalence between the original and obtained grammars. Thus it is problematic to identify the algorithmic difference that causes the performance. As a consequence, their comparison was only a suggestion as an empirical comparison of HPSG and LTAG parsers, as they noted in the paper.

On the other hand, we have performed a comparison between an HPSG parser with the HPSG factoring method and an HPSG parser with the LTAG factoring method, due to clarification of the correspondence between the original LTAG and converted HPSG-style grammars. We claim that the comparison of parsers in different grammar formalisms is valid only when we perform the experiments using the strongly equivalent grammars.

[5]We should mention that the LTAG parser employs the dynamic programming techniques analogous to the LTAG parser used in our first parsing comparison, while the HPSG parser employs the same CFG filtering techniques used in our second parsing comparison and further extends the algorithm by efficient unification techniques. The experimental results showed that the HPSG parser was more efficient than the LTAG parser.

There is another work on parsing comparison between different grammar formalisms. Schabes and Waters (Schabes and Waters 1995) showed strongly equivalent grammars based on context-free grammars and Lexicalized Tree Insertion Grammar (LTIG), which is a variant of TAG as to suppress its generative capacity. The objective of their work was not on the parsing comparison but on the strong lexicalization of CFG by LTIG. They reported that parsing with the TIG converted from a CFG achieved a parsing speed that was higher by a factor of 5 to 10 than parsing with the original CFG. It is noteworthy that they recognized their conversion from CFG to strongly equivalent LTIG as a kind of transformation of the grammar, and compared it with the other transformation on CFG such as the Greibach Normal Form (GNF: (Greibach 1965)) transformation and the Rosenkrantz procedure (Rosenkrantz 1967). We consider that the difference between the grammar conversion and these transformations lies in whether both input and output of the conversion are linguistic formalisms or not.[6] Conversion of grammars from one linguistic formalism to another is more intricate than the above transformations, because the conversion involves not only mathematical transformation but also linguistic correspondence between the two formalisms.

4.4 Further Collaboration between LTAG and HPSG using Our Results

There are two studies related to collaboration between the LTAG and HPSG formalisms. These studies made use of an HPSG-style grammar converted from an LTAG grammar by our grammar conversion and insights gained by results of parsing comparison, respectively.

Yakushiji et al. (Yakushiji et al. 2001) demonstrated that a debug tool *willex* designed for an HPSG grammar can be used for debugging of an HPSG-style grammar converted from the XTAG English grammar. In their study, they designed a grammar debugger for HPSG grammars using HPSG parsers. In their experiments, several defects of the XTAG English grammar have been successfully identified using HPSG parsers.

Oouchida et al. (Oouchida et al. 2004) established a concrete algorithm to approximate LTAG with CFG based on the fundamental idea of CF approximation for LTAG that we provided in Section 3.2.4, and derived a parsing algorithm for LTAGs. In their research, they observed the LTAG processing architecture and identified the grammatical construction that increases the number of non-terminals when applying the grammar rules to elementary trees during CF approximation of

[6]Note that the conversion from CFG to LTIG proposed by Schabes and Waters does not involve the linguistic correspondence between CFG and LTIG. This is because not only CFG but also LTIG is a synthetic representation framework and thus there is no linguistic requirement on the output LTIG, as is the same in CFG.

a given LTAG grammar, in order to obtain a better CFG approximation. Compared to the original CFG filtering designed for HPSG, this approximation makes use of the nature of the processing architecture chosen by LTAG, and is thus more suitable for parsing of LTAG. Their approach conforms to our claim that there must be different kind of optimization according to representation framework chosen by linguistic theories. Oouchida et al. have thus achieved a parsing algorithm that are not only empirically efficient but also has better bounds of theoretical time complexity $O(n^6)$,[7] which is not guaranteed when applying the HPSG parsers used in our experiments to HPSG-style grammars converted from LTAGs.

[7]This is achieved by using an LTAG parser in phase 2 of the two-phase parser.

Part II

Approach to Acquiring Lexical Resources from Corpora

Chapter 5

Background to Subcategorization Frame Acquisition

In this chapter, we introduce a subcategorization frame (SCF) as an essential information that should be included in lexical entries of lexicalized grammars. Section 5.1 describes linguistic phenomenon of subcategorization and its treatment in lexicalized grammars. Section 5.2 next reviews methods of acquiring SCFs automatically from raw or annotated corpora. Section 5.2.1 introduces studies on acquiring general SCF types from corpora while Section 5.2.2 focuses on studies on acquiring SCFs for lexicalized grammars. Section 5.3 finally mentions existing linguistic knowledge on subcategorization behaviors.

5.1 Verb Subcategorization and Its Treatment in Lexicalized Grammars

A subcategorization frame (SCF) for a word concerns *arguments*[1] of the predicate. Arguments are closely associated with the predicate and understood to complete its meaning, and then are distinguished from *adjuncts*. For example, in 1a, "*Mary*" is an argument for a predicate "*met*," but "*yesterday*" is an adjunct. The sentence is ungrammatical (1c) without the argument while grammatical without the adjunct (1b).

(1) a. He met Mary yesterday.

[1]Term *argument* in Section 2.1.1 slightly extends arguments in this context to modifiee for adjective/adverb. That is, in auxiliary tree, an anchor word (modifier) modifies another word (modifiee) whose label is the same as the foot node, and then we say that the modifiee is an argument of the modifier there.

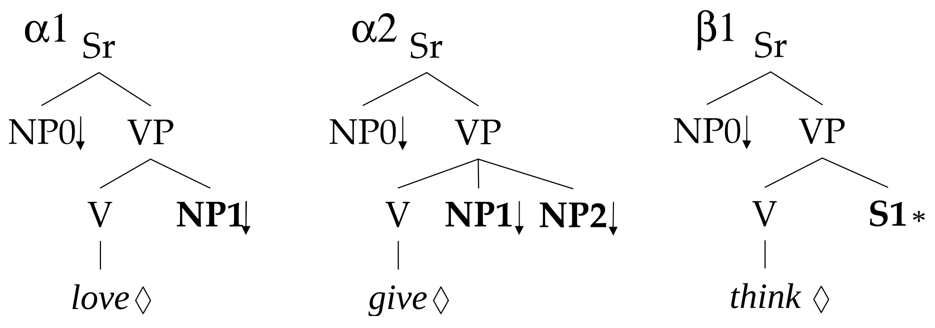


Figure 5.1: LTAG lexical entries for "*love*," "*give*," and "*think*" that exemplifies transitive, ditransitive, and sentential complement verbs.

b. He met Mary.

c. * He met.

A correct and coherent characterization of the argument-adjunct distinction is crucial for both defining and identifying SCF types. A variety of criteria have been proposed in the linguistic literatures to help make this distinction; examples include elimination test (Somers 1984), which involves eliminating an element from a sentence and observing whether the remaining sentence is still grammatical, and manually-tailored criteria and heuristics to examine argument-hood and adjunct-hood (Meyers et al. 1994), which is demonstrated in the sizable COMLEX syntactic dictionary (Grishman et al. 1994).

Given the argument-adjunct distinction, subcategorization concerns the specification of the number and type of arguments which the predicate requires for well-formedness. For example, some verbs take two NP complements (*e.g.*, *give* and *provide*), while others do not (*love* and *like*). Some verbs permit a *whether*-complement clause (*ask*, *wonder*), others permit a *that*-complement clause (*require*, *say*), while others permit neither (*give* and *provide*) and others permit both (*consider*). These specifications on the SCF types are sensitive to 'grammatical functions,' i.e., the specific grammatical roles the arguments can bear when present.

In the context of lexicalized grammars, subcategorization information for a word is used to select other words or phrases that the word can be combined with, and is thereby essential for lexical entries. As we have seen in Chapter 1 and Sections 2.1 and 3.1, the constraints on arguments of words are differently realized in individual grammar formalisms. In the LTAG formalism, the constraints on grammatical categories of arguments of a word are explicitly expressed in labels of leaf nodes in the elementary trees for the word. On the other hand, in the HPSG formalism,

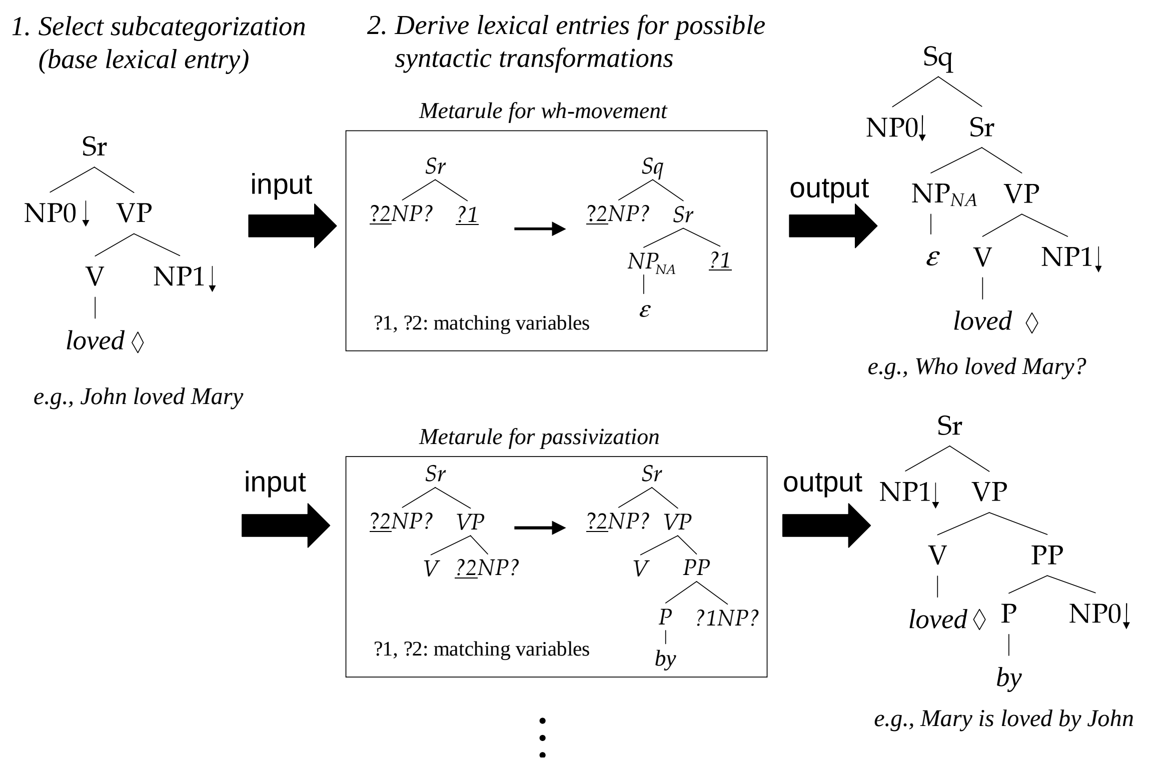

1. Select subcategorization (base lexical entry)
2. Derive lexical entries for possible syntactic transformations
Metarule for wh-movement
Metarule for passivization
input
output
?1, ?2: matching variables
e.g., John loved Mary
e.g., Who loved Mary?
e.g., Mary is loved by John

Figure 5.2: Metarule for lexicon organization: a case of deriving lexical entries for "*loved*"

these are simply listed in a feature structure for the word. For example, an elementary tree $\alpha1$ in Figure 5.1 shows an LTAG lexical entry for "*love*," which expresses the fact that "*love*" must be combined with one NP complement, i.e., a transitive verb, and an elementary tree $\beta1$ depicts a lexical entry for "*think*," which takes sentential complements.

Along with this SCF distinction, lexicalized grammars are usually equipped with an organized architecture of lexicon that derives lexical entries from one subcategorization frame and one possible syntactic alternation; *e.g.*, lexical rules in the HPSG framework (Pollard and Sag 1994; Briscoe and Copestake 1999; Nakanishi et al. 2004) and meta-rules (Becker 1994; Becker 2000; Prolo 2002) and metagrammar (Candito 1996; Abeillé and Candito 2000; Yoon 2004) in the LTAG framework. Figure 5.2 shows example generations of lexical entries by metarules; lexical entries for "*loved*" are derived from its declarative (or *base* entry). A Metarule takes a lexical entry whose structure matches the structural constraints specified by the left-hand side of the metarule, and executes structural replacement using structure matching variables (*?1* and *?2* in Figure 5.2). The key point is that metarules define syntactic alternations independently from each subcategorization frame, and thus all lexical entries can be derived by selecting one subcategorization frame and by applying possible syntactic alternations. Since the other architectures of lexical organization also express lexical entries by the types of subcategorization frames and syntactic alternation rules, we can decide possible lexical entries for words by selecting possible subcategorization frames for the word and possible syntactic alternation for the frame. Subcategorization acquisition thus equals to lexical acquisition if a grammar equips such organized architecture of lexicon.

We should mention that granularity of types of subcategorizations depends much on each linguistic theory. Briscoe and Carroll (Briscoe and Carroll 1997) extended the subcategorization types in two large-scale manually-tailored SCF lexicons (Boguraev and Briscoe 1987; Grishman et al. 1994), and identified 163 types of subcategorization frames, which are relatively independent from individual linguistic theories. These comprehensive SCF types are listed in Appendix A. Implemented lexicalized grammars usually have individual coding scheme of SCF types. The XTAG English grammar (XTAG Research Group 2001) has only 57 types of subcategorization frames and most of them are for compound expressions, while the LINGO English Resource grammar has finer distinction for subcategorizations and the number of the SCF types is 216. These differences stem from the fact that there are several different encodings of SCFs according to constraints the theories consider. The interested reader is referred to relevant discussions in Chapter 1 and Section 3.1.

5.2 Automatic SCF Acquisition

Because the subcategorization information is the core constraints included in lexical entries of the lexicalized grammars, it is quite important to determine the SCF types taken by a particular word. In this section, we first address problems on the manual construction of large-scale subcategorization lexicons. We then describe approaches to automatically acquiring subcategorization frames from raw or annotated corpora.

Because variations of types of subcategorization frames for a word are much more diverse than its part-of-speech variations, it is quite problematic to manually build comprehensive subcategorization lexicon; there are two problems reported for manually-tailored lexicons. First, the manually-tailored lexicons tend to show high precision but disappointing recall. Briscoe and Carroll (Briscoe 2001) manually analyzed associations between SCF types and predicates in 35,000 words of corpora, and then compared the resulting SCF lexicon with two manually-tailored SCF lexicons (Boguraev and Briscoe 1987; Grishman et al. 1994). Around 15% of the observed types of associations were not included in manually-tailored lexicons. Second, a manually-tailored lexicon lacks frequency information for SCFs that helps a lexicalized parser to accurately select the most probable parse (Carroll et al. 1998; Zeman 2002; Collins 2003).

In order to compensate for these shortcomings, automatic acquisition of SCF lexicon has emerged as one of the most important lexical acquisition tasks, and has been well studied in the literature (Brent 1993; Ushioda et al. 1993; Manning 1993; Ersan and Charniak 1996; Briscoe and Carroll 1997; Carroll and Rooth 1998; Gahl 1998; Lapata 1999; Kuhn et al. 1998; Sarkar 2000; Korhonen 2002; Miyao et al. 2004; Nakanishi et al. 2004). These studies are different in SCF types they assume and kinds of linguistic cues to identify arguments in the target corpora.

In what follows, we first introduce studies on the acquisition of general-purpose SCFs from corpora. We then mention studies on the SCF acquisition for lexicalized grammars.

5.2.1 SCF Acquisition for General SCF Types

Brent (Brent 1993) initiated the automatic subcategorization acquisition task. In his study, only six SCF types involving basic NP, sentential and infinitive phrases are extracted from raw corpora, by making use of a number of lexical cues to find unambiguous SCF appearances. Although the results show high precision, the recall is quite low. His approach is not extensible to all SCF types, and frequency information of each SCF type is totally unreliable due to the use of only unambiguous data.

Following Brent's study, various studies attempted to acquire finer SCF types from sizable raw corpora, making use of shallow analysis of sentences. Ushioda et al. (Ushioda et al. 1993) and Manning (Manning 1993) first employed an NP chunker for SCF acquisition. In their study, a finite-state NP chunker first parses part-of-speech tagged sentences. A set of SCF identification rules then extract SCFs from chunked data. Following their research, Gahl el al. (Gahl 1998) attempted the SCF acquisition in the context of corpus query systems. Lapata et al. (Lapata 1999) tried to acquire diathesis alternation using an NP chunker and a set of linguistics heuristics.

These approaches represent a clear improvement over the initial approach by Brent. The use of an NP chunker or a partial parser increases the number of corpus instances that include SCFs. However, in these chunker-based methods, due to lack of grammatical constraints that determine whether each chunk to be argument or adjunct, we need several heuristic rules that are difficult to design; for example, longest-match principle, a heuristic rule that determines a phrase boundary, is reported very unreliable.

In order to exploit more grammatical constraints to determine SCF types, some studies used phrase-structure parsers instead of NP chunkers to capture phrasal constraints. Ersan and Charniak (Ersan and Charniak 1996) started the use of a phrase-structure parser for SCF acquisition.

Their PCFG parser equips 1,209 rules for VP expansion, which are mapped to 16 SCF types. For example, if a rule VP $\rightarrow$ ADV NP is applied during parsing, this is mapped to a transitive SCF type. Carroll and Rooth (Carroll and Rooth 1998) trained their CFG parser using Expectation Maximization (EM) algorithm, with verb SCF as the hidden variable for the model. Briscoe and Carroll (Briscoe and Carroll 1997) attempted to extract comprehensive 163 types of SCFs that are extended from the types in two large manually-tailored SCF lexicons (Boguraev and Briscoe 1987; Grishman et al. 1994), using a full phrase structure parser.

Because the above research assumed a pre-determined set of SCF types to be extracted, it is problematic to apply their methods to new languages whose SCF types are not well studied. Making use of corpus with dependency annotation, Sarkar and Zaeman (Sarkar and Zeman 2000) first extracted associations between a predicate and its all dependants ('*observed frame*'). Since the dependants of a predicate include all of its arguments, observed frames subsume possible SCF types. However, in real-world sentences, a predicate is almost always accompanied by one or more adjuncts. They first determine subsumption relation among the observed frames according to subsumption among elements of the frames. For example, the frame NP-PP subsumes NP, NP-NP subsumes NP, and the like. They then identified correct SCF types by comparing frequency differences between an observed frame and its smaller subsets; when an observed frame is too infrequent and statistically rejected, then adding frequency counts of the observed frame to the most frequent frame of its subset frames.

These research efforts tried to extract general-purpose SCF types from corpora, However, when we want to extract SCFs with the individual SCF coding scheme defined by lexicalized grammars or lexical resources, we need some method to abstract away differences between acquired SCF types and the target SCF types or start with the target SCF type set. In the following section, we introduce methods (Kuhn et al. 1998; Carroll and Fang 2004; Miyao et al. 2004) for acquiring SCFs for lexicalized grammars.

5.2.2 SCF Acquisition for Lexicalized Grammars

First of all, Kuhn et al. (Kuhn et al. 1998) proposed semi-automatic method to acquire SCFs for their LFG grammar. They acquired three SCF types for their LFG grammar, taking part-of-speech tagged sentences as inputs. They first select sentences that include candidate verbs with the target SCF types using heuristic rules. They next parse the sentences with the grammar three times, each time hypothesizing a particular SCF type for the candidate verbs, and then select the correct SCF among hypothesized SCFs that obtain parses, using linguistic knowledge-based filters.

Finally, putative SCFs are examined by a human lexicographer. Although this method yields better precision and recall than a method without a grammar, we cannot obtain accurate SCF distributions due to the use of (strict) filtering of the target sentences. It is also difficult for this method to scale to finer SCF types the grammar defines.

Miyao et al. (Miyao et al. 2004) attempted to acquire HPSG lexical entries from parsed corpora. Their research follows the previous research on lexicalized grammar acquisition (Miyao et al. 2003), which has been proposed for LTAG (Xia 1999; Chen and Vijay-Shanker 2000; Chiang 2000) and CCG (Hockenmaier and Steedman 2002), and further pursues the grammar-engineering framework called *corpus-oriented grammar development*. In their method, they first assume a certain set of grammar rules in the target lexicalized grammar formalism. They then *externalize* linguistic knowledge to (annotated) sentences, in other words, annotate grammatical constraints that signs of each node in the parse trees must meet, in order to identify which grammar rule should be applied to construct each branching structure. They then acquire HPSG lexical entries, by inversely applying the identified grammar rule to each branching structure. Although resulting lexical entries are not lexically organized as in hand-crafted grammars, a subsequent study by Nakanishi et al. (Nakanishi et al. 2004) enable to decompose obtained lexical entries to SCFs (*lexeme*, in their study) and syntactic alternation defined by lexical rules. In their study, they manually design lexical rules that input one lexeme and output a lexical entry, and its inverse version that input a lexical entry and output a lexeme. By applying inverse lexical rules to each lexical entry, SCFs are induced. Thanks to the use of annotated corpora (Marcus et al. 1993) and heuristic rules that reflect linguistic knowledge, they obtained reliable SCFs for verbs in the annotated corpus as by-product. From viewpoints of SCF acquisition, as in the work by Sarkar and Zaeman (Sarkar and Zeman 2000), their method extracts SCF types and SCFs altogether from corpus. By using linguistic knowledge derived from the target linguistic theory instead of corpus-based statistics, they could obtain SCF types appropriate for the grammar they design. The validity of the linguistic knowledge is empirically attested by performing parsing experiments using the obtained lexical entries. However, as described in (Roland 2001), verb subcategorizations vary from one corpus to another, according to their discourse such as narrative, connected discourse, and single sentence production, and to their semantic domain. We cannot obtain possible SCF types for infrequent verbs when we use a finite size of annotated corpora.[2] That is, acquired lexicon accurate enough, but recall is somewhat disappointing for general purpose.

Finally, we introduce a method that makes use of acquisition methods for general-purpose SCF

[2]Briscoe and Carroll (Briscoe and Carroll 1997) discovered that manual analysis of 300 occurrences of each verb is sufficient to obtain an adequate SCF distribution for gold-standard. However in the WSJ corpus, only 48 verbs out of 3603 verbs appeared more than 300 times.

```
#S(EPATTERN :TARGET |yield|
            :SUBCAT (VSUBCAT NP)
            :CLASSES ((24 51 161) 5293)
            :RELIABILITY 0
            :FREQSCORE 0.26861903
            :FREQCNT 1 :TLTL (VV0)
            :SLTL ((|route| NN1))
            :OLT1L ((|result| NN2))
            :OLT2L NIL
            :OLT3L NIL :LRL 0))
```

Figure 5.3: An acquired SCF for a verb "*yield*"

types proposed by Carroll and Fang (Carroll and Fang 2004). In their study, they first acquire fine-grained SCFs using the unsupervised method proposed by Briscoe and Carroll (Briscoe and Carroll 1997) and Korhonen (Korhonen 2002). Figure 5.3 shows an example of one acquired SCF entry for a verb "*yield*." Each SCF entry has several fields about the observed SCF. We explain here only its portion related to this study. The `TARGET` field is a word stem, the first number in the `CLASSES` field indicates an SCF type, and the `FREQCNT` field shows how often words derivable from the word stem appeared with the SCF type in the training corpus. The obtained SCFs comprise the total 163 SCF types of relatively fine-grained SCFs, which are originally based on the SCFs in the ANLT (Boguraev and Briscoe 1987) and COMLEX (Grishman et al. 1994) dictionaries. In this example, the SCF type 24 corresponds to an SCF of transitive verb. They then obtain SCFs for the target lexicalized grammar (the LinGO ERG (Copestake 2002) in their study) using a handcrafted translation map from these 163 types to the SCF types in the target grammar. They reported that they could achieve a coverage improvement of 4.5% (52.7% to 57.2%) but that average parse time was doubled (9.78 sec. to 21.78 sec.). This is because they did not use any filtering method for the acquired SCFs to suppress the increase of the lexical ambiguity. Note that their method is extendable to any lexicalized grammars, if we could have a translation map from these 163 types to the SCF types in the target grammar.

As we have discussed above, there are several problems in existing methods of acquiring lexical resources for lexicalized grammars from raw or annotated corpora. When we use a method of acquiring lexical resources from annotated corpora, the precision of the acquired SCFs is high but it is dubious that the recall of the acquired SCFs is enough for our target domain. On the other hand, when we attempt to integrate SCFs acquired from raw corpora into existing lexical resources, it follows from results shown by Carroll and Fang (Carroll and Fang 2004) that we definitely need some method to control the quality of SCFs acquired from raw corpora.

5.3 Linguistic Knowledge on SCF behavior

This section introduces linguistic characteristics of subcategorization that can be employed for guiding filtering of noisy SCFs.

Because subcategorizations for a word involve information on arguments that complete the predicate's meaning, several studies in the linguistic literature (surveyed in (Levin 1993; McCarthy 2001)) have reported alternation relation among several SCFs for one word with the same meaning.

The alternation relation among SCFs for a word with the same meaning is called *diathesis alternation*. For example, some ditransitive verb taking SCF type NP-NP takes another SCF type NP-to_PP as follows:

(2) a. He gives [$_{\text{NP}}$ his father] [$_{\text{NP}}$ a letter]

b. He gives [$_{\text{NP}}$ a letter] [$_{\text{to\_PP}}$ to his father]

c. She presents [$_{\text{NP}}$ her daughter] [$_{\text{NP}}$ a doll]

d. She presents [$_{\text{NP}}$ a doll] [$_{\text{to\_PP}}$ to her daughter]

The alternation shown in the sentence 2a,2c and 2b,2d is called dative alternation. This alternation is known to be applicable to a verb of meaning similar to "*give*" and "*present*," which convey *change of possession*. The following sentences illustrate other alternations such as middle causative/inchoative alternation NP $\leftrightarrow$ none (3a $\leftrightarrow$ 3b), body-part possessor ascension alternation NP's-NP $\leftrightarrow$ NP-PP (4a $\leftrightarrow$ 4b), and an alternation related to SCFs of predicative complements NP-S $\leftrightarrow$ NP-NP (5a $\leftrightarrow$ 5b).

(3) a. Tome broke [$_{\text{NP}}$ the window]

b. The window broke

(4) a. Mary kissed [$_{\text{NP's}}$ John's] [$_{\text{NP}}$ shin]

b. He considers [$_{\text{NP}}$ John] [$_{\text{PP}}$ in the shin]

(5) a. He considers [$_{\text{NP}}$ John] [$_{\text{S}}$ to be a good teacher]

b. He considers [$_{\text{NP}}$ John] [$_{\text{NP}}$ a good teacher]

Levin (Levin 1993) has argued that the alternation behavior of SCFs for a verb can be predictable by the meaning of the verb, and inversely constructed verb classes that share the same alternation behavior of SCFs. Her hypothesis is attested against the established classes for verbs that take alternations involving prepositional phrases. Dorr (Dorr 1997) and Dang et al. (Dang et al. 1998) extended the Levin's classification by adding intersective and novel classes, respectively. Korhonen (Korhonen and Briscoe 2004) introduced 106 novel diathesis alternation as well as supplied new classes that are not comprehensively covered in the extant classification. These research efforts illustrated a regular behavior of SCF co-occurrences, which would be useful to guide subcategorization acquisition.

In the following two chapters, we first solve the task of identifying whether a word can have each observed SCF in order to augment lexicons that include only associations between words and SCFs. We assume that there are classes whose element words have identical SCF types. We obtain these classes by clustering acquired SCFs, using information available in the target lexicon, and then use the obtained classes to eliminate less likely SCFs. We second solve another task of obtaining accurate estimates for lexicons that include not only associations between words and SCFs but also their co-occurrence probabilities. In this task, we model co-occurrences between words and SCFs by the latent class models, assuming there are latent classes that govern the SCF distributions. Both of these studies are motivated from the aforementioned linguistic knowledge on the nature of SCF distributions.

Chapter 6

Filtering Method for SCF Lexicon Acquired from Raw Corpora

In this chapter, we describe a method of filtering putative SCFs acquired from raw corpora, in order to augment existing lexical resources that only include associations between words and SCFs, such as hand-coded lexicons of lexicalized grammars. We make use of the reliable information on co-occurrences between SCF types and words, which are available in the target lexical resource, in order to guide filtering of less likely SCFs in the acquired noisy lexicon.

The basic idea of our method is first to obtain word classes whose element words have the same set of SCFs, using not only acquired SCFs but also existing SCFs in the target lexicon. We eliminate less plausible acquired SCFs and add plausible unseen SCFs for a word according to a cluster to which the word belongs.

In the following sections, Section 6.1 introduces *an SCF confidence* of each SCF type for a word. An SCF confidence v_{ij} is a probabilistic quantity that expresses how strong the evidence is that the word w_i has an SCF f_j, and can be defined for both SCFs in the acquired and the target lexicons. Section 6.2 describes a clustering method of SCF confidence vectors for words. Each element of an SCF confidence vector for a word represents an SCF confidence of each SCF type for the word. Section 6.3 mentions cut-off methods that exploit the obtained clusters along with corpus-based statistics. Section 6.4 shows an application of our method to SCFs acquired from raw corpora in order to augment hand-coded lexicons of two large-scale lexicalized grammars. Section 6.5 shows related work to our filtering method using alternation behaviors between SCFs.

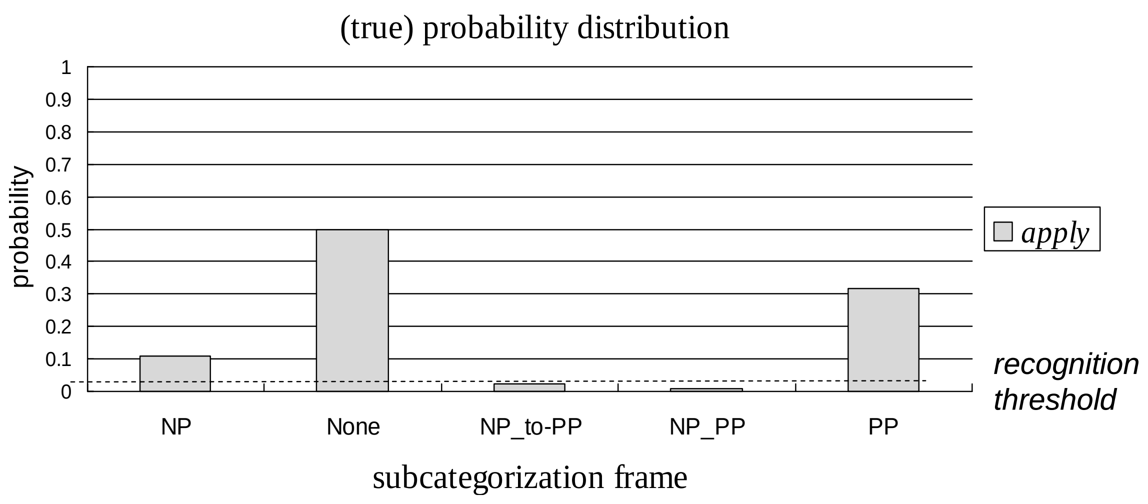


Figure 6.1: SCF probability distribution for "*apply*"

6.1 Estimation of SCF Confidence Vectors

We first create *an SCF confidence vector* v_i for each word w_i using acquired SCFs and the SCFs in the target lexicon. Each element v_{ij} in v_i represents *an SCF confidence* of an SCF type f_j for a word w_i, which expresses how strong the evidence is that the word w_i has SCF f_j. SCF confidence vectors for words are objects of our clustering, and the clustering algorithm will be described in Section 6.2.

Language Model with SCF Confidence In this study, we assume that a word w_i appears with each SCF type f_j with a certain (non-zero) probability $\theta_{ij} (= p(f_j|w_i) > 0$ where $\sum_j \theta_{ij} = 1)$, but only SCFs whose probabilities exceed a certain threshold are regarded as SCFs for the word. In other words, we regard an SCF f_j for a word w_i whose probability θ_{ij} is less than a certain threshold as a noise or a very exceptional usage for w_i. We hereafter call this threshold *recognition threshold*. Figure 6.1 depicts an imaginary (true) probability distribution of SCF for *apply*. We regard a probability that the probability of that SCF exceeds the recognition threshold as an SCF confidence of each SCF type, i.e., $v_{ij} = P(\theta_{ij} > \textit{recognition threshold})$. In Figure 6.1, SCF types NP, None, and PP are recognized as SCFs for "*apply*." We can reflect the reliability of SCFs associated with a word in the target lexicon by setting their SCF confidences close to 1.

One intuitive way to estimate an SCF confidence is to assume that an observed probability, i.e., relative frequency which is obtained by the maximum likelihood estimation, is equal to a probability θ_{ij} of an SCF type f_j for a word w_i ($\theta_{ij} = n(w_i, f_j)/\sum_j n(w_i, f_j)$ where $n(w_i, f_j)$ is a frequency that a word w_i appears with an SCF type f_j in corpora). When the relative frequency of f_j for a word w_i exceeds the recognition threshold, its SCF confidence v_{ij} is set to 1, and otherwise v_{ij} is set to 0. However, an observed probability is unreliable for infrequent words. For example, when we use an SCF confidence derived from a relative frequency as above, we cannot distinguish cases where a word w_1 appears once with an SCF type f_j and a word w_2 appears 100 times, always with the SCF type f_j, which are both the relative frequency 1. Moreover, when we want to encode an SCF confidence of reliable SCFs in the target lexicon, we cannot distinguish the SCF confidences of those SCFs with SCF confidences of the acquired SCFs.

Bayesian Estimation of SCF Confidence Another promising way to estimate an SCF confidence, which we adopt in this study, is to assume a probability θ_{ij} as a stochastic variable in the context of Bayesian statistics (Gelman et al. 1995). In this context, *a posteriori* distribution of the probability θ_{ij} of an SCF type f_j for a word w_i is given by:

$$\begin{aligned} p(\theta_{ij}|D) &= \frac{P(\theta_{ij})P(D|\theta_{ij})}{P(D)} \\ &= \frac{P(\theta_{ij})P(D|\theta_{ij})}{\int_0^1 P(\theta_{ij})P(D|\theta_{ij})d\theta_{ij}}, \end{aligned} \tag{6.1}$$

where $P(\theta_{ij})$ is *a priori* distribution, and D is the data we have observed. In this study, we assume that every occurrence of SCFs in the data D is independent of each other. In other words, we regard the data D as Bernoulli trials. When we observe the data D that a word w_i appears n times in total and $x(\leq n)$ times with SCF type f_j,[1] its conditional distribution is represented by a binomial distribution:

$$P(D|\theta_{ij}) = \binom{n}{x} \theta_{ij}^x (1 - \theta_{ij})^{(n-x)}. \tag{6.2}$$

To calculate the *a posteriori* distribution in Equation 6.1, we need to define the *a priori* distribution $P(\theta_{ij})$. The question is which probability distribution of θ_{ij} appropriately reflects prior knowledge. In other words, the *a priori* distribution should encode the knowledge we use to estimate SCFs for words unknown to us. We therefore determine the *a priori* distribution $P(\theta_{ij})$ from distributions of observed probability values of f_j for all the words seen in corpora[2] by us-

[1]The values of `FREQCNT` is used to obtain n and x.

[2]Note that we assume that $P(\theta_{ij})$ and $P(\theta_{ij'})$ are independent, although they are not independent when we consider diathesis alternation between subcategorization frames which we introduced in Section 5.3. By further assuming that $P(\theta_{ij})$ is equal to $P(\theta_j)$, we estimated *a priori* distribution separately for each type of SCF from words that appeared more than 50 times in the training corpus in the following experiments.

ing a method described in (Tsuruoka and Chikayama 2001). In their study, they assume *a priori* distribution as the *beta* distribution defined as:

$$p(\theta_{ij}|\alpha,\beta) \quad = \quad \frac{\theta_{ij}^{\alpha-1}(1-\theta_{ij})^{\beta-1}}{B(\alpha,\beta)}, \tag{6.3}$$

where $B(\alpha,\beta) = \int_0^1 \theta_{ij}^{\alpha-1}(1-\theta_{ij})^{\beta-1}d\theta_{ij}$. The values of α and β are determined by moment estimation.[3] By substituting Equations 6.2 and 6.3 into Equation 6.1, we finally obtain the *a posteriori* distribution $p(\theta_{ij}|D)$ as:

$$p(\theta_{ij}|\alpha,\beta,D) \quad = \quad c \cdot \theta_{ij}^{x+\alpha-1}(1-\theta_{ij})^{n-x+\beta-1}, \tag{6.4}$$

where

$$c \quad = \quad \frac{\binom{n}{x}}{B(\alpha,\beta)\int_0^1 P(\theta_{ij})P(D|\theta_{ij})d\theta_{ij}}. \tag{6.5}$$

When we set the recognition threshold to t, we can calculate an SCF confidence v_{ij} that a word w_i can have f_j by integrating the *a posteriori* distribution $p(\theta_{ij}|D)$ from the threshold t to 1:

$$v_{ij} \quad = \quad \int_t^1 c \cdot \theta_{ij}^{x+\alpha-1}(1-\theta_{ij})^{n-x+\beta-1}d\theta_{ij}. \tag{6.6}$$

By using this SCF confidence, we represent an SCF confidence vector v_i for a word w_i in the acquired SCF lexicon.[4]

In order to combine SCF confidence vectors for words acquired from corpora and those for words in the target lexicon, we also define an SCF confidence vectors v'_i for a word w'_i in the target lexicon by:

$$v'_{ij} \quad = \quad \begin{cases} 1-\epsilon_{fp} & w'_i \textit{ has } f_j \textit{ in the lexicon} \\ \epsilon_{fn} & \textit{otherwise}, \end{cases} \tag{6.8}$$

[3]The expectation and variance of the *beta* distribution are made equal to those of the observed probability values.

[4]By using the fact that $\int_0^1 P(\theta_{ij}|\alpha,\beta) = 1$, we can calculate v_{ij} as follows.

$$\begin{aligned} v_{ij} \quad &= \quad \frac{\int_t^1 c \cdot \theta_{ij}^{x+\alpha-1}(1-\theta_{ij})^{n-x+\beta-1}d\theta_{ij}}{\int_0^1 c \cdot \theta_{ij}^{x+\alpha-1}(1-\theta_{ij})^{n-x+\beta-1}d\theta_{ij}} \\ &= \quad \frac{\int_t^1 \theta_{ij}^{x+\alpha-1}(1-\theta_{ij})^{n-x+\beta-1}d\theta_{ij}}{\int_0^1 \theta_{ij}^{x+\alpha-1}(1-\theta_{ij})^{n-x+\beta-1}d\theta_{ij}} \end{aligned} \tag{6.7}$$

where ϵ_{fp} and ϵ_{fn} expresses a probability that a mapping w'_i to f_j in the lexicon is false (false positive) and a probability that a mapping w'_i to f_j excluded in the lexicon is in fact true (false negative), both of which express unreliability of the lexicon. In this study, we trust the lexicon as much as possible by setting ϵ_{fp} and ϵ_{fn} to the machine epsilon.

In the following experiments we focus on solving a problem called '*missing of known SCF types for unknown words*.' We express SCF confidence vectors for all the words known to the target lexicon by Equation 6.8 while we express SCF confidence vectors for the words unknown to the target lexicon (and hence included only in the acquired SCF lexicon) by Equation 6.6.[5]

6.2 Clustering of SCF Confidence Vectors

We next present a clustering algorithm of words according to their SCF confidence vectors. Given k initial representative vectors called *centroids*, our algorithm iteratively updates clusters by assigning each data object to its closest centroid and recomputing centroids until cluster members become static, as depicted in Figure 6.2. This clustering algorithm determines whether the word takes each SCF type or not.

Although this algorithm is roughly analogous to the k-Means algorithm, it is different from k-Means in important respects. Because our clustering algorithm does not handle probabilistic distributions on SCF types for words but SCF confidence vectors for words, the resulting clusters are assumed to be word classes that express *SCF co-occurrences* for the member words. Each element c_{ij} of the centroid c_i of the clusters must thus not be a continuous quantity but a discrete value of 1 or 0, which express whether the words in that classes C_i have the corresponding SCF type f_j or not.

We then derive a function d that calculates a probability which a word w_i should have an SCF set represented by a centroid c_m:

$$d(v_i, c_m) = \prod_{c_{mj}=1} v_{ij} \cdot \prod_{c_{mj}=0} (1 - v_{ij}). \tag{6.10}$$

[5]It should be noted that because human lexicographers must have carefully examined whether the SCF type is appropriate for the word, ϵ_{fn} should be much higher than ϵ_{fp} in nature. This leads to another serious problem called '*missing of known SCF types for known words*.' We can solve this problem by performing experiments using different SCF confidence vectors for words that are included in both the target lexicon and the acquired SCF lexicon, as follows:

$$v'_{ij} = \begin{cases} 1 - \epsilon_{fp} & w'_i \textit{ has } f_j \textit{ in the lexicon} \\ \int_t^1 c \cdot \theta_{ij}^{x+\alpha-1}(1-\theta_{ij})^{n-x+\beta-1} d\theta_{ij} & otherwise. \end{cases} \tag{6.9}$$

```
INPUT:   a set of SCF confidence
           vectors $\mathcal{V} = \{v_1, v_2, \ldots, v_n\} \subseteq \mathbf{R}^m$
         a function $d : \mathbf{R}^m \times \mathbf{Z}^m \to \mathbf{R}$
         a function to compute a centroid
           $\mu : \{v_{j_1}, v_{j_2}, \ldots, v_{j_l}\} \to \mathbf{Z}^m$
         initial centroids $\mathcal{C} = \{c_1, c_2, \ldots, c_k\} \subseteq \mathbf{Z}^m$
OUTPUT:  a set of clusters {$C_j$}

procedure clustering_scf_confidence_vectors($\mathcal{V}$, {$C_j$})
begin
  while cluster members are not stable do
    foreach cluster $C_j$
         $C_j = \{v_i \,|\, \forall c_l, d(v_i, c_j) \geq d(v_i, c_l)\}$    $\cdots$ (1)
    end foreach
    foreach clusters $C_j$
         $c_j = \mu(C_j)$                              $\cdots$ (2)
    end foreach
  end while
  return {$C_j$}
end
```

Figure 6.2: Clustering algorithm for SCF confidence vectors

By using this function, we can determine the closest cluster as $\underset{C_m}{\operatorname{argmax}}\, d(v_i, c_m)$ ((1) in Figure 6.2).

After every assignment, we calculate a next centroid c_m of each cluster C_m ((2) in Figure 6.2) by comparing a probability that the words in the cluster have an SCF type f_j and a probability that the words in the cluster do not have the SCF type f_j:

$$c_{mj} \quad = \quad \begin{cases} 1 \quad when \displaystyle\prod_{v_i \in C_m} v_{ij} > \prod_{v_i \in C_m} (1 - v_{ij}) \\ 0 \qquad\qquad\qquad\qquad\qquad \textit{otherwise}. \end{cases} \tag{6.11}$$

This calculation of each element of the centroid is motivated by the nature of SCF confidence. Since an SCF confidence of f_j for a word w_i expresses how strong the evidence is that the word w_i has an SCF type f_j. Our objective of this clustering is to determine whether the word w_i have an SCF type f_j, we assign it not to a continuous quantity but to a discrete value, which exactly determine whether the word w_i in that cluster have an SCF type f_j. This is determined by SCF confidences of all members in that cluster. As a result, the centroid expresses a SCF co-occurrence for the words in that cluster. Since cluster member consist of both SCF confidence vectors obtained from the acquired SCF lexicon and the target lexicon, SCF co-occurrence expressed by the previous centroids can be changed when a large number of corpus evidences, i.e., observed SCF confidence vectors, suggest that their class should have different SCF co-occurrence.

We should address the way to determine the number of clusters and initial centroids. In this study, we assume that the most of the possible set of SCF types for words are included in the target lexicon,[6] and take advantages of the existing sets of SCF types for the words in the lexicon to determine the number of clusters and initial centroids. We first extract SCF confidence vectors from the lexicon of the grammar. By eliminating duplications from them and regarding $\epsilon_{fp} = 0$ and $\epsilon_{fn} = 0$ and in Equation 6.8, we obtain initial centroids c_m. We then set the number of clusters k to the number of c_m.

6.3 Cut-off Methods Exploiting the Obtained Clusters

We finally update the acquired SCFs using the obtained clusters and corpus-based statistics. In experiments, we compare the following three cut-off methods which exploit SCF relative frequency, SCF confidence, and word classes obtained by clustering of SCF confidence vectors.

Frequency cut-off This cut-off method performs filtering of acquired SCFs by setting a threshold for relative frequencies of SCFs. Korhonen (Korhonen 2002) reported that this filtering method works better than the other filtering methods employed for SCF acquisition in the literature (*e.g.*, binomial hypothesis testing (Brent 1993; Manning 1993; Ersan and Charniak 1996; Briscoe and Carroll 1997; Lapata 1999; Sarkar and Zeman 2000) and log-likelihood ratio (Dunning 1993; Gorrell 2002; Sarkar 2000)).

Confidence cut-off This cut-off method uses SCF confidence for SCFs instead of their relative frequency in order to perform thresholding. Note that without thresholding, all the SCF types are associated with each word, because the Bayesian estimation of SCF confidences assigns nonnegative SCF confidences to all the SCF types for each word. We refer to the above procedure as *confidence cut-off* t when the SCF confidences are estimated under the recognition threshold t.

Centroid cut-off This cut-off method first obtains SCFs by eliminating an SCF type f_j for a word $w_i \in C_m$ when c_{mj} is 0, because the element c_{mj} of a centroid c_m in a cluster C_m represents

[6] When the target lexicon is less accurate, we can determine the number of clusters using algorithms that determines the number of clusters (Bischof et al. 1999; Pelleg and Moore 2000; Hamerly 2003).

whether the words in the cluster can have an SCF type f_j. We further performs filtering of the selected SCFs by setting a threshold for an SCF confidence. We refer to the above procedure as *centroid cut-off* t when the SCF confidences are estimated under the recognition threshold t.

Note that frequency cut-off and confidence cut-off use only corpus-based statistics to eliminate SCFs. By comparing the frequency cut-off and the confidence cut-off t estimated under the recognition threshold t, we can observe the impact of the SCF confidence estimation. On the other hand, by comparing the confidence cut-off t and centroid cut-off t, we can evaluate the effect of the use of word classes obtained by the clustering of SCF confidence vectors.

6.4 Experiments on Filtering SCF Lexicon Acquired from Raw Corpora

We applied our filtering methods to SCFs acquired from 135,902 sentences of mobile phone newsgroup postings archived by Google.com, which is the data used in (Carroll and Fang 2004). As shown in Section 5.2.2, the system of (Briscoe and Carroll 1997) distinguishes 163 different SCF types. The number of acquired SCFs was 14,783 for 3,864 word stems, while the number of SCF types observed for at least one verb in the data was 97 out of 163. We then translated these SCFs into the SCFs of the XTAG English grammar (XTAG Research Group 2001) and the LinGO ERG (Copestake 2002)[7] using translation mappings built by Ted Briscoe and Dan Flickinger. They defined a map from 23 of the 163 SCF types into 13 (out of 57 possible) XTAG SCF types, and 129 of the 163 SCF types into 54 (out of 216 possible) ERG SCF types. These mappings are listed in Appendix A. The acquired SCFs for which translation mappings are not defined are disregarded in the following experiments.

In the following filtering experiments, we evaluate our method in terms of prediction of SCF types for unknown words. We first chose test words from the words that are included both in the acquired lexicon and the lexicon of the lexicalized grammar. We then split each lexicon of the the lexicalized grammar into the lexicon for the test words (testing data) and the lexicon for the other words (training data). We then applied our method to the acquired SCFs for the test words using the training data. By evaluating the resulting SCFs with the testing data, we can estimate the extent to which our method can safely filter out only less likely SCFs for words unknown to the grammar.

[7]We used the same version of the LinGO ERG as (Carroll and Fang 2004) (1.4; April 2003) but the map is updated.

Table 6.1: Precision and recall of XTAG SCFs filtered using frequency cut-off, confidence cut-off, and centroid cut-off

Cut-off		Threshold	Precision	Recall	F_1
frequency cut-off		0	0.300	0.657	0.412
		0.1528	0.550	0.492	0.519
confidence cut-off	0.05	0	0.116	1	0.208
	0.05	0.9080	0.667	0.629	0.647
centroid cut-off	0.05	0	0.590	0.700	0.640
	0.05	0.87337	0.697	0.652	**0.674**

Here, we assume that the existing SCFs for the words in the lexicon are more reliable than the other SCFs for those words.

The XTAG lexicon was split into 9,437 SCFs for 8,399 word stems as training and 423 SCFs for 280 word stems as testing, while the ERG lexicon was split into 1,608 SCFs for 1,062 word stems as training and 292 SCFs for 179 word stems as testing. Both testing words are randomly selected. We extracted SCF confidence vectors from the training SCFs and the acquired SCFs for the words in the testing SCFs. The number of the resulting data objects was 8,679 for XTAG and 1,241 for ERG.

The number of initial centroids[8] extracted from the training SCFs was 49 for XTAG and 53 for ERG. We then performed clustering of 8,679 data objects into 49 clusters and 1,241 data objects into 53 clusters, and then evaluated the resulting SCFs by comparing them to the testing SCFs.

We evaluated the resulting SCFs for the test verbs in terms of *precision* and *recall*, which are respectively defined as follows:

$$\text{precision} = \frac{\text{Correct SCFs for the words in the resulting SCFs}}{\text{All SCFs for the words in the resulting SCFs}} \tag{6.12}$$

$$\text{recall} = \frac{\text{Correct SCFs for the words in the resulting SCFs}}{\text{All SCFs for the words in the test SCFs}} \tag{6.13}$$

We set the recognition threshold for confidence cut-off and centroid cut-off to 0.05 in these experiments. Following the naming conversion we introduced in Section 6.3, we refer to these cut-offs as confidence cut-off 0.05 and centroid cut-off 0.05. We then obtained the threshold t_{best} for relative frequencies and SCF confidences that maximize F_1 measure, which is the harmonic mean of

[8]We used the vectors that appeared for more than one word.

Table 6.2: Precision and recall of ERG SCFs filtered using frequency cut-off, confidence cut-off, and centroid cut-off

Cut-off		Threshold	Precision	Recall	F_1
frequency cut-off		0	0.170	0.736	0.276
		0.1574	0.455	0.541	0.495
confidence cut-off	0.05	0	0.030	1.000	0.059
	0.05	0.9536	0.500	0.610	0.549
centroid cut-off	0.05	0	0.416	0.651	0.507
	0.05	0.8742	0.515	0.630	**0.567**

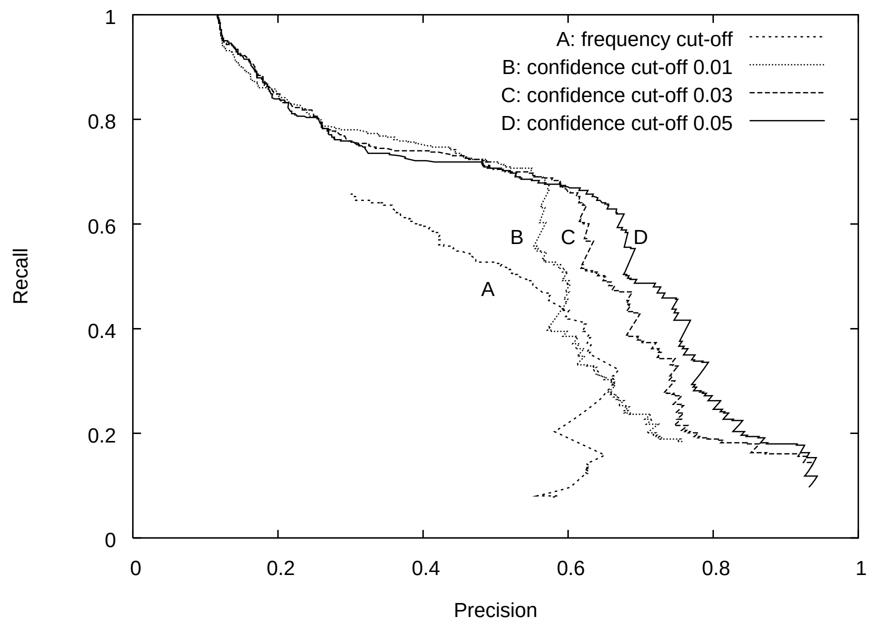
precision and recall.

$$F_1 = \frac{2 \cdot \text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}} \tag{6.14}$$

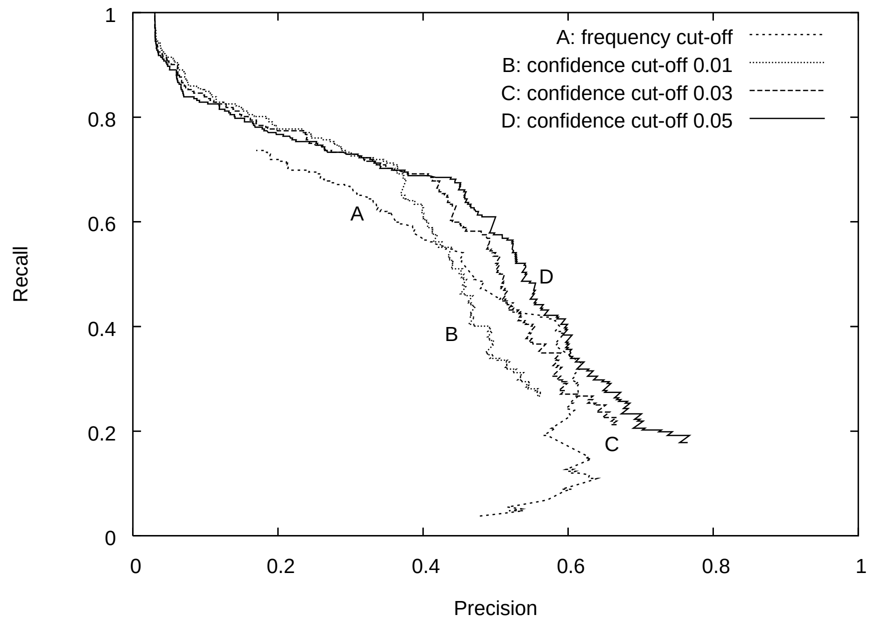
$$t_{best} = \operatorname*{argmax}_t F_1 \tag{6.15}$$

Table 6.1 and 6.2 show precision and recall with threshold t_{best} for the resulting SCFs using frequency cut-off, confidence cut-off 0.05, and centroid cut-off 0.05. The acquired SCFs are completely noisy when we use no filtering method (frequency cut-off of threshold 0). This is quite apparent for the resulting SCFs for ERG (F_1 = 0.276). The centroid cut-off performs best among all cut-offs. In the following, we examine the performance difference between these cut-off methods in detail, by ranging the recognition threshold to SCF confidences from 0.01 to 0.05.

We first compared confidence cut-off with frequency cut-off to observe the effect of Bayesian estimation. Figure 6.3 shows precision and recall of the SCFs obtained using frequency cut-off and confidence cut-off 0.01, 0.03, and 0.05 by varying threshold for the SCF confidences and the relative frequencies from 0 to 1. The graph indicates that the confidence cut-offs achieved higher recall than the frequency cut-off, thanks to the *a priori* distributions. Through a comparison among the three confidence cut-offs, we can conclude that we can improve precision using higher recognition thresholds while we can improve recall using lower recognition thresholds. This is quite consistent with our expectations. We can also observe that the precision of confidence cut-offs monotonously increased when we use the higher threshold for the SCF confidences, while the precision of frequency cut-offs decreased when we increases the threshold for relative frequencies above a certain value. This implies that SCF confidence is a better indicator to filter out implausible SCFs.

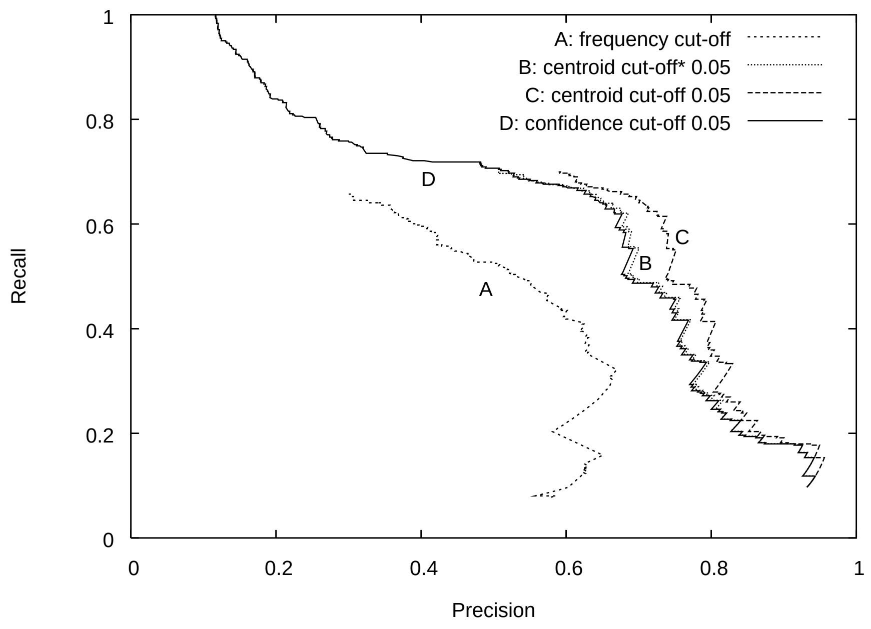


XTAG

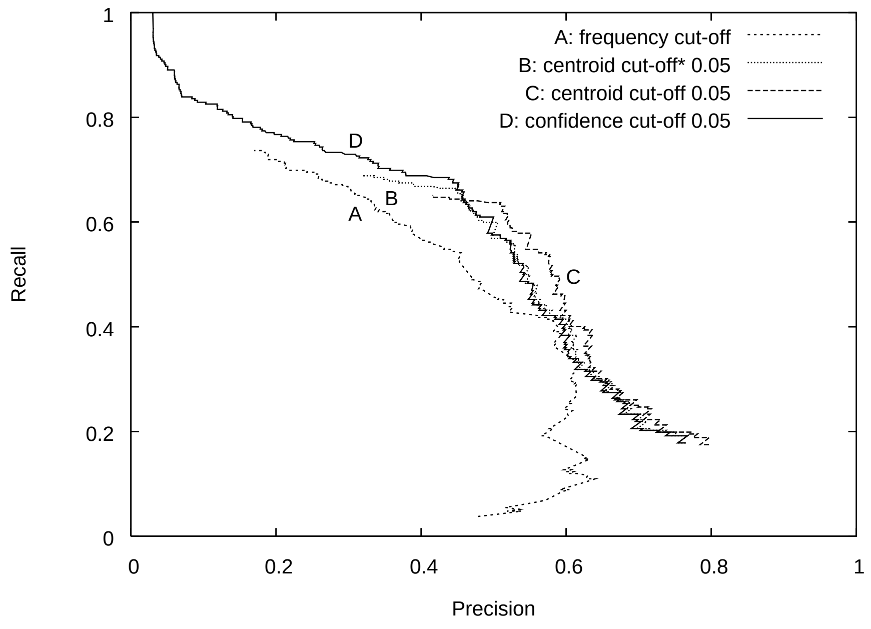


ERG

Figure 6.3: Precision and recall of the SCFs filtered using frequency cut-off and confidence cut-off: the XTAG English grammar (above) the LinGO ERG (below)



XTAG



ERG

Figure 6.4: Precision and recall of the resulting SCFs using confidence cut-off and centroid cut-off: the XTAG English grammar (left) the LinGO ERG (right)

We then compare centroid cut-off with confidence cut-off to observe the effect of clustering. Figure 6.4 shows precision and recall of the resulting SCFs using centroid cut-off 0.05 and the confidence cut-off 0.05 by varying the threshold for the SCF confidences. In order to show the effect of the use of the training SCFs, we also performed clustering of SCF confidence vectors in the acquired SCFs with random initialization (k = 49 (for XTAG) and 53 (for ERG); centroid cut-off 0.05*). The graph shows that clustering is meaningful only when we employ the reliable SCFs in the manually-coded lexicon. Also, the centroid cut-off using the lexicon of the grammar boosted precision compared to the confidence cut-off.

The difference between the effects of our method on XTAG and ERG would be due to the finer-grained SCF types of ERG. This resulted in lower precision of the acquired SCFs for ERG, which prevented us from distinguishing infrequent (correct) SCFs from SCFs acquired in error. However, since unusual SCFs tend to be included in the lexicon, we will be able to assign unknown words with smaller SCF variations to correct clusters as we have achieved in the experiments with XTAG.

6.5 Related Work

In this section, we mention methods of filtering SCFs using linguistic knowledge such as diathesis alternation and semantic classes induced by existing lexical resources, respectively. We also describe a method for clustering verbs' SCF patterns acquired from annotated corpora to induce verb semantic classes.

Korhonen (Korhonen 1998) employed diathesis alternation between SCFs to guide filtering of noisy SCFs acquired by Briscoe and Carroll's SCF acquisition system (Briscoe and Carroll 1997). She first constructed alternation rules that take the form of SCF A $\rightarrow$ SCF B from correlations between the two SCFs in the ANLT syntactic dictionary. She next assigned a probability for each alternation rule by dividing the number of ANLT verbs that have both SCF A and SCF B by the number of of ANLT verbs that have SCF A. She then used the obtained alternation probabilities to ameliorate observed probabilities.[9] Roughly speaking, when a verb has both SCF A and SCF B in the acquired lexicon, the probability that the verb has SCF A is improved by the alternation probability SCF A $\rightarrow$ SCF B; otherwise lowered by the alternation probability. Her method improved the system's type precision by 4% and type recall by 5% over the baseline results obtained using binomial hypothesis testing, which have been reported to be worse than thresholding according to relative frequencies (Korhonen 2002). Although this approach is applicable to augmenting lexicons

[9]Precisely speaking, she employed the alternation probabilities to ameliorate probabilities assigned by binomial hypothesis testing.

of lexicalized grammars, potential performance improvement will be less significant.

Exploiting the characteristics of correlations among a verb meaning and its syntax behaviors, Korhonen (Korhonen 2002) further promoted semantically-motivated subcategorization acquisition. She first semi-automatically determined verb semantic classes using Levin's verb classification (Levin 1993) and WordNet (Fellbaum 1998), which defines verb semantic hierarchy. She next employed SCF distributions for representative verbs in the obtained verb classes to calculate accurate back-off estimates for the verbs in the classes. She obtained SCFs by using Briscoe and Carroll's SCF acquisition system, and employed the obtained back-off estimates to perform semantically-driven smoothing to maximum-likelihood estimates of SCFs for (infrequent) verbs. She finally employed thresholding for the smoothed maximum-likelihood estimate of each SCF, and obtained more reliable sets of SCFs. This approach drastically improved the system's type precision and type recall. However, this method cannot scale to SCF acquisition for lexicons of lexicalized grammars, because the method is semi-automatic and depends entirely on the established semantic classes and the semantic hierarchy. On the other hand, our filtering method proposed in this chapter is fully automatic and only assumes the target lexicon resources. Our method is therefore applicable to any lexicalized grammar with a particular way of SCF encodings that describes any natural languages.

Oishi (Oishi 1998) demonstrated a method of obtaining deep case frames (analogous to semantic classes) of Japanese verbs by clustering the verbs according to their surface case patterns (analogous to subcategorization frames) which consist of postpositional particles along with possible substitutions by other postpositional particles. By virtue of the manually-tailored heuristic rules, he achieved relatively accurate deep case frames of verbs. His work is a strong indication that there are certain classes that govern surface case patterns (subcategorization frames) for words in languages other than English.

6.6 Chapter Summary

In this chapter, we presented a method of improving the quality of SCFs acquired from raw corpora using existing lexical resources. Our method assumes (discrete) associations between words and SCFs in the target lexical resources, and applies linguistically-motivated filtering to associations between words and SCF types in acquired SCF lexicon. We first acquire SCFs from raw corpora by existing subcategorization frame acquisition. We next maps SCF distributions for a word in the acquired SCF lexicon to SCF confidence vectors for the word by the Bayesian statistics approach. We also maps SCF associations for a word in the target lexicon to SCF confidence vectors for the word. By assuming the reliability of associations between SCF types and words in the target lexicon to be more reliable than those in the acquired SCF lexicon, we can take advantages of co-occurrence tendencies among SCF types that already exist in the target lexical resource. We then propose a new clustering algorithm for SCF confidence vectors, which classify the words into classes whose element words have the same set of SCF types. Finally, we introduced simple cut-off methods that use SCF confidences and word clusters obtained by clustering.

We applied our cut-off methods to SCFs acquired from raw corpora in order to augment lexicons of the XTAG English grammar and the LinGO ERG, respectively. Experimental results revealed that we can improve the recall of the resulting SCFs when we use Bayesian estimation instead of maximum-likelihood estimation to calculate a probability that the word have each SCF type. The results have also shown that we can eliminate implausible acquired SCFs, preserving correct SCFs, by virtue of obtained word classes that are derived from confidence vectors in both the acquired SCFs and the existing SCFs in the target lexicon. This is also first ever evaluation of augmenting the XTAG English grammar with SCFs acquired from corpora.

Chapter 7

Smoothing Method for SCF Lexicon Acquired from Annotated Corpora

In this chapter, we propose a method of constructing a probabilistic lexicon that consists of accurate SCF distributions for the other type of lexical resource, namely, a lexicon automatically acquired from annotated corpora. Since acquired associations between words and SCFs accompany their frequency counts, we conduct smoothing of the SCF distributions for words and thereby attempt to compensate for the lack of recall by assigning non-zero probabilities to new associations between known words and known SCF types.

As we have seen in Section 5.3, Levin extensively investigated groupings of verbs according to their alternation behaviors, and showed that there is a certain set of classes that share the same set of SCF types. In this chapter, we model this observation by a probabilistic model that handles co-occurrence events (words and SCFs, in this case), called *the latent semantic models*.

Section 7.1 introduces preliminaries for latent class models, which capture co-occurrence events by assuming unobserved latent class, and introduces its variant, the Probabilistic Latent Semantic Analysis (PLSA) (Hofmann 1999a; Hofmann 1999b), which we adopt to model co-occurrences between words and SCFs. Section 7.2 describes our modeling of co-occurrences between words and SCFs that makes use of the PLSA model. By using this PLSA model for SCF distributions, Section 7.3 describes a smoothing method of SCF distributions, which combines the PLSA model with models that are directly estimated from raw frequency counts by a method of linear interpolation. Section 7.5 mentions related work to our modeling of SCF distributions.

7.1 Preliminaries

In this section, we first describe the Probabilistic Latent Semantic Analysis, which is a kind of latent class models for co-occurrence data in natural language. We then introduce the Expectation Maximization (EM) Algorithm (Dempster et al. 1997), which is an iterative optimization algorithm used for maximum likelihood estimation of parameters in latent variable models.

7.1.1 Probabilistic Latent Semantic Analysis

The Probabilistic Latent Semantic Analysis (PLSA) (Hofmann 1999a; Hofmann 1999b) is a kind of latent class models (Brown et al. 1999; Li and Abe 1998; Pereira et al. 1993). In the following, we first address the latent class model, and then define the PLSA as its instance.

Latent class modeling is a method of obtaining reliable estimates for probabilities of events that have not occurred in training data. It smoothes raw frequency of observed events and assigns non-zero probabilities to unseen events by interpolation. This smoothing is done by hypothesizing existence of latent variables behind observed variables and assuming a conditional independence structure among the latent variables and the observed variables. In this dissertation, the latent variables in the models are nominal variables and can be interpreted as classes of observed variables or hidden causes of events.

A general form of latent class models is:

$$P(\mathbf{x}) \quad = \quad \sum_{\mathbf{c} \in C} P(\mathbf{x}, \mathbf{c}), \tag{7.1}$$

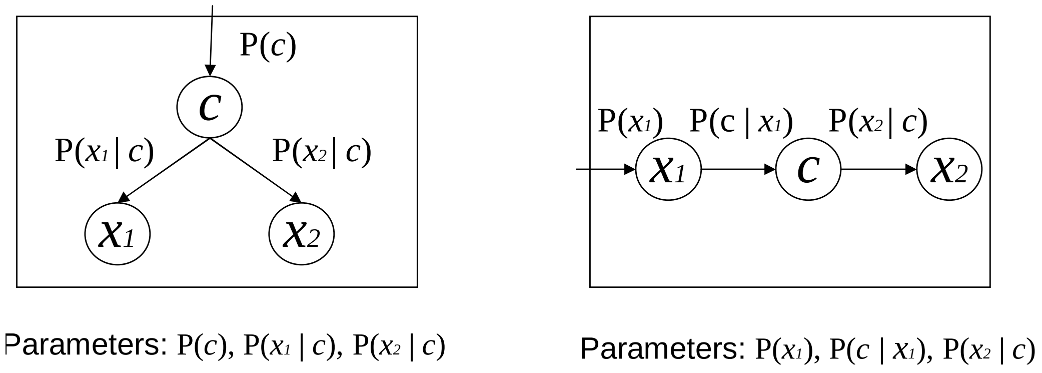
where $\mathbf{x}$ denotes a vector of observed variables, $\mathbf{c}$ denotes a vector of latent variables, and C denotes the set of all possible states of $\mathbf{c}$.

Typical examples of a conditional independence structures assumed in $P(\mathbf{x}, \mathbf{c})$ are depicted in the left-hand side and the right-hand side of Figure 7.1. In these examples, $\mathbf{x}$ has two components x_1 and x_2, and c is a scalar variable. Both examples assume conditional independence of x_1 and x_2 given c, but parameterized differently. In the first example, the probability of $\mathbf{x} = (x_1, x_2)$ is decomposed as:

$$P(x_1, x_2) \quad = \quad \sum_{c \in C} P(c)P(x_1|c)P(x_2|c) \tag{7.2}$$

and the model has three types of parameters: $P(x_1|c)$, $P(x_2|c)$ and $P(c)$. In the second example, $P(x_1, x_2)$ is decomposed as:

$$P(x_1, x_2) \quad = \quad P(x_1) \sum_{c \in C} P(c|x_1)P(x_2|c) \tag{7.3}$$



P(c)
c
P(x1 | c)
P(x2 | c)
x1
x2
Parameters: P(c), P(x1 | c), P(x2 | c)
P(x1)
P(c | x1)
P(x2 | c)
x1
c
x2
Parameters: P(x1), P(c | x1), P(x2 | c)

Figure 7.1: Typical examples of conditional independence structures assumed in $P(\mathbf{x}|c)$

and the model has three types of parameters: $P(x_1)$, $P(c|x_1)$, $P(x_2|c)$.

To see why such decompositions of $P(x_1, x_2)$ yield smoothing effects, let us count the number of free parameters in the two models. Letting $|X_1|$ and $|X_2|$ denote the numbers of possible values of x_1 and x_2 respectively, the number of free parameters in the first model, k_1, is:

$$k_1 = (|C| - 1) + (|X_1| - 1)|C| + (|X_2| - 1)|C|, \tag{7.4}$$

where the first, second, and third terms are the number for $P(c)$, $P(x_1|c)$, and $P(x_2|c)$. On the other hand, the number of free parameters in the second model, k_2, is:

$$k_2 = (|X_1| - 1) + |X_1|(|C| - 1) + (|X_2| - 1)|C|. \tag{7.5}$$

When the number of classes in the model, $|C|$, is set to much smaller value than $|X_1|$ and $|X_2|$, k_1 and k_2 become smaller than $k = |X_1||X_2| - 1$, which is the number of free parameters in the simple tabulation model, i.e., the case where all $P(x_1, x_2)$ are directly estimated from observed raw frequency. By fitting the model with those fewer parameters, probability mass for observed pair of (x_1, x_2) is decreased and distributed to unseen co-occurrences.

As can be seen in the above examples, latent class variables can be interpreted as representing hidden grouping of $\mathbf{x}$ (in the left-hand side of Figure 7.1) or x_2 (in the right-hand side of Figure 7.1), from which the observed variables $\mathbf{x}$ or x_2 are selected with some probability. An estimation algorithm for those models can be considered as a clustering algorithm that has a probabilistic interpretation.

The PLSA model is defined as an instance of the latent class models, which is a joint probability model of k variables $x_1, \ldots, x_k$ and takes the form of:

$$P(x_1, \ldots, x_k) \;=\; \sum_{c \in C} P(c) \prod_{i=1}^{k} P(x_i \,|\, c). \qquad \textit{symmetric parameterization} \tag{7.6}$$

In this dissertation, we concern the case where k = 2, that is, the PLSA model for two observed variables x and y. That PLSA model with two observed variables can be parameterized different way:

$$P(x, y) \;=\; \sum_{c \in C} P(c) P(x \,|\, c) P(y \,|\, c). \qquad \textit{symmetric parameterization}. \tag{7.7}$$

$$P(x, y) \;=\; P(x) \sum_{c \in C} P(c \,|\, x) P(y \,|\, c). \qquad \textit{asymmetric parameterization} \tag{7.8}$$

This model can be seen as a kind of soft clustering of x according to its co-occurrences. That is, there are $|C|$ latent classes and x is probabilistically assigned to $\forall c \in C$.

When we use the latent class models, there is an issue on how we determine the number of the latent variables. As we have seen in the paragraph including the equations 7.4 and 7.5, the number of the latent variables has an important effect on the performance of smoothing. We investigate the effect of the number of latent classes to our modeling of co-occurrences between words and SCFs in the experiments.

7.1.2 EM Estimation for the Probabilistic Latent Semantic Analysis

The Expectation Maximization (EM) Algorithm (Dempster et al. 1997) is an iterative optimization algorithm used for maximum likelihood estimation of parameters in latent variable models. The EM algorithm works especially efficiently in estimation of latent class models of word co-occurrence, where many parameters must be estimated from sparse training data.

In this section, we assume a PLSA model with k = 2 by asymmetric parameterization:

$$P(x, y) \;=\; P(x) \sum_{c \in C} P(c \,|\, x) P(y \,|\, c). \qquad \textit{asymmetric parameterization} \tag{7.9}$$

where x, y are observed variables in events and c is a latent class variable, which takes a value in a finite set C. We also assume that $P(x, c, y)$ is parameterized by a vector of parameters θ. Let $P_{\theta_t}(x, c, y)$, $P_{\theta_t}(x, y)$, etc. denote particular values of these probabilities with a particular parameter value θ_t.

Under the maximum likelihood estimation (MLE) criterion, we want the value of θ that maximizes the marginal (log-)likelihood function:

$$\begin{aligned} L(\theta) &= \sum_{(x_i,y_i)\in S} \log P_\theta(x_i, y_i) \\ &= \sum_{(x_i,y_i)\in S} \log P_\theta(x_i) \sum_{c\in C} P_\theta(c \,|\, x_i) P_\theta(y_i \,|\, c), \end{aligned} \tag{7.10}$$

where S is the (multi-)set of training data, $S = \{(x_1, y_1), (x_2, y_2), \ldots\}$.

The EM algorithm finds a local optimum of $L(\theta)$ by gradually improving the parameter value of θ. We consider the difference of log-likelihood when we update θ to $\overline{\theta}$:

$$\begin{aligned} L(\overline{\theta}) - L(\theta) &= \sum_{(x_i,y_i)\in S} \log P_{\overline{\theta}}(x_i, y_i) - \sum_{(x_i,y_i)\in S} \log P_\theta(x_i, y_i) \\ &= \sum_{(x_i,y_i)\in S} \log \frac{P_{\overline{\theta}}(x_i, y_i)}{P_\theta(x_i, y_i)} \\ &= \sum_{(x_i,y_i)\in S} \sum_{c\in C} P_\theta(c \,|\, x_i, y_i) \log \frac{P_{\overline{\theta}}(x_i, y_i)}{P_\theta(x_i, y_i)} \\ &= \sum_{(x_i,y_i)\in S} \sum_{c\in C} P_\theta(c \,|\, x_i, y_i) \log \left[\frac{P_{\overline{\theta}}(x_i, y_i, c)}{P_\theta(x_i, y_i, c)} \frac{P_\theta(c \,|\, x_i, y_i)}{P_{\overline{\theta}}(c \,|\, x_i, y_i)} \right] \\ &= \sum_{(x_i,y_i)\in S} \sum_{c\in C} P_\theta(c \,|\, x_i, y_i) \log \frac{P_{\overline{\theta}}(x_i, y_i, c)}{P_\theta(x_i, y_i, c)} \\ &\quad + \sum_{(x_i,y_i)\in S} \sum_{c\in C} P_\theta(c \,|\, x_i, y_i) \log \frac{P_\theta(c \,|\, x_i, y_i)}{P_{\overline{\theta}}(c \,|\, x_i, y_i)}. \end{aligned} \tag{7.11}$$

Because we can derive

$$\sum_{c\in C} P_\theta(c \,|\, x_i, y_i) \log \frac{P_\theta(c \,|\, x_i, y_i)}{P_{\overline{\theta}}(c \,|\, x_i, y_i)} \geq 0$$

from Jensen's inequality, we can obtain an inequality

$$\begin{aligned} L(\overline{\theta}) - L(\theta) &\geq \sum_{(x_i,y_i)\in S} \sum_{c\in C} P_\theta(c \,|\, x_i, y_i) \log \frac{P_{\overline{\theta}}(x_i, y_i, c)}{P_\theta(x_i, y_i, c)} \\ &\geq Q(\theta, \overline{\theta}) - Q(\theta, \theta), \end{aligned} \tag{7.12}$$

where

$$Q(\theta,\overline{\theta}) = \sum_{(x_i,y_i)\in S} \sum_{c\in C} P_\theta(c \,|\, x_i,y_i) \log P_{\overline{\theta}}(x_i,y_i,c)$$

$$= \sum_{(x_i,y_i)\in S} \sum_{c\in C} P_\theta(c \,|\, x_i,y_i) \log \left[P_{\overline{\theta}}(x_i) \sum_{c\in C} P_{\overline{\theta}}(c \,|\, x_i) P_{\overline{\theta}}(y_i \,|\, c) \right]. \quad (7.13)$$

By the inequality in 7.12, we can monotonically increase the log-likelihood by updating the parameter value of θ to $\overline{\theta}$ that maximizes $Q(\theta,\overline{\theta})$. In short, starting from some initial value $\theta^{(1)}$, at t-th iteration, the EM algorithm applies the following two steps to $\theta^{(t)}$ and yields a new value $\theta^{(t+1)}$:

E-step : Compute $Q(\theta,\theta^{(t)}) = \sum_{(x_i,y_i)\in C} \sum_{c\in C} P_{\theta^{(t)}}(c \,|\, x_i,y_i) \log \left[P_\theta(x_i) \sum_{c\in C} P_\theta(c \,|\, x_i) P_\theta(y_i \,|\, c) \right].$

M-step : Set $\theta^{(t+1)}$ to that θ which maximizes $Q(\theta,\theta^{(t)})$.

7.2 Probabilistic Latent Semantic Analysis for Modeling Verb Subcategorization

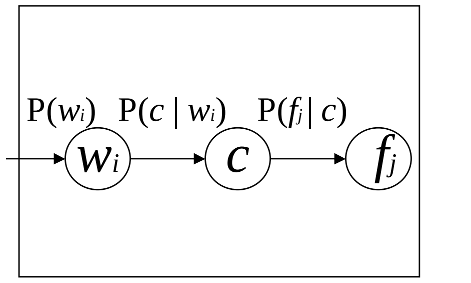
In this section, we apply the probabilistic latent semantic analysis to co-occurrence between words and SCFs.

7.2.1 Model Definition

We propose the probabilistic latent semantic analysis for modeling co-occurrence between words and SCFs where the latent variables are *semantic classes*. In this dissertation, we define semantic classes as word classes whose members have the same SCF distribution. This semantic class is analogous to the one defined by Levin, who defines verb semantic classes according to the verbs' SCF alternation behavior. The PLSA model is suitable for this modeling because they perform a kind of soft clustering, which can naturally treat highly polysemic nature of verbs.

Let a conditional probability that a word $w_i \in W$ appear as a member of a semantic class $c \in C$ be $P(c \,|\, w_i)$, and each semantic class $c \in C$ takes an SCF $f_j \in F$ with a conditional probability $P(f_j \,|\, c)$. Figure 7.2 shows this SCF modeling. When we assume that a word w_i occurs with a probability $P(w_i)$, a probability $P(w_i, f_j)$, with which the word $w_i \in W$ occurs with $f_j \in F$ is:

$$P(w_i, f_j) = P(w_i) \sum_{c\in C} P(f_j \,|\, c) P(c \,|\, w_i). \quad (7.14)$$



Parameters: P(w_i), P(c | w_i), P(f_j | c)

Figure 7.2: Probabilistic Latent Semantic Analysis of co-occurrence between words and SCFs

7.2.2 EM Estimation of the Probabilistic Latent Semantic Model for SCFs

In this section, we derive the EM estimation of our PLSA model for co-occurrences between words and SCFs. As described in the previous section, our model is formalized as follows:

$$P(w_i, f_j) \quad = \quad P_\theta(w_i) \sum_{c \in C} P_\theta(c \,|\, w_i) P_\theta(f_j \,|\, c), \tag{7.15}$$

where $\sum_{c \in C} P_\theta(c \,|\, w_i) = 1$, and $\sum_{j \in F} P_\theta(f_j \,|\, c) = 1$. We estimate $P(c \,|\, w_i)$ and $P(f_j \,|\, c)$ from observed data by applying the EM estimation to $P_\theta(w_i, f_j, c) = P_\theta(w_i) P_\theta(c \,|\, w_i) P_\theta(f_j \,|\, c)$. By substituting $x_i = w_i$, $y_i = f_j$, to Equation 7.13, the Q-function of the EM estimation is defined as follows:

E-step:

$$\begin{aligned} Q(\theta, \overline{\theta}) \quad = \quad & \sum_{(w_i, f_j) \in W \times F} \sum_{c \in C} \\ & n(w_i, f_j) P_\theta(c \,|\, w_i, f_j) \log[P_{\overline{\theta}}(w_i) \sum_{c \in C} P_{\overline{\theta}}(f_j \,|\, c) P_{\overline{\theta}}(c \,|\, w_i)], \end{aligned} \tag{7.16}$$

where

$$P_\theta(c | w_i, f_j) \quad = \quad \frac{P_\theta(f_j \,|\, c) \cdot P_\theta(c \,|\, w_i)}{\sum_{c \in C} P_\theta(f_j \,|\, c) \cdot P_\theta(c \,|\, w_i)}. \tag{7.17}$$

$n(w_i, f_j)$ is the observed frequency of a co-occurrence between w_i and f_j, while $n(w_i)$ is the observed frequency of a word w_i.

We then have $P_{\overline{\theta}}(f_j \,|\, c)$, $P_{\overline{\theta}}(c \,|\, w_i)$, and $P_{\overline{\theta}}(w_i)$, which maximize $Q(\theta, \overline{\theta})$ by the Lagrange multiplier method, as:

M-step:

$$P_{\overline{\theta}}(c \,|\, w_i) = \frac{\sum_{f_j \in F} n(w_i, f_j) P_\theta(c|w_i, f_j)}{n(w_i)} \tag{7.18}$$

$$P_{\overline{\theta}}(f_j \,|\, c) = \frac{\sum_{w_i \in W} n(w_i, f_j) P_\theta(c|w_i, f_j)}{\sum_{w_i \in W} \sum_{f_j \in F} n(w_i, f_j) P_\theta(c|w_i, f_j)} \tag{7.19}$$

$$P_{\overline{\theta}}(w_i) = \frac{n(w_i)}{\sum_{w_i \in W} n(w_i)} \tag{7.20}$$

7.3 Smoothing Method for SCF Distributions

Finally, we describe a method of obtaining accurate estimates for co-occurrence probabilities of words and SCFs by using the PLSA model defined in the previous section. We interpolate three different models that predict the probability. The first model is the raw observed probability defined as follows:

$$P_{raw}(f_j|w_i) = \frac{n(w_i, f_j)}{\sum_{f_j \in F} n(w_i, f_j)} \tag{7.21}$$

The second model is a smoothed probability based on the PLSA model, which is calculated by the following equation:

$$P_{PLSA}(f_j|w_i) = \sum_{c \in C} P(c|w_i) P(f_j|c), \tag{7.22}$$

where $P(f_j|c)$ and $P(c|w_i)$ are ones estimated under the PLSA model $\mathcal{M}_{\mathcal{PLSA}}$. The last model is $P_{raw}(f_j)$, which is the maximum likelihood estimation of $P(f_j)$, defined as follows:

$$P_{raw}(f_j) = \frac{\sum_{w \in W} n(w, f_j)}{\sum_{w \in W, f_j \in F} n(w, f_j)} \tag{7.23}$$

We combine these three models by linear interpolation:

$$P(f_j|w_i) = \lambda_1 P_{raw}(f_j|w_i) + \lambda_2 P_{PLSA}(f_j|w_i) + \lambda_3 P_{raw}(f_j) \tag{7.24}$$

where λ_1, λ_2, and λ_3 are *weights* of each probabilistic model, which satisfy the constraint $\lambda_1 + \lambda_2 + \lambda_3 = 1$. We can estimate λ_i by an EM algorithm using held-out data.

Table 7.1: The specification of SCFs for HPSG acquired from WSJ Sections 2-21 and their subsets

Corpus	**2**	**2-4**	**2-6**	**2-8**	**2-10**	**2-21**
# of SCF types	78	115	139	152	169	253
# of words	1,020	1,641	2,072	2,285	2,577	3,586
# of SCFs	5,862	16,667	28,220	35,644	47,346	112,974
# SCFs types per words	1.46	1.58	1.64	1.68	1.71	1.84

The above interpolated model can provide co-occurrence probabilities for words and SCFs that are seen in the training data. However, we also concern a probabilistic model for words unseen in the training data. In this study we simply calculate a co-occurrence probability for words unseen in the training data by:

$$p(f_j|w_i) \quad = \quad P_{raw}(f_j) \tag{7.25}$$

In summary, we calculate a co-occurrence probability of a word w_i and an SCF type f_j is described as:

$$P(f_j|w_i) \quad = \quad \begin{cases} \lambda_1 P_{raw}(f_j|w_i) + \lambda_2 P_{PLSA}(f_j|w_i) + \lambda_3 P_{raw}(f_j) & (w_i \in the\ training\ data) \\ P_{raw}(f_j) & otherwise \end{cases} \tag{7.26}$$

In the following experiments, we compare the above smoothing method with a more simple smoothing method, which estimates co-occurrence probabilities from $P_{raw}(f_j|w_i)$ and $P_{raw}(f_j)$ as follows:

$$P(f_j|w_i) = \begin{cases} \lambda_1 P_{raw}(f_j|w_i) + \lambda_2 P_{raw}(f_j) & (w_i \in the\ training\ data) \\ P_{raw}(f_j) & otherwise \end{cases} \tag{7.27}$$

7.4 Experiments on Smoothing SCF Lexicon Acquired from Annotated Corpora

In this section we investigate the effect of our smoothing method on SCFs acquired for lexicalized grammars from annotated corpora. Our objective for smoothing is to achieve accurate estimates of co-occurrence probabilities between words and SCFs from as a small amount of annotated corpora as possible.

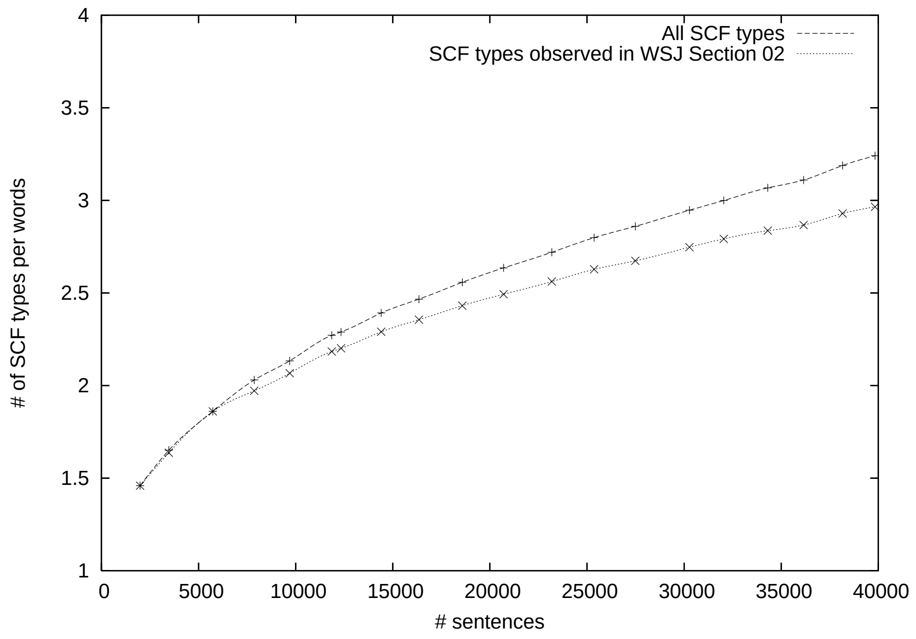


Figure 7.3: The average number of SCF types assigned to words in WSJ Section 02

We start by extracting $\langle SCF, word \rangle$ pairs from Sections 2-21 of the Wall Street Journal (WSJ) portion of the Penn Treebank and their subset sections by a method introduced in Section 5.2.2 (Miyao et al. 2004; Nakanishi et al. 2004). Table 7.1 shows the specification of the acquired SCFs. When we acquire SCFs from a larger amount of annotated corpora, the SCF types rapidly increase. However, the average number of SCF types per words increases rather mildly. However, when we calculate the average number of SCF types for the words observed in WSJ Section 2, the situation is somewhat different. Figure 7.3 shows the average number of SCF types assigned to the words in WSJ Section 2. The average number of SCF types for only words in WSJ Section 2 increases more rapidly than that for all the words. This implies that frequent verbs tend to take the larger number of SCF types than infrequent verbs. The acquired SCF lexicons indeed suffer from the problem on unknown associations between known subcategorization frames and known words.

We then applied our smoothing method to these acquired SCFs. We constructed several PLSA models for each set of SCFs acquired from corpora, by ranging the number of latent variables from 160 to 1,280. We hereafter refer to the PLSA models for SCFs acquired from WSJ Sections x as

Table 7.2: Test-set perplexity of $P(f_j|w_i)$ against the test SCFs acquired from WSJ Section 24 for the SCF types are observed in WSJ Section 2

Corpus	# classes	**2**	**2-4**	**2-6**	**2-8**	**2-10**	**2-21**
unknown		10.809	10.770	10.768	10.749	10.755	10.739
naive		4.030	3.557	3.351	3.298	3.250	3.116
PLSA	160	3.918	3.433	3.237	3.181	3.128	3.025
	320	3.823	3.399	*3.199*	3.177	3.105	3.023
	640	3.848	3.381	3.202	3.155	3.121	3.041
	1,280	*3.820*	*3.351*	3.206	*3.147*	*3.101*	**3.009**

P_{PLSA_x}. We then evaluate the effect of our smoothing method on the acquired SCF distributions. We combined the above $P_{PLSA_x}(f_j|w_i)$ with $P_{raw}(f_j|w_i)$ and $P_{raw}(f_j)$ by linear interpolation. The weights are estimated by the EM algorithm using SCFs acquired from WSJ Section 22 as held-out data. In order to evaluate accuracy of estimated co-occurrence probability, we employ *test-set perplexity*, which is defined by:

$$PP = \exp\left(-\frac{\sum_{(w_i,f_j)\in S_t} n(w_i,f_j)\log_e P(w_i,f_j)}{\sum_{w_i\in S_t} n(w_i)}\right) \tag{7.28}$$

where S_t is a test data. This measure indicates the complexity of the task that determines an SCF type for a given verb.

Table 7.2 shows a test-set perplexity against part of SCFs acquired from WSJ Section 24, which are for (SCF) types observed in WSJ Section 2. In the table, *unknown* refers to a model that uses only raw frequency for SCFs, $P_{raw}(f_j)$, which is a model used for unknown words in the other models, as shown in Equations 7.26 and 7.27. This indicates the difficulty of this task. The model *naive* refers to the interpolated models that use only raw frequency, which is defined in Equation 7.27, while *PLSA* refers to the interpolated models that use the PLSA model, which is defined in 7.26. We can clearly observe an improvement by models that use the *PLSA* model. All *PLSA* models except a model using WSJ Sections 02-21 as training data achieved a better test-set perplexity compared to *naive* when they are estimated using the same size of corpora. Furthermore, *PLSA* models achieved better test-set perplexity than that obtained by *naive* with around twice as much training data (cf. *naive* with WSJ Section 2-21 v.s. *PLSA* with WSJ Section Section 2-10). It is also noteworthy that we can achieve an improvement by setting the number of latent

Table 7.3: Test-set perplexity of $P(f_j|w_i)$ against the test SCFs acquired from WSJ Section 24

Corpus	# classes	**2**	**2-4**	**2-6**	**2-8**	**2-10**	**2-21**
unknown		10.809	10.995	11.182	11.161	11.218	11.354
naive		4.030	3.642	3.476	3.420	3.389	3.275
PLSA	160	3.918	3.507	3.314	3.289	3.245	3.168
	320	3.823	3.471	**3.306**	3.282	**3.223**	3.166
	640	3.848	3.456	3.312	3.259	3.237	**3.153**
	1,280	**3.820**	**3.428**	3.315	**3.249**	3.224	**3.153**

classes larger. Comparing *PLSA* models whose number of latent variables is 160 and *PLSA* models whose number of latent variables is 320, all the models with the larger number of latent variables outperformed the models with the smaller number of latent variables. However, when we use the larger number of latent variables more than 320, some models decreases its test-set perplexity. This implies that for these training data, there is an appropriate number around 320 of classes to achieve the test-set perplexity. Since the number of latent variables in the latent class models controls the degree of generalization of the models, we could use larger number of latent variables to increase the test-set perplexity if we have a larger amount of annotated data.

Table 7.2 shows a test-set perplexity against part of SCFs acquired from WSJ Section 24, which are for (SCF) types observed in the training data. Even when we use the all SCF types in the training data for testing, we achieve more accurate estimates by the interpolated model using the PLSA model.

7.5 Related Work

Our PLSA model can be regarded as clustering of verbs according to their SCF distributions. In this section, we introduce methods for clustering verbs' SCF distributions acquired from raw corpora.

There are two studies that perform clustering of verbs' SCF distributions (Schulte im Walde and Brew 2002; Korhonen et al. 2003). Their objective is not to obtain accurate estimates of co-occurrence probabilities between words and SCFs, but to induce verb semantic classes by clustering verbs according to their SCF distributions acquired from raw corpora. They first represent a verb SCF distribution by an n-dimensional vector for each verb. Each element in the SCF distribution represents a co-occurrence probability between the verb and each SCF type. Schulte im Walde and Brew (Schulte im Walde and Brew 2002) used the k-Means (Forgy 1965) algorithm to cluster SCF distributions for monosemous German verbs, while Korhonen et al. (Korhonen et al. 2003) conducted clustering of SCF distributions for English verb using other clustering methods and then investigated the effect of polysemic verbs on clustering.

Although these studies demonstrated that there is a certain classification of verbs according to their SCF distributions, they did not focus on the improvement of the quality of the SCF lexicon. On the other hand, our smoothing method given in this chapter aims at obtaining accurate estimates of co-occurrence probabilities between words and SCFs, and takes SCF distributions acquired from annotated corpora as inputs. Although our smoothing method does not aim at obtaining semantic classes, we can employ our smoothing method to induce semantic classes by assuming latent classes of the PLSA model as semantic classes. For example, when we initialize the latent classes to some fine-grained verb semantic classes such as ones provided in WordNet, we will be able to obtain semantic classes purely based on syntactic behaviors by merging latent semantic classes according to similarity between their SCF distributions.

7.6 Chapter Summary

We presented a method of constructing a probabilistic lexicon that consists of accurate estimates of co-occurrence probabilities between words and SCFs, for the other type of lexical resource that include associations between words and SCFs along with their co-occurrence probability. We assumed that there are latent classes that govern the SCF alternation behaviors, and perform smoothing of raw observed frequency of co-occurrences between words and SCFs by modeling the co-occurrences with the PLSA model. We also proposed the interpolated model that uses the PLSA model and raw frequency counts.

We applied our smoothing method to SCFs for an HPSG grammar acquired from the Penn Treebank. By varying the number of latent classes in the PLSA model, the proposed interpolation model that uses both the PLSA model and the observed raw frequency achieved a comparatively higher test-set perplexity against the linear interpolation model that only uses the observed raw frequency. By using accurate estimates obtained for co-occurrence between words and SCFs, we can construct an appropriate set of lexicons for the grammar from the obtained estimates, because they defines a co-occurrence probability of possible associations between words and SCFs, which can be used for thresholding of SCFs.

Conclusions

We have proposed two distinctive approaches to accelerating the development of the lexicalized grammar-based NLP by solving the two bottlenecks in applying lexicalized grammar-based NLP to practical applications. The first problem is the difficulty in developing static components such as grammar resources and parsing techniques for the lexicalized grammar formalisms. The second problem is the difficulty in developing dynamic lexical resources, which are essential when we apply lexicalized grammars to a new target domain.

Methodology for Collaboration between LTAG and HPSG

The first part of this dissertation concerns a methodology for developing static components such as grammar resources and parsing techniques for lexicalized grammar formalisms. We described a novel approach to collaborative development of the static components among the lexicalized formalisms, in particular Lexicalized Tree Adjoining Grammar (LTAG) and Head-Driven Phrase Structure Grammar (HPSG), by a method of grammar conversion from LTAG to HPSG-style grammar.

Method for Resource Sharing and Parsing Comparison by Grammar Conversion

We first proposed an algorithm for the conversion of grammars from an arbitrary FB-LTAG grammar into a strongly equivalent HPSG-style grammar. Our algorithm first converts LTAG elementary trees into a set of tree structures that have only one word and can be decomposed into immediate constituency. We then convert the tree structures into HPSG feature structures by encoding the branching structures in stacks. A set of pre-determined rules manipulate the stack to emulate substitution and adjunction. The definition of strong equivalence and a formal proof on the strong equivalence between the original and obtained grammars are also provided. The nature of strong equivalence guaranteed by the grammar conversion enables us to obtain the parsing results of an

LTAG grammar from the parsing results of the HPSG-style grammar obtained by the conversion. The obtained grammar successfully abstracts away surface differences in computation devices that underlie the two formalisms. Our method thus enables the sharing of LTAG resources with the HPSG community, the application of HPSG technologies to LTAG grammars, and the clarification of the differences between linguistic analysis according to the two grammar formalisms.

We next demonstrated applications of our grammar conversion to bridge the LTAG and HPSG formalisms through experiments on grammar resource sharing and parsing comparison. First, we showed that we can obtain a large-scale HPSG-style grammar that is compatible with HPSG systems, by converting from a large-scale LTAG grammar. The XTAG English grammar, a large-scale LTAG grammar, was successfully converted into an HPSG-style grammar. We investigated the specification of the obtained HPSG-style grammar, and then discussed different ways of encodings of grammatical constructions in the both framework by comparing accounts of several linguistic phenomena by the obtained HPSG-style grammar and HPSG. Some of the accounts given by the obtained HPSG-style grammar are analogous to the one employed in the HPSG formalism, especially in the way to specify a syntactic structure taken by a node to be subcategorized, although they are still different in some important respects including the treatment of some type of unbounded dependency.

Second, we empirically compared two pairs of LTAG and HPSG parsers based on dynamic programming and CFG filtering. Experiments comparing parsers using dynamic programming showed that the different implementations of the factoring scheme caused a difference in the empirical time complexity of the parsers. This result suggests that for LTAG parsing we can achieve a drastic speed-up by merging two states whose elementary trees have the same unprocessed parts. Another experiment comparing parsers with CFG filtering showed that the CF approximation of HPSG produced a more effective filter than that of LTAG. This result also suggests that we can obtain an effective CFG filter for LTAG by approximating the LTAG with a CFG by applying substitution and adjunction along tree traversal and regarding unprocessed parts of generated tree structures as nonterminals of the CFG.

Future work for Collaboration between the LTAG and HPSG formalisms

We are going to pursue collaboration among lexicalized formalisms in order to develop generic NLP components within the lexicalized formalisms. Although we showed comparison of parsers based on theoretical parsing techniques in this paper, our approach is extendable to other types of NLP technologies such as disambiguation module or grammar development environment. For example, although parsers based on current statistical parsing technologies in the lexicalized formalisms achieved similar precision and recall for the identification task of predicate-argument relations, we have difficulty to work out the reason for the improvement. This is because statistical parsers involve both the statistical model and the grammar altogether in the model, which both contribute to the performance. If we could make use of strongly equivalent grammars as we have done in the comparison of theoretical parsers, comparison of purely statistical model for different formalisms would be possible. We are also going to apply our methodology to other pairs of lexicalized formalisms by proposing grammar conversion between them.

We will investigate the equivalence between a particular LTAG grammar and an HPSG grammar (Pollard and Sag 1994) by linguistically-motivated approaches. Miller (Miller 1999) discusses strong equivalence between grammars in different formalisms. He defines the equivalence of structural descriptions given by two grammars according to the equivalence of linguistic properties expressed by the structural descriptions. The structural descriptions are mapped by Interpretation Functions into several Interpretation Domains, which express linguistically relevant properties of a formalism and are defined by users of the formalism. Examples of Interpretation Domains are dependency, constituency and ordering. Following his approach, we can discuss the equivalence between an LTAG grammar and an HPSG grammar. Another candidate approach to investigate the equivalence of grammars is to express both LTAG and HPSG grammars in Abstract Categorial Grammar (ACG) (de Groote 2001), a grammatical framework in which other existing grammatical models may be encoded, and to discuss the equivalence of the grammars within the ACG framework. However, since the generative capacity of the typed feature structures is beyond context-sensitive formalisms, it is doubtful whether ACG can represent grammars represented in the typed feature structures, *e.g.*, HPSG. We will thus take advantages of the HPSG computational architecture defined in this dissertation in order to capture the HPSG formalism in ACG.

Methods for Acquiring Lexical Resources Acquired from Corpora

The second part of this dissertation concerns two methods of acquiring lexical resources from corpora in order to compensate for the lack of recall in the two types of existing lexical resources available for lexicalized grammars.

Methods for Filtering Reliable Lexical Resources Acquired from Raw Corpora

We presented a method of improving the quality of SCFs acquired from corpora in order to augment one type of lexical resource that include only association between words and SCFs. We first obtain verb clusters whose elements share the same set of SCFs. We then maps observed SCF distributions to SCF confidence vectors by the Bayesian statistics approach, in order to make use of discrete co-occurrence tendencies among SCFs taken by each word that already exists in the target lexical resource. A new clustering algorithm for SCF confidence vectors is proposed as well. We have also proposed a filtering method that eliminates SCFs acquired in error by using simple cut-off based on the obtained clusters.

We applied our filtering method to SCFs acquired from corpora using lexicons of the XTAG English grammar and the LinGO ERG, respectively. Experimental results revealed that we can increase the recall of the resulting SCFs when we use Bayesian estimation instead of maximum-likelihood estimation to calculate the probability that the word has each SCF type. The results have also shown that we can eliminate implausible SCFs, preserving more reliable SCFs that exist in the target lexicon, by virtue of clustering results that are derived from both confidence vectors in the acquired SCFs and the existing SCFs in the target grammar. This is also first ever evaluation of augmenting the XTAG English grammar with SCFs acquired from corpora.

Methods for Smoothing Lexical Resources Acquired from Annotated Corpora

We next presented a method of obtaining accurate estimates of co-occurrence probabilities between words and SCFs, for the other type of lexical resource that include associations between words and SCFs along with their co-occurrence probability. We assumed that there are latent classes that govern the SCF alternation behaviors, and perform smoothing of raw observed frequency of co-occurrences between words and SCFs by modeling the co-occurrences with the PLSA model. We also proposed the interpolated model that uses the PLSA model and raw frequency counts.

We applied our smoothing method to SCFs for an HPSG grammar acquired from the Penn Treebank. By varying the number of latent classes in the PLSA model, the proposed interpolation model that uses both the PLSA model and the observed raw frequency achieved a comparatively higher test-set perplexity against the linear interpolation model that only uses the observed raw frequency. By using accurate estimates obtained for co-occurrence between words and SCFs, we can construct an appropriate set of lexicons for the grammar from the obtained estimates, because they define a co-occurrence probability of possible associations between words and SCFs, which can be used for thresholding of SCFs.

Future work for Acquisition of Lexical Resources from Corpora

We are going to employ our filtering method to solve a problem called “*missing of known SCF types for known words*.” As mentioned in Section 6.1, we can simply handle this problem by using different SCF confidence vectors for words that are included in both the target lexicon and the acquired lexicon. We are also going to evaluate the quality of the SCFs selected from those acquired from raw corpora by manual analysis and by their impact on parsing performance. We will also investigate other clustering methods such as hierarchical clustering, and use other information for clustering such as semantic preference of arguments of SCFs to have more accurate clusters. As we introduced in Section 5.3, a classification of verbs according to their SCF types are closely related to verb semantic classes. Although our current classification does not focus on the semantic classification of verbs, we are going to employ a method of obtaining verb semantic classes by clustering of verb SCF distributions to assign semantic constraints to unknown words. Because our methods can determine the reliable set of SCFs for unknown words, we will able to obtain more accurate semantic classes compared to the methods using noisy SCF distributions.

We will investigate the parsing performance using a probabilistic lexicon obtained by our PLSA-based smoothing, in order to evaluate the quality of the lexicon and the effect of thresholding to select a reliable set of lexicons. We will also investigate whether a lexicon estimated from parsing results of large amounts of raw corpora provides more accurate estimates of co-occurrence probabilities between words and SCFs than that obtained from a small amount of annotated corpora. We will also employ our PLSA modeling of co-occurrences between words and SCFs to acquire a semantic classification of verbs, because our PLSA modeling is suitable to handle polysemic nature of verbs. We modeled co-occurrence between words and SCFs using semantically-motivated classes defined in WordNet (Fellbaum 1998) as initial latent classes, generalized the model by a variant of Li and Abe's method (Li and Abe 1998), and then achieved the improvement of the quality of the probabilistic lexicon in terms of the test-set perplexity. Although the improvement was lower than the results shown in Section 7.4, the obtained latent classes would be more meaningful to be examined.

Bibliography

Abeillé, A. (1993). *Les nouvelles syntaxes: grammaires d'unification et analyse du français*. Armanda Colin. in French. (cited in page 3)

Abeillé, A. and M.-H. Candito (2000). FTAG: A Lexicalized Tree Adjoining Grammar for French. In A. Abeillé and O. Rambow (Eds.), *Tree Adjoining Grammars: formalisms, linguistic analysis and processing*, pp. 305–329. CSLI publications. (cited in pages 3, 5, 22, 87)

Abeillé, A., Y. Schabes, and A. Joshi (1990). Using Lexicalized TAGs for machine translation. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING 1990)*, Helsinki, Finland, pp. 1–6. (cited in page 21)

Baeza-Yates, R. and B. Ribeiro-Neto (1999). *Modern Information Retrieval*. Addison. (cited in page 1)

Baldwin, B., C. Dorran, J. Reynar, M. Niv, B. Srinivas, and M. Wasson (1997). EAGLE: An Extensible Architecture for General Linguistic Engineering. In *Proceedings of the fifth RIAO Conference Computer-Assisted Information Searching on Internet (RIAO)*, Montreal, Quebec, Canada, pp. 271–283. (cited in page 21)

Barthélemy, F., P. Boullier, P. Deschamp, and Éric Villemonte de la Clergerie (2001). Guided parsing of range concatenation languages. In *Proceedings of the 39st Annual Meeting of the Association for Computational Linguistics (ACL 2001)*, Toulouse, France, pp. 42–49. (cited in pages 3, 22, 75)

Becker, T. (1994). Patterns in metarules. In *Proceedings of the third International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+3)*, Paris, France, pp. 9–11. (cited in page 87)

Becker, T. (2000). Patterns in metarules for TAG. In A. Abeillé and O. Rambow (Eds.), *Tree Adjoining Grammars: formalisms, linguistic analysis and processing*, pp. 331–342. CSLI publications. (cited in page 87)

Becker, T. and P. Lopez (2000). Adapting HPSG-to-TAG compilation to wide-coverage grammars. In *Proceedings of the fifth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+5)*, Paris, France, pp. 47–54. (cited in pages 3, 74, 75)

Bender, E. M., D. Flickinger, and S. Oepen (2002). The grammar matrix: An open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. In J. Carroll, N. Oostdijk, and R. Sutcliffe (Eds.), *Proceedings of the Workshop on Grammar Engineering and Evaluation at the 19th International Conference on Computational Linguistics (COLING 2002)*, Taipei, Taiwan, pp. 8–14. (cited in page 22)

Bischof, H., A. Leonardis, and A. Selb (1999). MDL principle for robust vector quantization. *Pattern Analysis and Applications 2*(1), 59–72. (cited in page 101)

Bod, R. (2001). What is the minimal set of fragments that achieves maximal parse accuracy? In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL 2001)*, Toulouse, France, pp. 66–73. (cited in page 1)

Boguraev, B. and E. J. Briscoe (1987, June). Large lexicons for natural language processing: utilising the grammar coding system of LDOCE. *Computational Linguistics 13*(2), 203–218. (cited in pages 88, 90, 92, 151)

Bouillon, P., V. Claveau, C. Fabre, and P. Sébillot (2003, May). Learning semantic lexicons from a part-of-speech and semantically tagged corpus using inductive logic programming. *Journal of Machine Learning Research 4*(4), 493–525. (cited in page 1)

Boullier, P. (1998). A generalization of mildly context-sensitive formalisms. In *Proceedings of the fourth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4)*, Philadelphia, PA, USA, pp. 17–20. (cited in page 75)

Boullier, P. (1999). On TAG parsing Multicomponent TAG parsing. Technical Report Research Report N 3668, INRIA-Rocquencourt. (cited in page 75)

Bouma, G., G. van Noord, and R. Malouf (2000). Alpino: Wide coverage computational analysis of Dutch. In *Proceedings of the 11th Meeting of Computational Linguistics in the Netherlands (CLIN 2000)*, Tilburg, Netherlands, pp. 45–59. (cited in page 23)

Brants, T. (2000). TnT – a statistical part-of-speech tagger. In *Proceedings of the sixth Conference on Applied Natural Language Processing (ANLP 2000)*, Seattle, WA, USA, pp. 224–231. (cited in page 1)

Brent, M. R. (1993, June). From grammar to lexicon. *Computational Linguistics 19*(2), 243–262. (cited in pages 6, 89, 101)

Brill, E. (1994). Some advances in transformation-based part of speech tagging. In *Proceedings of the 12th National Conference on Artificial Intelligence*, Seattle, WA, USA, pp. 722–727. (cited in page 1)

Briscoe, E. J. (2001). From dictionary to corpus to self-organizing dictionary: learning valency associations in the face of variation and change. In *Proceedings of the Corpus Linguistics*, Lancaster, UK, pp. 79–89. (cited in pages 5, 6, 88)

Briscoe, E. J. and J. Carroll (1993, March). Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics 19*(1), 25–60. (cited in page 3)

Briscoe, E. J. and J. Carroll (1997). Automatic extraction of subcategorization from corpora. In *Proceedings of the fifth Conference on Applied Natural Language Processing (ANLP 1997)*, Washington, DC, USA, pp. 356–363. (cited in pages 6, 88, 89, 90, 91, 92, 101, 102, 107, 151)

Briscoe, E. J. and A. Copestake (1999). Lexical rules in constraint-based grammar. *Computational Linguistics 25*(4), 487–526. (cited in page 87)

Brown, P. F., V. J. D. Pietra, P. V. deSouza, J. C. Lai, and R. L. Mercer (1999, December). Class-based n-gram models of natural language. *Computational Linguistics 18*(4), 467–479. (cited in page 112)

Callmeier, U. (2000). PET — a platform for experimentation with efficient hpsg processing techniques. *Natural Language Engineering 6*(1), 99–108. (cited in page 80)

Candito, M.-H. (1996). A principle-based hierarchical representation of LTAGs. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING 1996)*, Copenhagen, Denmark, pp. 194–199. (cited in page 87)

Carpenter, B. (1992). *The Logic of Typed Feature Structures*. Cambridge University Press. (cited in pages 12, 15, 20)

Carreras, X. and L. Màrques (2004). Introduction to the conll-2004 shared task: Semantic role labeling. In H. T. Ng and E. Riloff (Eds.), *Proceedings of the eighth Workshop on Computational Language Learning (CoNLL 2004)*, pp. 89–97. Boston, MA, USA. (cited in page 1)

Carroll, G. and M. Rooth (1998). Valence induction with a head-lexicalized PCFG. In *Proceedings of the third Conference on Empirical Methods in Natural Language Processing (EMNLP 1998)*, Granada. Spain, pp. 36–45. (cited in pages 6, 89, 90)

Carroll, J. (1994). Relating complexity to practical performance in parsing with wide-coverage unification grammars. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics (ACL 1994)*, Las Cruces, NM, USA, pp. 287–294. (cited in pages 3, 66)

Carroll, J., A. Copestake, D. Flickinger, and V. Poznanski (1999). An efficient chart generator for (semi-)lexicalist grammars. In *Proceedings of the seventh European Workshop on Natural Language Generation (EWNLG 1999)*, Toulouse, France, pp. 86–95. (cited in page 22)

Carroll, J. and A. C. Fang (2004). The automatic acquisition of verb subcategorizations and their impact on the performance of an HPSG parser. In *Proceedings of the first International Joint Conference on Natural Language Processing(ijc-NLP 2004)*, Hainan Island, China, pp. 107–114. (cited in pages 6, 90, 92, 93, 102)

Carroll, J., G. Minnen, and E. J. Briscoe (1998). Can subcategorization probabilities help a statistical parser? In *Proceedings of the sixth ACL/SIGDAT Workshop on Very Large Corpora*, Montreal, Quebec, Canada, pp. 1–9. (cited in page 89)

Chandrasekar, R. and B. Srinivas (1997). Gleaning information from the web: Using syntax to filter out irrelevant information. In *Proceedings of AAAI 1997 Spring Symposium on Natural Language Processing on the World Wide Web*, Washington, DC, USA, pp. 27–34. (cited in page 21)

Charniak, E. (1997). Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI 1997)*, Menlo Park, CA, USA, pp. 598–603. AAAI Press/MIT Press. (cited in page 1)

Chen, J. and O. Rambow (2003). Use of deep linguistic features for the recognition and labeling of semantic arguments. In *Proceedings of the eighth Conference on Empirical Methods in Natural Language Processing (EMNLP 2003)*, Sapporo, Japan, pp. 41–48. (cited in page 1)

Chen, J. and K. Vijay-Shanker (1997). Towards a reduced-commitment D-theory style TAG parser. In *Proceedings of the fifth International Workshop on Parsing Technologies (IWPT 1997)*, Boston, MA, USA, pp. 18–29. (cited in page 78)

Chen, J. and K. Vijay-Shanker (2000). Automated extraction of TAGs from the Penn Treebank. In *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT 2000)*, Trento, Italy, pp. 65–76. (cited in page 91)

Chiang, D. (2000). Statistical parsing with an automatically-extracted Tree Adjoining Grammar. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL 2000)*, pp. 456–463. (cited in page 91)

Chomsky, N. (1963). Formal properties of grammar. In R. D. Luce, R. R. Bush, and E. Galanter (Eds.), *Handbook of Mathematical Psychology*, Volume II, pp. 323–418. John Wiley and Sons, Inc. (cited in page 3)

Collins, M. (1996). A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL 1996)*, Santa Cruz, CA, USA, pp. 184–191. (cited in page 1)

Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and the eighth Conference of the European Chapter of the Association for Computational Linguistics (ACL–EACL 1997)*, Madrid, Spain, pp. 16–23. (cited in page 1)

Collins, M. (2003, December). Head-driven statistical models for natural language parsing. *Computational Linguistics 29*(4), 589–637. (cited in pages 1, 89)

Copestake, A. (2002). *Implementing typed feature structure grammars*. CSLI publications. (cited in pages 6, 92, 102)

Copestake, A., D. Flickinger, R. Malouf, S. Riehemann, and I. Sag (1995). Translation using Minimal Recursion Semantics. In *Proceedings of the Sixth International Conference on Theoretical and Methodological Issues in Machine Translation (TMI 1995)*, Leuven, Belgium, pp. 15–32. (cited in pages 1, 22)

Copestake, A., F. Lambeau, A. Villavicencio, F. Bond, T. Baldwin, I. A. Sag, and D. Flickinger (2002). Multiword expressions: Linguistic precision and reusability. In *Proceedings of the third*

International Conference on Language Resources and Evaluation (LREC 2002), Las Palmas, Canary Islands, Spain, pp. 1941–1947. (cited in pages 36, 52)

Cutting, D., J. Kupiec, J. Pedersen, and P. Sibun (1992). A practical part-of-speech tagger. In *Proceedings of the third Conference on Applied Natural Language Processing (ANLP 1992)*, Trento, Italy, pp. 133–140. (cited in page 1)

Dang, H. T., K. Kipper, M. Palmer, and J. Posenzweig (1998). Investigating regular sense extensions based on intersective Levin classes. In *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics (COLING–ACL 1998)*, Montreal, Quebec, Canada, pp. 293–299. (cited in page 94)

de Groote, P. (2001). Towards abstract categorial grammars. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL 2001)*, Toulouse France, pp. 148–155. (cited in page 127)

Dempster, A. P., N. M. Laird, and D. B. Rubin (1997). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society 39*(1), 1–38. (cited in pages 112, 114)

Doran, C., B. A. Hockey, A. Sarkar, B. Srinivas, and F. Xia (2000). Evolution of the XTAG system. In A. Abeillé and O. Rambow (Eds.), *Tree Adjoining Grammars: formalisms, linguistic analysis and processing*, pp. 371–403. CSLI publications. (cited in pages 3, 5, 22)

Doran, C. and B. Srinivas (2000). Developing a wide-coverage CCG system. In A. Abeillé and O. Rambow (Eds.), *Tree Adjoining Grammars: formalisms, linguistic analysis and processing*. CSLI publications. (cited in page 74)

Dorr, B. (1997). Large-scale dictionary construction for foreign language tutoring and interlingual machine translation. *Machine Translation 12*(4), 271–325. (cited in page 94)

Dowty, D. (1991). Thematic proto-roles and argument selection. *Languages 67*(3), 547–619. (cited in page 1)

Dunning, T. (1993, March). Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics 19*(1), 61–74. (cited in page 101)

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the Association for Computing Machinery (ACM) 6*(8), 451–455. (cited in pages 3, 67)

Eisner, J. (1996). Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING 1996)*, Copenhagen, Denmark, pp. 340–345. (cited in page 1)

Ersan, M. and E. Charniak (1996). A statistical syntactic disambiguation program and what it learns. In S. Wermter, E. Riloff, and G. Scheler (Eds.), *Connectionist, Statistical and Symbolic Approaches to Learning for Natural Language Processing*, pp. 146–159. Springer-Verlag. (cited in pages 6, 89, 101)

Evans, R. and D. J. Weir (1998). A structure-sharing parser for lexicalized grammars. In *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics (COLING–ACL 1998)*, Montreal, Quebec, Canada, pp. 372–378. (cited in page 77)

Fellbaum, C. (Ed.) (1998). *WordNet: An Electronic Lexical Database*. The MIT Press. (cited in pages 108, 129)

Flickinger, D. (2002). On building a more efficient grammar by exploiting types. In S. Oepen, D. Flickinger, J. Tsujii, and H. Uszkoreit (Eds.), *Collaborative Language Engineering*, pp. 1–17. CSLI Publications. (cited in pages 4, 5, 22, 80)

Flickinger, D., S. Oepen, J. Tsujii, and H. Uszkoreit (Eds.) (2000). *Natural Language Engineering – Special Issue on Efficient Processing with HPSG: Methods, Systems, Evaluation*. Cambridge University Press. (cited in page 23)

Forgy, E. W. (1965). Cluster analysis of multivariate data: Efficiency vs. interpretability of classifications. *Biometrics 21*, 768–780. (cited in page 123)

Gahl, S. (1998). Automatic extraction of subcorpora based on subcategorization frames from a part-of-speech tagged corpus. In *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics (COLING–ACL 1998)*, Montreal, Quebec, Canada, pp. 428–432. (cited in pages 6, 89)

Garside, R., G. Leech, and G. Sampson (1987). *The Computational Analysis of English: A Corpus-Based Approach*. Longman, London. (cited in page 151)

Gazdar, G. (1988). Applicability of indexed grammars to natural languages. In U. Reyle and C. Rohrer (Eds.), *Natural Language Parsing and Linguistic Theories*, pp. 69–94. D. Reidel, Dordrecht. (cited in pages 1, 73)

Gelman, A., J. B. Carlin, H. S. Stern, and D. B. Rubin (Eds.) (1995). *Bayesian Data Analysis*. Chapman and Hall. (cited in page 97)

Gildea, D. and M. Palmer (2002). The necessity of parsing for predicate argument recognition. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL 2002)*, Philadelphia, PA, USA, pp. 239–246. (cited in page 1)

Gorn, S. (1962). Processors for infinite codes of shannon-fano type. In *Proceedings of the Symposium on Mathematical Theory of Automata*, pp. 223–240. (cited in page 39)

Gorrell, G. (2002). Acquiring subcategorisation from textual corpora. MPhil dissertation, University of Cambridge, UK. (cited in page 101)

Greibach, S. A. (1965). A new normal-form theorem for context-free phrase structure grammars. *Journal of the Association for Computing Machinery (JACM) 12*(1), 42–52. (cited in page 81)

Griffith, J. (1995). Optimizing feature structure unification with dependent disjunctions. In *Proceedings of the Workshop on Grammar Formalism for NLP at the sixth European Summer School in Logic, Language and Information (ESSLLI 1994)*, pp. 37–60. (cited in page 78)

Griffith, J. (1996). Modularizing contexted constraints. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING 1996)*, pp. 448–453. (cited in page 78)

Grishman, R., C. Macleod, and A. Meyers (1994). Comlex syntax: Building a computational lexicon. In *Proceedings 15th International Conference on Computational Linguistics (COLING 1994)*, Kyoto, Japan, pp. 268–272. (cited in pages 86, 88, 90, 92, 151)

Haas, A. R. (1987). Parallel parsing for unification grammars. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 1987)*, Milan, Italy, pp. 615–618. (cited in pages 3, 4, 23, 55, 57, 61)

Hamerly, G. (2003). *Learning structure and concepts in data through data clustering*. Ph. D. thesis, University of California, San Diego, CA, USA. (cited in page 101)

Harabagiu, S., D. Moldovan, M. Pasca, R. Mihalcea, M. Surdeanu, R. Bunescu, R. Girju, V. Rus, and P. Morarescu (2001). The role of lexico-semantic feedback in open-domain textual question-answering. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL 2001)*, Toulouse France, pp. 274–281. (cited in page 1)

Harbusch, K. (1990). An efficient parsing algorithm for Tree Adjoining Grammars. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics (ACL 1990)*, Milan, Italy, pp. 284–291. (cited in pages 3, 5, 22, 55, 58, 67, 69)

Hockenmaier, J. and M. Steedman (2002). 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. (cited in page 91)

Hofmann, T. (1999a). Probabilistic latent semantic analysis. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI 1999)*, Stockholm, Sweden, pp. 289–269. (cited in pages 111, 112)

Hofmann, T. (1999b). Probabilistic latent semantic indexing. In *Proceedings of the 22th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1999)*, Berkeley, CA, USA, pp. 50–57. (cited in pages 111, 112)

Huang, M., Z. Xiaoyan, Y. Hao, D. G. Payan, K. Qu, and M. Li (2004). Discovering patterns to extract protein-protein interactions from full texts. *Bioinformatics 20*(18), 3604–3612. (cited in page 1)

Imai, H., Y. Miyao, and J. Tsujii (1998, September). GUI for HPSG parsers. *IPSJ SIG Notes NL*(127), 173–178. (in Japanese). (cited in page 23)

Jackendoff, R. (1990). *Semantic Structures*. MIT Press. (cited in page 1)

Jelinek, F., J. Lafferty, D. Magerman, R. Mercer, A. Ratnaparkhi, and S. Roukos (1994). Decision tree parsing using a hidden derivation model. In *Proceedings of the ARPA Workshop on Human Language Technology (HLT 1994)*, San Francisco, CA, USA, pp. 272–277. (cited in page 1)

Joshi, A., K. Vijay-Shanker, and D. J. Weir (1991). The convergence of mildly context-sensitive grammar formalisms. In T. Wasow and P. Sells (Eds.), *Processing of Linguistic Structure*, pp. 31–81. MIT Press. (cited in page 73)

Joshi, A. K. (1987). Word-order variation in natural language generation. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI)*, pp. 550–555. (cited in page 21)

Joshi, A. K. (1990). Processing crossed and nested dependencies: an automaton perspective on the psycholinguistic results. *Language and Cognitive Processes 5*(1), 1–27. (cited in page 21)

Joshi, A. K., L. S. Levy, and M. Takahashi (1975). Tree Adjunct Grammars. *Computer and System Science 10*(1), 136–163. (cited in pages 12, 73)

Kanayama, H., K. Torisawa, Y. Mitsuishi, and J. Tsujii (2000). Hybrid Japanese parser with hand-crafted grammar and statistics. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING 2000)*, Saarbrücken, Germany, pp. 411–417. (cited in page 23)

Kaplan, R. M. and J. Bresnan (1982). Lexical Functional Grammar: A formal system for grammatical representation. In J. Bresnan (Ed.), *The Mental Representation of Grammatical Relation*, pp. 173–181. MIT Press. (cited in pages 1, 79)

Kasami, T. (1965). An efficient recognition and syntax algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Lab., Bedford, Mass. (cited in page 3)

Kasper, R. (1998). TAG and HPSG. Talk given in the tutorial session at the fourth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4). (cited in page 3)

Kasper, R., B. Kiefer, K. Netter, and K. Vijay-Shanker (1995). Compilation of HPSG to TAG. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL 1995)*, Cambridge, MA, USA, pp. 92–99. (cited in pages 3, 19, 26, 31, 74)

Kay, M. (2000, February). Guides and oracles for linear-time parsing. In *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT 2000)*, Trento, Italy, pp. 6–9. (cited in page 3)

Kay, M., J. M. Gawron, and P. Norvig (1994). *Verbmobil: A Translation System for Face-to-Face Dialog*. CSLI Publications. (cited in pages 4, 22)

Keller, B. (1994). *Feature Logics, Infinitary Descriptions and Grammars*. CSLI publications. (cited in page 20)

Kiefer, B. and H.-U. Krieger (2000). A context-free approximation of Head-Driven Phrase Structure Grammar. In *Proceedings of the sixth International Workshop on Parsing Technologies (IWPT 2000)*, Trento, Italy, pp. 135–146. (cited in pages 3, 5, 23, 55, 60, 67)

Kim, A., B. Srinivas, and J. Trueswell (1990). The convergence of lexicalist perspectives in psycholinguistics and computational linguistics. In P. Merlo and S. Stevenson (Eds.), *Sentence Processing and the Lexicon: Formal, Computational and Experimental Perspectives*, pp. 109–135. John Venjamins Publisher. (cited in page 21)

Kinyon, A. (1999). Some remarks on the psycholinguistic relevance of LTAGs. In *Proceedings of the 10th Meeting of Computational Linguistics in the Netherlands (CLIN 1999)*, Utrecht, Netherlands, pp. 99–108. (cited in page 21)

Korhonen, A. (1998). Automatic extraction of subcategorization frames from corpora - improving filtering with diathesis alternations. In *Proceedings of the Workshop on Automated Acquisition of Syntax and Parsing at the 10th European Summer School in Logic, Language and Information (ESSLLI 1998)*, Saarbrücken, Germany, pp. 49–56. (cited in page 107)

Korhonen, A. (2002). *Subcategorization Acquisition*. Ph. D. thesis, University of Cambridge, Cambridge, UK. (cited in pages 6, 89, 92, 101, 107, 108)

Korhonen, A. and E. J. Briscoe (2004). Extended lexical-semantic classification of English verbs. In D. Moldovan and R. Girju (Eds.), *Proceedings of the Computational Lexical Semantics Workshop at Human Language Technology Conference and the fifth Annual Meeting of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2004)*, Boston, MA, USA, pp. 38–45. (cited in page 94)

Korhonen, A., Y. Krymolowski, and Z. Marx (2003). Clustering polysemic subcategorization frame distributions semantically. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL 2003)*, Sapporo, Japan, pp. 64–71. (cited in pages 122, 123)

Kornai, A. and G. K. Pullum (1990). The X-bar theory of phrase structure. *Language 66*, 24–50. (cited in page 3)

Kroch, A. (1987). Subjacency in a Tree Adjoining Grammar. *Mathematics of Language*, 143–172. (cited in page 13)

Kroch, A. (1989). Asymmetries in long-distance extraction in a Tree-Adjoining Grammar. In M. Baltin and A. Kroch (Eds.), *Alternative Conceptions of Phrase Structure*, pp. 66–98. University of Chicago Press. (cited in pages 13, 36)

Kroch, A. and A. K. Joshi (1986). Analyzing extraposition in a Tree Adjoining Grammar. In G. Huck and A. Ojeda (Eds.), *Syntax & Semantics: Discontinuous Constituents*, pp. 107–149. (cited in page 13)

Kudo, T. and Y. Matsumoto (2002). Japanese dependency analysis using cascaded chunking. In *Proceedings of the sixth Workshop on Computational Language Learning (CoNLL 2002)*, Taipei, Taiwan, pp. 63–69. (cited in page 1)

Kuhn, J., J. Eckle-Kohler, and C. Rohrer (1998). Lexicon acquisition with and for symbolic NLP-systems — a bootstrapping approach. In *Proceedings of the first International Conference on Language Resources and Evaluation (LREC 1998)*, Granada, Spain, pp. 89–95. (cited in pages 6, 89, 90)

Kurohashi, S. and M. Nagao (1994, December). A syntactic analysis method of long japanese sentences based on the detection of conjunctive structures. *Computational Linguistics 20*(4), 507–534. (cited in page 1)

Lapata, M. (1999). Acquiring lexical generalization from corpora: A case study for diathesis alternations. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL 1999)*, College Park, MD, USA, pp. 397–404. (cited in pages 6, 89, 101)

Lavelli, A. and G. Satta (1991). Bidirectional parsing of Lexicalized Tree Adjoining Grammars. In *Proceedings of the fifth Conference of the European Chapter of the Association for Computational Linguistics (EACL 1991)*, Berlin, Germany, pp. 27–32. (cited in page 22)

Lee, L. (2002). Fast context-free grammar parsing requires fast Boolean matrix multiplication. *Journal of the ACM 49*(1), 1–15. (cited in page 3)

Leermakers, R. (1992). A recursive ascent Earley parser. *Information Processing Letters 41*(2), 87–91. (cited in page 66)

Levin, B. (1993). *English Verb Classes and Alternations: A Preliminary Investigation*. University of Chicago Press. (cited in pages 1, 6, 93, 94, 108)

Li, H. and N. Abe (1998). Word clustering and disambiguation based on co-occurrence data. In *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics (COLING–ACL 1998)*, Montreal, Quebec, Canada, pp. 749–755. (cited in pages 112, 129)

Li, X. and D. Roth (2001). Exploring evidence for shallow parsing. In W. Daelemans and R. Zajac (Eds.), *Proceedings of the fifth Workshop on Computational Language Learning (CoNLL 2001)*, pp. 38–44. Toulouse, France. (cited in page 1)

Magerman, D. (1995). Statistical decision-tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL 1995)*, Cambridge, MA, USA, pp. 276–283. (cited in page 1)

Makino, T., M. Yoshida, K. Torisawa, and J. Tsujii (1998). LiLFeS — towards a practical HPSG parsers. In *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics (COLING–ACL 1998)*, Montreal, Quebec, Canada, pp. 807–811. (cited in page 80)

Malouf, R., J. Carroll, and A. Copestake (2003). Efficient feature structure operations without compilation. In S. Oepen, D. Flickinger, J. Tsujii, and H. Uszkoreit (Eds.), *Collaborative Language Engineering*, pp. 103–124. CSLI Publications. (cited in pages 23, 80)

Manning, C. D. (1993). Automatic acquisition of a large subcategorization dictionary from corpora. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics (ACL 1993)*, Columbus, OH, USA, pp. 235–242. (cited in pages 6, 89, 101)

Marcus, M., B. Santorini, and M. A. Marcinkiewicz (1993, June). Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics 19*(2), 313–330. (cited in pages 7, 61, 67, 91)

Maxwell III, J. T. and R. M. Kaplan (1993, December). The interface between phrasal and functional constraints. *Computational Linguistics 19*(4), 571–590. (cited in pages 3, 79)

McCarthy, D. F. (2001). *Lexical Acquisition at the Syntax-Semantics Interface: Diathesis Alternations, Subcategorization Frames and Selectional Preferences*. Ph. D. thesis, University of Sussex, Brighton, UK. (cited in pages 6, 93)

McCoy, K. F., K. Vijay-Shanker, and G. Yang (1992). A functional approach to generation with TAG. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics (ACL 1992)*, Newark, DE, USA, pp. 48–55. (cited in page 21)

Meyers, A., C. Macleod, and R. Grishman (1994). Standardization of the complement/adjunct distinction. Technical report, New York University. (cited in page 86)

Miller, P. H. (1999). *Strong Generative Capacity*. CSLI publications. (cited in pages 3, 41, 127)

Mitsuishi, Y., K. Torisawa, and J. Tsujii (1998). HPSG-style underspecified Japanese grammar with wide coverage. In *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics (COLING–ACL 1998)*, Montreal, Quebec, Canada, pp. 876–880. (cited in page 23)

Miyao, Y. (1999). 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 1999)*, Maryland, NB, Canada, pp. 579–584. (cited in page 79)

Miyao, Y., T. Ninomiya, and J. Tsujii (2003). Lexicalized grammar acquisition. In *the Companion Volume to the Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2003)*, Budapest, Hungary, pp. 127–130. (cited in pages 67, 91)

Miyao, Y., T. Ninomiya, and J. Tsujii (2004). Corpus-oriented grammar development for acquiring a Head-driven Phrase Structure Grammar from the Penn Treebank. In *Proceedings of the first International Joint Conference on Natural Language Processing(ijc-NLP 2004)*, Hainan Island, China. (cited in pages 5, 23, 89, 90, 91, 120)

Nakanishi, H., Y. Miyao, and J. Tsujii (2004). Using inverse lexical rules to acquire a wide-coverage lexicalized grammar. In *Proceedings of the Workshop on Beyond Shallow Analyses at the first International Joint Conference on Natural Language Processing (ijc-NLP 2004)*, Hainan Island, China. (cited in pages 87, 89, 91, 120)

Nederhof, M.-J. (1998). An alternative LR algorithm for TAGs. In *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics (COLING–ACL 1998)*, Montreal, Quebec, Canada, pp. 946–952. (cited in page 3)

Nederhof, M.-J. (1999). Models of tabulation for TAG parsing. In *Proceedings of the sixth Meeting on Mathematics of Language (MOL 6)*, Orlando, FL, USA, pp. 143–158. (cited in page 22)

Nishida, K., K. Torisawa, and J. Tsujii (1999). An efficient HPSG parsing algorithm with array unification. In *Proceedings of the fifth Natural Language Processing Pacific Rim Symposium (NLPRS 1999)*, Beijing, China, pp. 144–149. (cited in page 80)

Oepen, S., D. Flickinger, J. Tsujii, and H. Uszkoreit (Eds.) (2002). *Collaborative Language Engineering*. CSLI Publications. (cited in page 23)

Oishi, A. (1998). *Semantic Structures of Japanese Verb Phrases — Acquisition, Representation and Use*. Ph. D. thesis, Nara Institute of Science and Technology, Nara, Japan. (cited in page 108)

Oouchida, K., N. Yoshinaga, and J. Tsujii (2004). Context-free approximation of ltag towards cfg filtering. In *Proceedings of TAG+7*, Vancouver, BC, Canada, pp. 171–177. (cited in page 81)

Palmer, M., O. Rambow, and A. Nasr (1998). Rapid prototyping of domain-specific machine translation. In *Proceedings of the third conference of the Association for Machine Translation in the Americas (AMTA 1998)*, Langhorne, PA, USA, pp. 95–102. (cited in pages 1, 21)

Paroubek, P., Y. Schabes, and A. K. Joshi (1992). XTAG - a graphical workbench for developing Tree-Adjoining Grammars. In *Proceedings of the third Conference on Applied Natural Language Processing (ANLP 1992)*, Trento, Italy, pp. 223–230. (cited in page 22)

Pelleg, D. and A. Moore (2000). X-means: Extending K-means with efficient estimation of the number of clusters. In *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, San Francisco, CA, USA, pp. 727–734. (cited in page 101)

Pereira, F. C., N. Tishby, and L. Lee (1993). Distributional clustering of english words. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics(ACL 1993)*, Columbus, OH, USA, pp. 183–190. (cited in page 112)

Pinker, S. (1989). *Learnability and Cognition: The Acquisition of Argument Structures*. MIT Press. (cited in page 1)

Pollard, C. (1984). *Generalized Phrase Structure Grammars, Head Grammars and Natural Language*. Ph. D. thesis, Stanford University, Palo Alto, CA, USA. (cited in page 73)

Pollard, C. and I. A. Sag (1994). *Head-Driven Phrase Structure Grammar*. University of Chicago Press and CSLI Publications. (cited in pages 1, 2, 11, 19, 20, 35, 54, 87, 127)

Poller, P. (1994, November). Incremental parsing with LD/TLP-TAGs. *Computational Intelligence 10*(4), 549–562. (cited in pages 3, 5, 22, 55, 58, 59)

Poller, P. and T. Becker (1998). Two-step TAG parsing revisited. In *Proceedings of the fourth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4)*, Philadelphia, PA, USA, pp. 143–146. (cited in pages 3, 5, 22, 55, 58, 67)

Prolo, C. A. (2002). Generating the XTAG english grammar using metarules. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002)*, Taipei, Taiwan, pp. 814–820. (cited in page 87)

Pustejovsky, J. (1995). *The Generative Lexicon*. MIT Press. (cited in page 1)

Rajasekaran, S. and S. Yooseph (1998, February). TAL recognition in $O(M(n^2))$ time. *Journal of Computer and System Science 56*(1), 83–89. (cited in pages 3, 22)

Ratnaparkhi, A. (1996). A maximum entropy part-of-speech tagger. In *Proceedings of the first Conference on Empirical Methods in Natural Language Processing (EMNLP 1996)*, Philadelphia, PA, USA, pp. 491–497. (cited in page 1)

Riehemann, S. (2001). *A Constructional Approach to Idioms and Word Formation*. Ph. D. thesis, Stanford University, Palo Alto, CA, USA. (cited in pages 37, 52)

Roland, D. (2001). *Verb sense and verb subcategorization probabilities*. Ph. D. thesis, University of Colorado, Boulder, CO, USA. (cited in pages 2, 5, 91)

Rosenkrantz, D. J. (1967). Matrix equations and normal forms for context-free grammars. *Journal of the Association for Computing Machinery 14*(3), 501–507. (cited in page 81)

Sarkar, A. (2000). Practical experiments in parsing using Tree Adjoining Grammars. In *Proceedings of the fifth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+5)*, Paris, France, pp. 193–198. (cited in pages 4, 6, 22, 55, 61, 89, 101)

Sarkar, A., F. Xia, and A. K. Joshi (2000). Some experiments on indicators of parsing complexity for lexicalized grammars. In *Proceedings of the Workshop on Efficiency in Large-Scale Parsing Systems at the 17th International Conference on Computational Linguistics (COLING 2000)*, Saarbrücken, Germany, pp. 37–42. (cited in pages 7, 66, 76)

Sarkar, A. and D. Zeman (2000). Automatic extraction of subcategorization frames for Czech. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING 2000)*, Saarbrücken, Germany, pp. 691–697. (cited in pages 90, 91, 101)

Satta, G. (1994, June). Tree Adjoining Grammar parsing and boolean matrix multiplication. *Computational Linguistics 20*(2), 173–191. (cited in pages 3, 22)

Schabes, Y. (1994). Left-to-right parsing in Lexicalized Tree Adjoining Grammar. *Computational Intelligence 10*(4), 506–524. (cited in pages 3, 22)

Schabes, Y., A. Abeillé, and A. K. Joshi (1988). Parsing strategies with 'lexicalized' grammars: application to Tree Adjoining Grammars. In *Proceedings 12th International Conference on Computational Linguistics (COLING 1988)*, Budapest, Hungary, pp. 578–583. (cited in pages 1, 2, 11, 12)

Schabes, Y. and A. Joshi (1988). An earley-type parsing algorithm for Tree Adjoining Grammars. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics (ACL 1988)*, Buffal, NY, USA, pp. 258–269. (cited in page 22)

Schabes, Y. and S. M. Shieber (1994). An alternative conception of Tree-Adjoining derivation. *Computational Linguistics 20*(1), 91–124. (cited in page 36)

Schabes, Y. and R. C. Waters (1995, December). Tree Insertion Grammar: A cubic-time parsable formalism that lexicalizes context-free grammar without changing the tree produced. *Computational Linguistics 21*(4), 479–513. (cited in pages 3, 41, 81)

Schulte im Walde, S. and C. Brew (2002). Inducing German semantic verb classes from purely syntactic subcategorisation information. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL 2003)*, Sapporo, Japan, pp. 223–230. (cited in pages 1, 122, 123)

Sebastiani, F. (2002, March). Machine learning in automated text categorization. *ACM Computing Surveys 34*(1), 1–47. (cited in page 1)

Sekine, S. (1998). *Corpus-based Parsing and Sublanguage Studies*. Ph. D. thesis, New York University, New York, NY, USA. (cited in pages 2, 5)

Shaumyan, O., J. Carroll, and D. J. Weir (2002). Evaluation of ltag parsing with supertag compaction. In *Proceedings of the sixth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+6)*, Venice, Italy, pp. 201–205. (cited in page 78)

Shieber, S. M. (1985). Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics (ACL 1985)*, Chicago, IL., USA, pp. 145–152. (cited in pages 60, 79)

Somers, H. L. (1984). On the validity of the complement-adjunct distinction in valency grammar. *Linguistics 22*, 507–530. (cited in page 86)

Steedman, M. (1985). Dependency and coordination in the grammar of Dutch and English. *Language 61*, 523–568. (cited in page 73)

Steedman, M. (1986). Combinators and grammars. In R. Oehrle, E. Bach, and D. Wheeler (Eds.), *Categorial Grammars and Natural Language Structures*, pp. 417–442. Foris, Dordrecht. (cited in pages 1, 73)

Steedman, M. (2000). *The Syntactic Process*. The MIT Press. (cited in page 2)

Stone, M. and C. Doran (1997). Sentence planning as description using Tree Adjoining Grammar. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and the eighth Conference of the European Chapter of the Association for Computational Linguistics (ACL–EACL 1997)*, Madrid, Spain, pp. 198–205. (cited in page 21)

Surdeanu, M., S. Harabagiu, J. Williams, and P. Aarseth (2003). Using predicate-argument structures for information extraction. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL 2003)*, Sapporo, Japan, pp. 8–15. (cited in page 1)

Tateisi, Y., K. Torisawa, Y. Miyao, and J. Tsujii (1998). Translating the XTAG English grammar to HPSG. In *Proceedings of the fourth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4)*, Philadelphia, PA, USA, pp. 172–175. (cited in pages 3, 23, 35, 36, 74, 80)

The XTAG Research Group (1995). A Lexicalized Tree Adjoining Grammar for English. http://www.cis.upenn.edu/˜xtag/. (cited in page 74)

Tomita, M. (1986). *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems*. Kluwer Academic Publisher. (cited in page 3)

Tomuro, N. and S. L. Lytinen (2001). Abstract left-corner parsing for unification grammars. In *Proceedings of the sixth Natural Language Processing Pacific Rim Symposium (NLPRS 2001)*, Tokyo, Japan, pp. 367–374. (cited in page 23)

Torisawa, K., K. Nishida, Y. Miyao, and J. Tsujii (2000). An HPSG parser with CFG filtering. *Natural Language Engineering 6*(1), 63–80. (cited in pages 3, 5, 23, 55, 60, 67, 80)

Torisawa, K. and J. Tsujii (1995). Compiling HPSG-style grammar to object-oriented language. In *Proceedings of the third Natural Language Processing Pacific Rim Symposium (NLPRS 1995)*, Seoul, Korea, pp. 320–325. (cited in page 3)

Torisawa, K. and J. Tsujii (1996). Computing phrasal-signs in HPSG prior to parsing. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING 1996)*, Copenhagen, Denmark, pp. 949–955. (cited in pages 3, 5, 23, 55, 60, 80)

Tsujii, J. (2001). LiLFeS/GENIA project – NLP tools and a biology domain corpus –. In *Proceedings of the sixth Natural Language Processing Pacific Rim Symposium (NLPRS 2001)*, Tokyo, Japan, pp. 765–766. (cited in page 22)

Tsuruoka, Y. and T. Chikayama (2001). Estimating reliability of contextual evidences in decision-list classifiers under Bayesian learning. In *Proceedings of the sixth Natural Language Processing Pacific Rim Symposium (NLPRS 2001)*, Tokyo, Japan, pp. 701–707. (cited in page 98)

Ushioda, A., D. A. Evans, T. Gibson, and A. Waibel (1993). The automatic acquisition of frequencies of verb subcategorization frames from tagged corpora. In B. Boguraev and J. Pustejovsky (Eds.), *SIGLEX ACL Workshop on the Acquisition of Lexical Knowledge from Text*, pp. 95–106. (cited in pages 6, 89)

Uszkoreit, H., R. Backofen, S. Busemann, A. K. Diagne, E. A. Hinkelman, W. Kasper, B. Kiefer, H.-U. Krieger, K. Netter, G. Neumann, S. Oepen, and S. P. Spackman (1994). DISCO – an HPSG-based NLP system and its application for appointment scheduling. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING 1994)*, Kyoto, Japan, pp. 436–440. (cited in page 22)

Valiant, L. G. (1975, April). General context-free recognition in less than cubic time. *Journal of Computer and System Science 10*(2), 308–315. (cited in page 3)

van Noord, G. (1994). Head corner parsing for TAG. *Computational Intelligence 10*(4), 525–534. (cited in pages 22, 55, 61, 70)

Vijay-Shanker, K. (1987). *A study of Tree Adjoining Grammars*. Ph. D. thesis, University of Pennsylvania, Philadelphia, PA, USA. (cited in pages 13, 22, 39)

Vijay-Shanker, K. and A. K. Joshi (1985). Some computational properties of Tree Adjoining Grammars. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics (ACL 1985)*, Chicago, IL., USA, pp. 82–93. (cited in page 3)

Vijay-Shanker, K. and A. K. Joshi (1988). Feature structures based Tree Adjoining Grammars. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING 1988)*, Budapest, Hungary, pp. 714–719. (cited in page 13)

Vijay-Shanker, K. and D. J. Weir (1994, December). Parsing some constrained grammar formalisms. *Computational Linguistics 19*(4), 591–636. (cited in page 73)

Weir, D. J. (1988). *Characterizing Mildly Context-Sensitive Grammar Formalisms*. Ph. D. thesis, University of Pennsylvania, Philadelphia, PA, USA. (cited in page 73)

Xia, F. (1999). Extracting Tree Adjoining Grammars from bracketed corpora. In *Proceedings of the fifth Natural Language Processing Pacific Rim Symposium (NLPRS 1999)*, Beijing, China, pp. 398–403. (cited in pages 5, 78, 91)

XTAG Research Group (2001). A Lexicalized Tree Adjoining Grammar for English. Technical Report IRCS-01-03, IRCS, University of Pennsylvania. (cited in pages 4, 6, 14, 21, 51, 52, 53, 54, 88, 102)

Yakushiji, A., Y. Tateisi, Y. Miyao, and J. Tsujii (2001). Event extraction from biomedical papers using a full parser. In *Proceedings of the sixth Pacific Symposium on Biocomputing (PSB 2001)*, Big Island, HI, USA, pp. 408–419. (cited in pages 1, 22, 81)

Yakushiji, A., Y. Tateisi, Y. Miyao, N. Yoshinaga, and J. Tsujii (2003). A debug tool for practical grammar development. In *the Companion Volume to the Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL 2003)*, Sapporo, Japan, pp. 173–176. (cited in page 23)

Yamada, H. and Y. Matsumoto (2003). Statistical dependency analysis with Support Vector Machines. In *Proceedings of the eighth International Workshop on Parsing Technologies (IWPT 2003)*, Nancy, France, pp. 195–206. (cited in page 1)

Yoon, S. (2004). Using a Meta-Grammar for LTAG Korean grammar. In *Proceedings of the seventh International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+7)*, Vancouver, BC, Canada, pp. 211–218. (cited in page 87)

Yoshida, K. (2005). Corpus-oriented method for developing a practical Japanese HPSG parser. University of Tokyo, Tokyo, Japan. (cited in page 23)

Yoshida, M., T. Ninomiya, K. Torisawa, T. Makino, and J. Tsujii (1999). Efficient FB-LTAG parser and its parallelization. In *Proceedings of Pacific Association for Computational Linguistics (PACLING 1999)*, Waterloo, Ontario, Canada, pp. 90–103. (cited in pages 3, 22, 80)

Younger, D. H. (1967, February). Recognition and parsing of context-free languages in time n^3. *Information and Control 2*(10), 189–208. (cited in pages 3, 67)

Zeman, D. (2002). Can subcategorization help a statistical dependency parser? In *Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002)*, Taipei, Taiwan, pp. 1156–1162. (cited in page 89)

Appendix A

Fine-grained 163 SCF Types

This appendix lists 163 fine-grained general-purpose SCF types employed in Briscoe and Carroll's SCF acquisition (Briscoe and Carroll 1997). Each entry expresses one type of SCF. The first line shows the SCF codings provided in the two manually-tailored large-scale syntax dictionaries, namely ANLT (Boguraev and Briscoe 1987) and COMLEX (Grishman et al. 1994). The second line gives a mapping of each SCF type to an SCF type in the XTAG and the ERG coding scheme. 'gap' means that the type is missing in the lexicons. The third line shows an example sentence tagged by claws-2 tagset (Garside et al. 1987).

1.	ADJP / (SUBCAT SC_AP, SUBTYPE EQUI) ⇒ XTAG:Tnx0Va1 / ERG:v_subj_equi_prd_adj_le ex. his_AT reputation_NN1 sank_VVD low_JJ
2.	ADJP-PRED-RS / (SUBCAT SC_AP, SUBTYPE RAIS) ⇒ XTAG:Tnx0Ax1 / ERG:v_prd_ssr_any_le ex. he_NP1 appears_VVZ crazy_JJ / distressed_VVN
3.	ADVP / (SUBCAT ADVP) ⇒ XTAG:gap / ERG:v_adv_le ex. he_NP1 meant_VVD well_RP
4.	ADVP-PRED-RS / (SUBCAT ADVP, SUBTYPE RAIS) ⇒ XTAG:gap / ERG:v_prd_ssr_any_le ex. He_NP1 seems_VVZ well_RP
5.	AS-NP / (SUBCAT SC_NP, SUBTYPE EQUI, PREP as) ⇒ XTAG:gap / ERG:gap ex. I_NP1 worked_VVZ as_CSA an_AT1 apprentice_NN1 cook_NN1
6.	EXTRAP-NP-S / (SUBCAT NP_SFIN, SUBTYPE EXTRAP, AGR N2[NFORM IT]) ⇒ XTAG:gap / ERG:v_expl_it_subj_np_cp_le ex. it_PPH1 annoys_VVZ them_PPHO2 that_CST she_PPHS1 left_VVD
7.	S-SUBJ-NP-OBJ / (SUBCAT NP_SFIN, SUBTYPE EXTRAP, AGR S[FIN +]) ⇒ XTAG:Ts0Vnx1 / ERG:v_np_trans_le ex. that_CST she_PPHS1 left_VVD annoys_VVZ them_PPHO2

8.	TO-INF-SUBJ-NP-OBJ / (SUBCAT OC_INF, SUBTYPE EQU_EXTRAP, AGR VP[FIN -]) ⇒ XTAG:gap / ERG:v_np_trans_le ex. to_TO read_VV0 pleases_VVZ them_PPHO2
9.	EXTRAP-TO-INF / (SUBCAT VPINF, SUBTYPE EXTRAP, AGR N2[NFORM IT]) ⇒ XTAG:gap / ERG:gap ex. it_PPH1 remains_VVZ to_TO find_VV0 a_AT1 cure_NN1
10.	EXTRAP-FOR-TO-INF / (SUBCAT SINF, SUBTYPE EXTRAP, AGR N2[NFORM IT]) ⇒ XTAG:gap / ERG:gap ex. it_PPH1 remains_VVZ for_IF us_PPHO2 to_TO find_VV0 a_AT1 cure_NN1
11.	EXTRAP-NP-TO-INF / (SUBCAT OC_INF, SUBTYPE EQU_EXTRAP, AGR N2[NFORM IT]) ⇒ XTAG:gap / ERG:v_expl_it_subj_np_cp_le ex. it_PPH1 pleases_VVZ them_PPHO2 to_TO find_VV0 a_AT1 cure_NN1
12.	EXTRAP-TO-NP-S / (SUBCAT PP_SFIN, SUBTYPE EXTRAP, PFORM to, AGR N2[NFORM IT]) ⇒ XTAG:gap / ERG:v_expl_it_subj_pp_cp_le ex. it_PPH1 matters_VVZ to_II them_PPHO2 that_CST she_PPHS1 left_VVD
13.	EXTRAP-TO-NP-TO-INF / (SUBCAT PP_VPINF, SUBTYPE EXTRAP, PFORM to) ⇒ XTAG:gap / ERG:gap ex. it_PPH1 occurred_VVD to_II them_PPHO2 to_TO watch_VV0
14.	S-SUBJ-TO-NP-OBJ / (SUBCAT PP_SFIN, SUBTYPE EXTRAP, AGR S[FIN +]) ⇒ XTAG:gap / ERG:v_prep_intrans_le ex. that_CST she_PPHS1 left_VVD matters_VVZ to_II them_PPHO2
15.	FOR-TO-INF / (SUBCAT SINF) ⇒ XTAG:gap / ERG:v_cp_inf_le ex. I_PPHS1 prefer_VV0 for_IF her_PPHO1 to_TO do_VV0 it_PPH1
16.	HOW-S / (SUBCAT WHS) ⇒ XTAG:gap / ERG:v_cp_ques_fin_inf_non_trans_le ex. he_PPHS1 asked_VVD how_RGQ she_PPHS1 did_VDD it_PPH1
17.	HOW-TO-INF / (SUBCAT WHVP) ⇒ XTAG:gap / ERG:v_cp_fin_inf_non_trans_le ex. he_PPHS1 explained_VVD how_RGQ to_TO do_VV0 it_PPH1
18.	INF-AC / gap ⇒ XTAG:gap / ERG:v_subj_equi_bse_le ex. he_PPHS1 helped_VVD bake_VV0 the_AT cake_NN1
19.	ING-NP-OMIT / (SUBCAT SC_ING, SUBTYPE EQUI) ⇒ XTAG:gap / ERG:v_subj_equi_prp_le ex. his_AT hair_NN1 needs_VVZ combing_VVG
20.	ING-SC / (SUBCAT SC_ING, SUBTYPE RAIS) ⇒ XTAG:gap / ERG:v_subj_equi_prp_le ex. she_PPHS1 stopped_VVD smoking_VVG
21.	ING-AC / gap ⇒ XTAG:gap / ERG:v_np_trans_le ex. she_PPHS1 discussed_VVD writing_VVG novels_NN2
22.	INTRANS / (SUBCAT NULL) ⇒ XTAG:gap / ERG:v_unerg_le ex. he_PPHS1 went_VVD
23.	INTRANS-RECIP(SUBJ-PL / (SUBCAT NULL) ⇒ XTAG:gap / ERG:v_unerg_le

			ex. They_PPHS2 met_VVD
24.	NP / (SUBCAT NP)	⇒ XTAG:Tnx0Vnx1 / ERG:v_np_trans_le	ex. he_PPHS1 loved_VVD her_PPHO1
25.	NP-ADJP / (SUBCAT OC_AP, SUBTYPE EQUI)	⇒ XTAG:gap / ERG:v_obj_equi_prd_le	ex. he_PPHS1 painted_VVD the_AT car_NN1 black_JJ
26.	NP-ADJP-PRED / (SUBCAT OC_AP, SUBTYPE RAIS)	⇒ XTAG:Tnx0Vs1 / ERG:v_obj_equi_prd_le	ex. she_PPHS1 considered_VVD him_PPHO1 foolish_JJ
27.	NP-ADVP / (SUBCAT NP_ADVP)	⇒ XTAG:gap / ERG:v_np_prep_trans_dors_le	ex. he_PPHS1 put_VVD it_PPH1 there_RL
28.	NP-ADVP-PRED / (SUBCAT NP_LOC)	⇒ XTAG:Tnx0Vs1 / ERG:v_obj_equi_prd_le	ex. they_PPHS2 mistakenly_RA thought_VVD him_PPHO1 here_RL
29.	NP-AS-NP / (SUBCAT SC_NP_NP, SUBTYPE RAIS, PREP as)	⇒ XTAG:gap / ERG:v_np_comp_le	ex. I_PPHS1 sent_VVD him_PPHO1 as_CSA a_AT1 messenger_NN1
30.	NP-AS-NP-SC / (SUBCAT SC_NP_NP, SUBTYPE RAIS, PREP as)	⇒ XTAG:gap / ERG:gap	ex. she_PPHS1 served_VVD the_AT firm_NN1 as_CSA a_AT1 researcher_NN1
31.	NP-FOR-NP / (SUBCAT NP_PP, SUBTYPE DMOVT, PFORM for)	⇒ XTAG:gap / ERG:v_empty_prep_trans_le	ex. she_PPHS1 bought_VVD a_AT1 book_NN1 for_IF him_PPHO1
32.	NP-INF / (SUBCAT OC_BSE, SUBTYPE RAIS)	⇒ XTAG:Tnx0Vs1 / ERG:v_sorb_le	ex. he_PPHS1 made_VVD her_PPHO1 sing_VV0
33.	NP-INF-OC / (SUBCAT OC_BSE, SUBTYPE EQUI)	⇒ XTAG:gap / ERG:v_obj_equi_bse_le	ex. he_PPHS1 helped_VVD her_PP$ bake_VV0 the_AT cake_NN1
34.	NP-ING / (SUBCAT OC_ING, SUBTYPE RAIS)	⇒ XTAG:Tnx0Vs1 / ERG:v_obj_equi_prd_le	ex. I_PPHS1 kept_VVD them_PPHO2 laughing_VVG
35.	NP-ING-OC / (SUBCAT OC_ING, SUBTYPE EQUI)	⇒ XTAG:gap / ERG:v_obj_equi_prd_le	ex. I_PPHS1 caught_VVD him_PPHO1 stealing_VVG
36.	NP-ING-SC / gap	⇒ XTAG:gap / ERG:v_np_trans_le	ex. he_PPHS1 combed_VVD the_AT woods_NN2 looking_VVG for_IF her_PPHO1
37.	NP-NP / (SUBCAT NP_NP)	⇒ XTAG:Tnx0Vnx1nx2 / ERG:v_ditrans_le	ex. she_PPHS1 asked_VVD him_PPHO1 his_AT name_NN1
38.	NP-NP-PRED / (SUBCAT OC_NP, SUBTYPE EQUI)	⇒ XTAG:Tnx0Vs1 / ERG:gap	ex. they_PPHS2 appointed_VVD him_PPHO1 professor_NN1
39.	NP-P-ING / (SUBCAT OC_PP_ING, PFORM from, SUBTYPE PVERB_OR, ORDER POSTNP)	⇒ XTAG:gap / ERG:gap	

ex. I_PPHS1 prevented_VVD her_PPHO1 from_II leaving_VVG

40. NP-P-ING-OC / (SUBCAT OC_PP_ING, PFORM, SUBTYPE PVERB_OE, ORDER POSTNP)
⇒ XTAG:gap / ERG:v_empty_prep_trans_le
ex. I_PPHS1 accused_VVD her_PPHO1 of_IO murdering_VVG her_AT husband_NN1

41. NP-P-ING-SC / gap
⇒ XTAG:gap / ERG:v_empty_prep_trans_le
ex. he_PPHS1 wasted_VVD time_NNT1 on_II fussing_VVG with_IW his_AT hair_NN1

42. NP-P-ING-AC / gap
⇒ XTAG:gap / ERG:v_empty_prep_trans_le
ex. he_PPHS1 told_VVD her_PPHO1 about_II climbing_VVG the_AT mountain_NN1

43. NP-P-NP-ING / gap
⇒ XTAG:gap / ERG:v_empty_prep_trans_le
ex. he_PPHS1 attributed_VVD his_AT failure_NN1 to_II noone_NP1 buying_VVG his_AT books_NN2

44. NP-P-POSSING / gap
⇒ XTAG:gap / ERG:v_empty_prep_trans_le
ex. They_PPHS2 asked_VVD him_PPHO1 about_II his_PPHO1 participating_VVG in_II

45. NP-P-WH-S / (SUBCAT NP_WHS, PREP)
⇒ XTAG:gap / ERG:v_empty_prep_trans_le
ex. they_PPHS2 made_VVD a_AT1 great_JJ fuss_NN1 about_II whether_CSW they_PPHS2 should_VM participate_VV0

46. NP-P-WHAT-S / (SUBCAT NP_WHS, PREP)
⇒ XTAG:gap / ERG:v_empty_prep_trans_le
ex. they_PPHS2 made_VVD a_AT1 great_JJ fuss_NN1 about_II what_DDQ they_PPHS2 should_VM do_VV0

47. NP-P-WHAT-TO-INF / (SUBCAT NP_WHVP, PREP)
⇒ XTAG:gap / ERG:v_empty_prep_trans_le
ex. they_PPHS2 made_VVD a_AT1 great_JJ fuss_NN1 about_II what_DDQ to_TO do_VV0

48. NP-P-WH-TO-INF / (SUBCAT NP_WHS, PREP)
⇒ XTAG:gap / ERG:v_empty_prep_trans_le
ex. they_PPHS2 made_VVD a_AT1 great_JJ fuss_NN1 about_II whether_CSW to_TO go_VV0

49. NP-PP / (SUBCAT NP_PP, PFORM, SUBTYPE NONE/PVERB?)
⇒ XTAG:Tnx0Vnx1pnx2 / ERG:v_empty_prep_trans_le
ex. she_PPHS1 added_VVD the_AT flowers_NN2 to_II the_AT bouquet_NN1

50. NP-PP-PRED / (SUBCAT NP_PP, PFORM of, SUBTYPE NONE, PRD +)
⇒ XTAG:gap / ERG:v_obj_equi_prd_le
ex. I_PPHS1 considered_VVD that_AT problem_NN1 of_IO little_JJ concern_NN1

51. NP-PRED-RS / (SUBCAT SC_NP, SUBTYPE RAIS)
⇒ XTAG:gap / ERG:gap
ex. he_PPHS1 seemed_VVD a_AT1 fool_NN

52. NP-S / (SUBCAT NP_SFIN, SUBTYPE NONE)
⇒ XTAG:Tnx0Vnx1s2 / ERG:v_np_trans_cp_le
ex. he_PPHS1 told_VVD the_AT audience_NN1 that_CST he_PPHS1 was_VBZ leaving_VVG

53. NP-TO-INF-OC / (SUBCAT OC_INF, SUBTYPE EQUI)
⇒ XTAG:gap / ERG:v_obj_equi_le
ex. I_PPHS1 advised_VVD Mary_NP1 to_TO go_VV0

54. NP-TO-INF-SC / (SUBCAT SC_NP_INF, SUBTYPE EQUI)

	⇒ XTAG:gap / ERG:v_anom_equi_le
	ex. John_NP1 promised_VVD Mary_NP1 to_TO resign_VV0
55.	NP-TO-INF-VC / gap
	⇒ XTAG:gap / ERG:v_obj_equi_le
	ex. they_PPHS2 badgered_VVD him_PPHO1 to_TO go_VV0
56.	NP-TO-NP / (SUBCAT NP_PP, PFORM to, SUBTYPE DMOVT)
	⇒ XTAG:Tnx0Vnx1Pnx2 / ERG:v_empty_to_trans_le
	ex. he_PPHS1 gave_VVD a_AT1 big_JJ kiss_NN1 to_II his_AT mother_NN1
57.	NP-TOBE / (SUBCAT OC_INF, SUBTYPE RAIS)
	⇒ XTAG:gap / ERG:v_obj_equi_le
	ex. I_PPHS1 found_VVD him_PPHO1 to_TO be_VB0 a_AT1 good_JJ doctor_NN1
58.	NP-VEN-NP-OMIT / (SUBCAT OC_PASS, SUBTYPE EQUI/RAISING)
	⇒ XTAG:gap / ERG:v_obj_equi_prd_le
	ex. he_PPHS1 wanted_VVD the_AT children_NN2 found_VVN
59.	NP-WH-S / (SUBCAT NP_WHS)
	⇒ XTAG:gap / ERG:v_np_trans_cp_ques_le
	ex. they_PPHS2 asked_VVD him_PPHO1 whether_CSW he_PPHS1 was_VBZ going_VVG
60.	NP-WHAT-S / (SUBCAT NP_WHS)
	⇒ XTAG:gap / ERG:v_np_trans_cp_ques_le
	ex. they_PPHS2 asked_VVD him_PPHO1 what_DDQ he_PPHS1 was_VBZ doing_VVG
61.	NP-WH-TO-INF / (SUBCAT NP_WHVP)
	⇒ XTAG:gap / ERG:v_np_trans_cp_ques_le
	ex. he_PPHS1 asked_VVD him_PPHO1 whether_CSW to_TO clean_VV0 the_AT house_NN1
62.	NP-WHAT-TO-INF / (SUBCAT NP_WHVP)
	⇒ XTAG:gap / ERG:v_np_trans_cp_ques_le
	ex. he_PPHS1 asked_VVD him_PPHO1 what_DDQ to_TO do_VV0
63.	P-ING-SC / (SUBCAT SC_ING, SUBTYPE EQUI, PREP)
	⇒ XTAG:gap / ERG:v_empty_prep_intrans_le
	ex. they_PPHS2 failed_VVD in_II attempting_VVG the_AT climb_NN1
64.	P-ING-AC / gap
	⇒ XTAG:gap / ERG:v_empty_prep_intrans_le
	ex. they_PPHS2 disapproved_VVD of_IO attempting_VVG the_AT climb_NN1
65.	P-NP-ING / (SUBCAT OC_PP_ING, PFORM @p, SUBTYPE PVERB_OR/OE, ORDER PRENP)
	⇒ XTAG:gap / ERG:v_empty_prep_intrans_le
	ex. they_PPHS2 worried_VVD about_II him_PPHO1 drinking_VVG
66.	P-NP-TO-INF(-SC) / (SUBCAT SC_PP_INF, PFORM @p, SUBTYPE EQUI)
	⇒ XTAG:gap / ERG:v_oeq_pp_inf_le
	ex. he_PPHS1 conspired_VVD with_IW them_PPHO2 to_TO do_VV0 it_PPH1
67.	P-NP-TO-INF-OC / (SUBCAT OC_PP_INF, PFORM @p, SUBTYPE PVERB_OE/OR/EQUI)
	⇒ XTAG:gap / ERG:v_oeq_pp_inf_le
	ex. he_PPHS1 beckoned_VVD to_II him_PPHO1 to_TO come_VV0
68.	P-NP-TO-INF-VC / gap
	⇒ XTAG:gap / ERG:v_oeq_pp_inf_le
	ex. she_PPHS1 appealed_VVD to_II him_PPHO1 to_TO go_VV0
69.	P-POSSING / (SUBCAT OC_PP_ING, PFORM @p, SUBTYPE PVERB_OR, ORDER PRENP)
	⇒ XTAG:gap / ERG:v_empty_prep_intrans_le
	ex. they_PPHS2 argued_VVD about_II his_PP$ coming_VVG
70.	P-WH-S / (SUBCAT WHS, PRT/PREP @p)

	⇒ XTAG:gap / ERG:v_empty_prep_intrans_le ex. he_PPHS1 thought_VVD about_II whether_CSW he_PPHS1 wanted_VVD to_TO go_VV0
71.	P-WHAT-S / (SUBCAT WHS, PRT/PREP @p) ⇒ XTAG:gap / ERG:v_empty_prep_intrans_le ex. he_PPHS1 thought_VVD about_II what_DDQ he_PPHS1 wanted_VVD
72.	P-WH-TO-INF / (SUBCAT WHVP, PREP @p) ⇒ XTAG:gap / ERG:gap ex. he_PPHS1 thought_VVD about_II whether_CSW to_TO go_VV0
73.	P-WHAT-TO-INF / (SUBCAT WHVP, PREP @p) ⇒ XTAG:gap / ERG:v_empty_prep_intrans_le ex. he_PPHS1 thought_VVD about_II what_DDQ to_TO do_VV0
74.	PART / (SUBCAT NULL, PRT) ⇒ XTAG:Tnx0Vpl / ERG:v_particle_le ex. she_PPHS1 gave_VVD up_RL
75.	PART-ING-SC / (SUBCAT SC_ING, SUBTYPE EQUI, PRT/PREP) ⇒ XTAG:gap / ERG:v_particle_np_le ex. he_PPHS1 ruled_VVD out_II paying_VVG her_AT debts_NN2
76.	PART-NP / (SUBCAT NP, PRT) (ORDER FREE) ⇒ XTAG:Tnx0Vplnx1 / ERG:v_particle_np_le ex. I_PPHS1 looked_VVD up_RL the_AT entry_NN1
77.	PART-NP-PP / (SUBCAT NP_PP, PFORM, PRT, SUBTYPE NONE/PVERB?) (ORDER FREE) ⇒ XTAG:gap / ERG:v_particle_np_pp_from_le ex. I_PPHS1 separated_VVD out_II the_AT three_JJ boys_NN2 from_II the_AT crowd_NN1
78.	PART-PP / (SUBCAT PP, PFORM, PRT, SUBTYPE PVERB) ⇒ XTAG:gap / ERG:v_particle_pp_le ex. she_PPHS1 looked_VVD in_II on_II her_AT friend_NN1
79.	PART-WH-S / (SUBCAT WHS, PRT, SUBTYPE NONE) ⇒ XTAG:gap / ERG:v_particle_cp_le ex. they_PPHS2 figured_VVD out_II whether_CSW she_PPHS1 had_VHD n't_XX done_VVD her_AT job_NN1
80.	PART-WHAT-S / (SUBCAT WHS, PRT, SUBTYPE NONE) ⇒ XTAG:gap / ERG:v_particle_cp_le ex. they_PPHS2 figured_VVD out_II what_DDQ she_PPHS1 had_VHD n't_XX done_VVD
81.	PART-WH-TO-INF / (SUBCAT WHVP, PRT, SUBTYPE NONE) ⇒ XTAG:gap / ERG:v_particle_cp_le ex. they_PPHS2 figured_VVD out_II whether_CSW to_TO go_VV0
82.	PART-WHAT-TO-INF / (SUBCAT WHVP, PRT, SUBTYPE NONE) ⇒ XTAG:gap / ERG:v_particle_cp_le ex. they_PPHS2 figured_VVD out_II what_DDQ to_TO do_VV0
83.	PART-THAT-S / (SUBCAT SFIN, PRT, SUBTYPE NONE) ⇒ XTAG:gap / ERG:v_particle_cp_le ex. they_PPHS2 figured_VVD out_II that_CST she_PPHS1 had_VHD n't_XX done_VVD her_AT job_NN1
84.	POSSING / (SUBCAT OC_ING, SUBTYPE RAIS) ⇒ XTAG:gap / ERG:v_np_trans_le ex. he_PPHS1 dismissed_VVD their_PP$ writing_VVG novels_NN2
85.	POSSING-PP / gap

⇒ XTAG:gap / ERG:v_empty_to_trans_le
ex. she_PPHS1 attributed_VVD his_PP$ drinking_VVG too_RA much_RA to_II his_AT anxiety_NN1

86. ING-PP / gap
⇒ XTAG:gap / ERG:v_empty_to_trans_le
ex. they_PPHS2 limited_VVD smoking_VVG a_AT pipe_NN1 to_II the_AT lounge_NN1

87. PP / (SUBCAT PP/LOC, PFORM, SUBTYPE NONE/PVERB)
⇒ XTAG:Tnx0Vpnx1 / ERG:v_empty_prep_intrans_le
ex. they_PPHS2 apologized_VVD to_II him_PPHO1

88. PP-FOR-TO-INF / (SUBCAT PP_SINF, PFORM)
⇒ XTAG:gap / ERG:v_double_pp_le
ex. they_PPHS2 contracted_VVD with_IW him_PPHO1 for_IF the_AT man_NN1 to_TO go_VV0

89. PP-HOW-S / (SUBCAT PP_WHS, PFORM)
⇒ XTAG:gap / ERG:v_pp_cp_le
ex. he_PPHS1 explained_VVD to_II her_PPHO1 how_RGQ she_PPHS1 did_VDD it_PPH1

90. PP-HOW-TO-INF / (SUBCAT PP_WHVP, PFORM)
⇒ XTAG:gap / ERG:v_pp_cp_le
ex. he_PPHS1 explained_VVD to_II them_PPHO2 how_RGQ to_TO do_VV0 it_PPH1

91. PP-P-WH-S / gap
⇒ XTAG:gap / ERG:gap
ex. I_PPHS1 agreed_VVD with_IW him_PPHO1 about_II whether_CSW he_PPHS1 should_VM

92. PP-P-WHAT-S / gap
⇒ XTAG:gap / ERG:v_double_pp_le
ex. I_PPHS1 agreed_VVD with_IW him_PPHO1 about_II what_DDQ he_PPHS1 should_VM do_VV0

93. PP-P-WHAT-TO-INF / gap
⇒ XTAG:gap / ERG:v_double_pp_le
ex. I_PPHS1 agreed_VVD with_IW him_PPHO1 about_II what_DDQ to_TO do_VV0

94. PP-P-WH-TO-INF / gap
⇒ XTAG:gap / ERG:v_double_pp_le
ex. I_PPHS1 agreed_VVD with_IW him_PPHO1 about_II whether_CSW to_TO go_VV0

95. PP-PP / (SUBCAT PP_PP)
⇒ XTAG:gap / ERG:v_double_pp_le
ex. they_PPHS2 flew_VVD from_II London_NP1 to_II Rome_NP1

96. PP-PRED-RS / (SUBCAT PP, SUBTYPE RAIS)
⇒ XTAG:gap / ERG:v_prdp_pp_ssr_a_or_p_le
ex. the_AT matter_NN1 seems_VVZ in_II dispute_NN1

97. PP-THAT-S / (SUBCAT PP_SFIN, SUBTYPE NONE, PFORM)
⇒ XTAG:gap / ERG:v_pp_cp_fin_le
ex. they_PPHS2 admitted_VVD to_II the_AT authorities_NN2 that_CST they_PPHS2 had_VHD entered_VVD illegally_RA

98. PP-THAT-S-SUBJUNCT / (SUBCAT PP_SBSE, PFORM, S[BSE, that])
⇒ XTAG:gap / ERG:v_pp_cp_le
ex. they_PPHS2 suggested_VVD to_II him_PPHO1 that_CST he_PPHS1 go_VV0

99. PP-TO-INF-RS / (SUBCAT SC_PP_INF, SUBTYPE RAIS, PFORM, VP[to])
⇒ XTAG:gap / ERG:v_pp_inf_ssr_le

ex. he_PPHS1 appeared_VVD to_II her_PPHO1 to_TO be_VB0 ill_JJ

100. PP-WH-S / (SUBCAT PP_WHS, PFORM)
⇒ XTAG:gap / ERG:gap
ex. they_PPHS2 asked_VVD about_II everybody_NP1 whether_CSW they_PPHS2 had_VHD enrolled_VVN

101. PP-WHAT-S / (SUBCAT PP_WHS, PFORM)
⇒ XTAG:gap / ERG:gap
ex. they_PPHS2 asked_VVD about_II everybody_NP1 what_DDQ they_PPHS2 had_VHD done_VVN

102. PP-WH-TO_INF / (SUBCAT PP_WHVP)
⇒ XTAG:gap / ERG:v_pp_cp_le
ex. they_PPHS2 deduced_VVD from_II kim_NP1 whether_CSW to_TO go_VV0

103. PP-WHAT-TO-INF / (SUBCAT PP_WHVP)
⇒ XTAG:gap / ERG:v_pp_cp_le
ex. they_PPHS2 deduced_VVD from_II kim_NP1 what_DDQ to_TO do_VV0

104. S / (SUBCAT SFIN, SUBTYPE NONE)
⇒ XTAG:Tnx0Vs1 / ERG:v_cp_prop_non_trans_le
ex. they_PPHS2 thought_VVD that_CST he_PPHS1 was_VBZ always_RA late_JJ

105. S-SUBJ-S-OBJ / (SUBCAT SFIN, SUBTYPE EXTRAP, AGR S[FIN -])
⇒ XTAG:gap / ERG:v_cp_prop_non_trans_le
ex. for_IF him_PPHO1 to_TO report_VV0 the_AT theft_NN1 indicates_VVD that_CST he_PPHS1 was_VBZ n't_XX guilty_JJ

106. S-SUBJUNCT / (SUBCAT SBSE)
⇒ XTAG:gap / ERG:v_cp_subj_le
ex. She_PPHS1 demanded_VVD that_CST he_PPHS1 leave_VV0 immediately_RA

107. SEEM-S / (SUBCAT SFIN, SUBTYPE EXTRAP, AGR N2[NFORM IT])
⇒ XTAG:gap / ERG:v_expl_it_subj_pp_cp_le
ex. it_PPH1 seems_VVZ that_CST they_PPHS2 left_VVD

108. SEEM-TO-NP-S / (SUBCAT PP_SFIN, SUBTYPE EXTRAP, PFORM, AGR N2[NFORM IT])
⇒ XTAG:gap / ERG:v_expl_it_subj_pp_cp_le
ex. it_PPH1 seems_VVZ to_II her_PPHO1 that_CST they_PPHS2 were_VBDR wrong_JJ

109. THAT-S / (SUBCAT SFIN, SUBTYPE NONE)
⇒ XTAG:Tnx0Vs1 / ERG:v_expl_it_subj_pp_cp_le
ex. he_PPHS1 complained_VVD that_CST they_PPHS2 were_VBDR coming_VVG

110. TO-INF-AC / gap
⇒ XTAG:gap / ERG:v_subj_equi_le
ex. He_PPHS1 helped_VVD to_TO save_VV0 the_AT child_NN1

111. TO-INF-RS / (SUBCAT SC_INF, SUBTYPE RAIS)
⇒ XTAG:gap / ERG:v_ssr_le
ex. he_PPHS1 seemed_VVD to_TO come_VV0

112. TO-INF-SC / (SUBCAT SC_INF, SUBTYPE EQUI)
⇒ XTAG:gap / ERG:v_subj_equi_le
ex. I_PPHS1 wanted_VVD to_TO come_VV0

113. WH-S / (SUBCAT WHS)
⇒ XTAG:Tnx0Vs1 / ERG:v_cp_ques_fin_inf_non_trans_le
ex. he_PPHS1 asked_VVD whether_CSW he_PPHS1 should_VM come_VV0

114. WHAT-S / (SUBCAT WHS)
⇒ XTAG:Tnx0Vs1 / ERG:v_cp_ques_fin_inf_non_trans_le

ex. he_PPHS1 asked_VVD what_DDQ he_PPHS1 should_VM do_VV0

115. WH-TO-INF / (SUBCAT WHVP)
⇒ XTAG:Tnx0Vs1 / ERG:v_cp_ques_fin_inf_non_trans_le
ex. he_PPHS1 asked_VVD whether_CSW to_TO clean_VV0 the_AT house_NN1

116. WHAT-TO-INF / (SUBCAT WHVP)
⇒ XTAG:Tnx0Vs1 / ERG:v_cp_ques_fin_inf_non_trans_le
ex. he_PPHS1 asked_VVD what_DDQ to_TO do_VV0

117. XTAG:Tnx0Vplnx1nx2 / (SUBCAT NP_NP, PRT)
⇒ XTAG:Tnx0Vplnx1nx2 / ERG:gap
ex. I_PPHS1 opened_VVD him_PPHO1 up_RP a_AT new_JJ bank_NN1 account_NN1

118. XTAG:Light-verbs (various classes) / gap
⇒ XTAG:Light-verbs / ERG:v_np_prep_trans_le
ex. he_PPHS1 made_VVD comments_NN2 on_II the_AT paper_NN1

119. gap / (SUBCAT PP/LOC, PFORM, PRT, SUBTYPE NONE)
⇒ XTAG:gap / ERG:v_particle_pp_le
ex. he_PPHS1 breaks_VVZ away_RP from_II the_AT abbey_NN1

120. gap / (SUBCAT NP_PP, PFORM, PRT, SUBTYPE DMOVT)
⇒ XTAG:gap / ERG:v_np_particle_pp_for_le
ex. he_PPHS1 brought_VVD a_AT book_NN1 back_RP for_IF me_PPHO1

121. gap / (SUBCAT PP_PP, PFORM, PRT)
⇒ XTAG:gap / ERG:gap
ex. he_PPHS1 came_VVD down_RP on_II him_PPHO1 for_IF his_AT bad_JJ behaviour_NN1

122. gap / (SUBCAT NP_PP_PP, PFORM)
⇒ XTAG:gap / ERG:v_np_trans_double_pp*_le
ex. he_PPHS1 turned_VVD it_PPHO1 from_II a_AT disaster_NN1 into_II a_AT victory_NN1

123. gap / (SUBCAT MP)
⇒ XTAG:gap / ERG:v_np_non_trans_le
ex. it_PPHS1 cost_VVD ten_MC pounds_NNU2

124. gap / (SUBCAT NP_MP)
⇒ XTAG:gap / ERG:v_ditrans_only_le
ex. it_PPHS1 cost_VVD him_PPHO1 ten_MC pounds_NNU2

125. gap / (SUBCAT NP_MP, PRT)
⇒ XTAG:gap / ERG:gap
ex. it_PPHS1 set_VVD him_PPHO1 back_RP ten_MC pounds_NNU2

126. gap / (SUBCAT ADL, PRT)
⇒ XTAG:gap / ERG:gap
ex. he_PPHS1 came_VVD off_RP badly_RP

127. gap / (SUBCAT ADV_PP, PFORM)
⇒ XTAG:gap / ERG:gap
ex. things_NN2 augur_VV0 well_RP for_IF him_PPHO1

128. gap / (SUBCAT SFIN, AGR N2[NFORM IT], PRT)
⇒ XTAG:gap / ERG:v_expl_it_subj_prtcl_cp_le
ex. it_PPHS1 turns_VVZ out_RP that_CST he_PPHS1 did_VVD it_PPHO1

129. gap / (SUBCAT SFIN, AGR S[FIN +], SUBTYPE EXTRAP)
⇒ XTAG:gap / ERG:v_unerg_le

ex. that_CST he_PPHS1 came_VVD matters_VVZ

130. gap / (SUBCAT NP_SFIN, SUBTYPE NONE, PRT)
⇒ XTAG:gap / ERG:gap
ex. he_PPHS1 had_VVD her_PPHO1 on_RP that_CST he_PPHO1 attended_VVD

131. gap / (SUBCAT PP_SFIN, SUBTYPE NONE, PRT)
⇒ XTAG:gap / ERG:gap
ex. she_PPHS1 gets_VVZ through_RP to_II him_PPHO1 that_CST he_PPHS1 came_VVD

132. gap / (SUBCAT NP_NP_SFIN)
⇒ XTAG:gap / ERG:v_np_np_cp_le
ex. he_PPHS1 bet_VVD her_PPHO1 ten_MC pounds_NNU2 that_CST he_PPHS1 came_VVD

133. gap / (SUBCAT NP_SBSE)
⇒ XTAG:gap / ERG:gap
ex. he_PPHS1 petitioned_VVD them_PPHO2 that_CST he_PPHS1 be_VB0 freed_VVN

134. gap / (SUBCAT IT_WHS, SUBTYPE IF, AGR N2[NFORM IT])
⇒ XTAG:gap / ERG:v_expl_obj_cp_le
ex. I_PPHS1 would_VM appreciate_VV0 it_PPHO1 if_CF he_PPHS1 came_VVD

135. gap / (SUBCAT PP_WHS, PFORM, AGR N2[NFORM IT])
⇒ XTAG:gap / ERG:v_expl_it_subj_pp_cp_le
ex. it_PPHS1 dawned_VVD on_II him_PPHO1 what_DDQ he_PPHS1 should_VM do_VV0

136. gap / (SUBCAT SC_NP, PRT, SUBTYPE RAIS/EQUI, PRD +)
⇒ XTAG:gap / ERG:gap
ex. he_PPHS1 turned_VVD out_RP a_AT fool_NN1

137. gap / (SUBCAT SC_AP, PRT, SUBTYPE EQUI/RAIS)
⇒ XTAG:gap / ERG:v_particle_prd_le
ex. he_PPHS1 started_VVD out_RP poor_JJ

138. gap / (SUBCAT SC_INF, PRT, SUBTYPE RAIS)
⇒ XTAG:gap / ERG:v_particle_inf_le
ex. he_PPHS1 turned_VVD out_RP to_TO be_VB0 a_AT crook_NN1

139. gap / (SUBCAT SC_INF, PRT, SUBTYPE EQUI)
⇒ XTAG:gap / ERG:v_particle_inf_le
ex. he_PPHS1 set_VVD out_RP to_TO win_VV0

140. gap / (SUBCAT SC_ING, PREP, PRT, SUBTYPE EQUI)
⇒ XTAG:gap / ERG:v_particle_inf_le
ex. he_PPHS1 got_VVD around_RP to_II leaving_VVG

141. gap / (SUBCAT SC_PASS, SUBTYPE RAIS)
⇒ XTAG:gap / ERG:v_subj_equi_prd_le
ex. he_PPHS1 got_VVD given_VVN a_AT book_NN1

142. gap / (SUBCAT SC_BSE, SUBTYPE EQUI)
⇒ XTAG:gap / ERG:gap
ex. he_PPHS1 dared_VVD dance_VV0

143. gap / (SUBCAT SC_NP_AP, SUBTYPE RAIS, PREP as)
⇒ XTAG:gap / ERG:v_np_obj_comp_le
ex. he_PPHS1 strikes_VVZ me_PPHO1 as_CSA foolish_JJ

144. gap / (SUBCAT OC_NP, SUBTYPE RAIS)
⇒ XTAG:gap / ERG:v_ditrans_only_le
ex. he_PPHS1 considers_VVZ Fido_NP1 a_AT fool_NN1

145. gap / (SUBCAT OC_AP, SUBTYPE RAIS, PRT)
⇒ XTAG:gap / ERG:gap

```
      ex. he_PPHS1 makes_VVD him_PPHO1 out_RP crazy_JJ
146.  gap / (SUBCAT OC_AP, SUBTYPE EQUI, PRT)
      ⇒ XTAG:gap / ERG:gap
      ex. he_PPHS1 sands_VVZ it_PPHO1 down_RP smooth_JJ
147.  gap / (SUBCAT OC_AP, SUBTYPE EQUI, PREP as)
      ⇒ XTAG:gap / ERG:v_np_obj_comp_le
      ex. he_PPHS1 condemned_VVD him_PPHO1 as_CSA stupid_JJ
148.  gap / (SUBCAT OC_AP, SUBTYPE EQUI, PREP as, PRT)
      ⇒ XTAG:gap / ERG:gap
      ex. he_PPHS1 put_VVD him_PPHO1 down_RP as_CSA stupid_JJ
149.  gap / (SUBCAT OC_INF, SUBTYPE RAIS, PRT)
      ⇒ XTAG:gap / ERG:gap
      ex. he_PPHS1 made_VVD him_PPHO1 out_RP to_TO be_VV0 crazy_JJ
150.  gap / (SUBCAT OC_INF, SUBTYPE EQUI, PRT)
      ⇒ XTAG:gap / ERG:gap
      ex. he_PPHS1 spurred_VVD him_PPHO1 on_RP to_TO try_VV0
151.  gap / (SUBCAT OC_PP_INF, SUBTYPE PVERB_OE, PFORM, PRT)
      ⇒ XTAG:gap / ERG:gap
      ex. he_PPHS1 kept_VVD on_RP at_II him_PPHO1 to_TO join_VV0
152.  gap / (SUBCAT OC_PP_ING, SUBTYPE PVERB_OE, PFORM, PRT)
      ⇒ XTAG:gap / ERG:gap
      ex. he_PPHS1 talked_VVD him_PPHO1 around_RP into_II leaving_VVG
153.  gap / (SUBCAT OC_PP_BSE, PFORM, SUBTYPE PVERB_OE)
      ⇒ XTAG:gap / ERG:gap
      ex. he_PPHS1 looked_VVD at_II him_PPHO1 leave_VV0
154.  gap / (SUBCAT VPINF, SUBTYPE EXTRAP, AGR VP[FIN-])
      ⇒ XTAG:gap / ERG:v_unerg_le
      ex. to_TO see_VV0 them_PPHO2 hurts_VVZ
155.  gap / (SUBCAT NP_ADL)
      ⇒ XTAG:gap / ERG:gap
      ex. he_PPHS1 stood_VVD it_PPHO1 alone_RL
156.  *NP-HOW-S / gap
      ⇒ XTAG:gap / ERG:v_np_trans_cp_ques_le
      ex. he_PPHS1 asked_VVD him_PPHO1 how_RGQ he_PPHS1 came_VVD
157.  *NP-FOR-TO-INF / gap
      ⇒ XTAG:gap / ERG:gap
      ex. he_PPHS1 gave_VVD money_NN2 for_IF him_PPHO1 to_TO go_VV0
158.  *IT-PASS-SFIN / gap
      ⇒ XTAG:gap / ERG:gap
      ex. it_PPHS1 is_VBZ believed_VVN that_CST he_PPHS1 came_VVD
159.  *AS-IF-SFIN / gap
      ⇒ XTAG:gap / ERG:gap
      ex. he_PPHS1 seems_VVZ as_CS if_CS he_PPHS1 is_VBZ clever_JJ
160.  gap / (SUBCAT ADL)
      ⇒ XTAG:gap / ERG:v_adv_le
      ex. it_PPHS1 carves_VVZ easily_RP
161.  gap / (SUBCAT SC_NP SUBTYPE EQUI)
      ⇒ XTAG:gap / ERG:gap
```

	ex. he_PPHS1 felt_VVD a_AT fool_NN1
162.	*AS-VPPRT / gap ⇒ XTAG:gap / ERG:v_np_comp_le ex. he_PPHS1 accepted_VVD him_PPHO1 as_II/CSA associated_VVN
163.	*AS-VPING / gap ⇒ XTAG:gap / ERG:v_np_comp_le ex. he_PPHS1 accepted_VVD him_PPHO1 as_II/CSA being_VBG normal_JJ

Index