SEMANTIC PARSING USING LEXICALIZED WELL-FOUNDED GRAMMARS

by

GAURAV KHARKWAL

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Computer Science

Written under the direction of

Dr. Matthew Stone

and approved by

_____

_____

_____

New Brunswick, New Jersey

January, 2014

ABSTRACT OF THE THESIS

Semantic Parsing using Lexicalized Well-Founded Grammars

By GAURAV KHARKWAL

Thesis Director:

Dr. Matthew Stone

Research in semantic parsing has focused on developing computational systems capable of simultaneously performing syntactic, i.e. structural, and semantic, i.e., meaning-based, analyses of given sentences. We present an implementation of a semantic parsing system using a constraint-based grammatical formalism called Lexicalized Well-Founded Grammars (LWFGs). LWFGs are a type of Definite Clause Grammars, and use an ontology-based framework to represent syntactico-semantic information in the form of compositional and interpretation constraints. What makes LWFGs particularly interesting is the fact that these are the only constraint-based grammars that are provably learnable. Furthermore, there exist tractable learning algorithms for LWFGs, which make these especially useful in resource-poor language settings.

In this thesis, we present a revised parsing implementation for Lexicalized Well-Founded Grammars. Previous work implemented semantic parsers using Prolog, a declarative language, which is slow and does not allow for an easy extension to a stochastic parsing framework. Our implementation utilizes Python's Natural Language Toolkit which not only allows us to easily interface our work with the natural language processing community, but also allows for a future possibility of extending the parser to support broad-coverage and stochastic parsing.

Table of Contents

List of Tables

List of Figures

1. Introduction

A long-term goal of any research endeavor in natural language understanding is to develop an autonomous system that can interpret natural language utterances. Semantic parsing is a sub-area of natural language understanding where the specific goal is to produce a computational system that is capable of analyzing written sentences and generating an abstract representation of its "meaning." Here, meaning is usually operationalized based on the intended application, but for most applications, being able to identify "who did what to whom (when where and how)" fulfills the criterion.

In recent history, three different approaches have been taken to tackle this problem. One such attempt is called *shallow semantic parsing*, or *semantic role labeling*. Here, instead of coming up with a detailed meaning representation of sentences, the objective is to label words and phrases with *semantic roles*. These semantic roles specify the relation between the word and the predicate of the sentence (i.e., the main verb). Work in this domain has been largely motivated by Gildea and Jurafsky (2002), who presented a statistical machine-learning approach to developing shallow semantic parsing systems. Since then, many other researchers have used similar techniques to further develop and refine such systems (e.g., Chen and Rambow, 2003; Fleischman et al., 2003; Xue and Palmer, 2004; Pradhan, 2006; Surdeanu et al., 2007; Toutanova et al., 2008; Vickrey, 2010).

However, as mentioned before, the goal of shallow semantic parsing systems is not to generate a full meaning representation of a sentence. Another approach has therefore been to use statistical machine learning techniques to develop semantic parsers which are capable of generating more detailed meaning representations (e.g., Ge and Mooney, 2005; Wong and Mooney, 2007; Lu et al., 2008; Poon and Domingos, 2009; Zettlemoyer and Collins, 2009; Kwiatkowski et al., 2010). In this approach, a set of sentences are manually annotated with a "meaning representation language." Subsequently, techniques similar to those used for statistical machine translation are used to perform alignment between the syntactic structures and the meaning representation structures of the sentences. Lastly, machine-learning techniques are used to develop

synchronous grammars to simultaneously perform a syntactic and a semantic analysis on sentences.

One conceptual limitation of such an approach is the divorce between the underlying grammatical formalism and the meaning representation language. As such, this approach tends to be agnostic of linguistic research on semantics. Thus, another approach often taken is to use linguistic constraint-based grammatical formalisms that represent both syntactic and semantic properties of the language as constraints in the grammar. Examples of such formalisms include Lexical Functional Grammar (Bresnan, 1982, 2001), Head-driven Phrase Structure Grammar (Pollard and Sag, 1987, 1994), and Generalized Phrase Structure Grammar (Gazdar, 1985). Essentially, semantic parsing systems built using such formalisms are simply parsing systems where a semantic interpretation naturally follows from a syntactic derivation of a sentence.

1.1 Thesis Contributions

In this thesis, we present an approach to generate an abstraction of meaning given an input sentence. Our approach utilizes an ontology-based grammatical framework called Lexicalized Well-Founded Grammar (Muresan, 2006; Muresan and Rambow, 2007; Muresan, 2010, 2011). The framework is similar in principle to other constraint-based grammatical formalisms, and provides an effective way of representing meaning at a lexical level and combining these lexical meaning representations to generate a meaning representation for the whole sentence.

What makes this formalism particularly interesting is the fact that, unlike other such formalisms, Lexicalized Well-Founded Grammars have been shown to be provably learnable from data (Muresan and Rambow, 2007). This, together with the fact that there exist tractable learning algorithms for the formalism (Muresan and Rambow, 2007; Muresan, 2010), make Lexicalized Well-Founded Grammars a better choice for applications in resource-poor settings where statistical semantic parsing is ill-suited.

The work presented in this thesis builds upon an earlier work by Muresan (2006, 2010) to present a processing system (i.e., a semantic parser) that can generate

a sentence-level meaning representation given a Lexicalized Well-Founded Grammar. Specifically, we present a revised implementation of their algorithm in an imperative programming language. We justify our approach in two ways: 1) Using a programming language like Python, which supports a large open-source natural language processing community, allows us to interface our work with the rest of the community. In addition, it allows us to use and extend other NLP resources supported by Python. 2) The nature of the programming language will allow us to more easily extend our work to support broad-coverage and stochastic parsing. This is unlike Prolog which would have proved to be a challenge due to its not being naturally suited for numerical programming

We evaluate our implementation using random sentences generated using a sample Lexicalized Well-Founded Grammar. We further compared our parser with a similar parsing system that instead uses a context-free grammar. Not unexpectedly, the CFG parser proved much faster than ours. However, our parser was able to generate a semantic representation for every sentence that we tested. Furthermore, our parser provided significantly less derivations per sentence than the CFG parser (i.e., there was less ambiguity per sentence), which is so because cases of a mismatch in grammatical properties like number, case, etc. are detected early and removed from consideration.

1.2 Thesis Outline

The thesis is organized as follows:

- Chapter 2 presents a review of previous research on semantic parsing, and sets up the background for this thesis.
- Chapter 3 provides a detailed review of Lexicalized Well-Founded Grammars, and the parsing algorithm developed by Muresan (2006).
- Chapter 4 presents our implementation of the LWFG parsing algorithm. It begins with a discussion on the motivation behind our work. That is followed by a detailed description of the data structures used and the actual implementation. It ends with a discussion on complexity analysis.

- Chapter 5 presents an empirical evaluation of our parser.
- Chapter 6 discusses directions for future research.
- Chapter 7 offers our final conclusion.

2. Related Work

In this chapter, we present a discussion on other attempts at developing semantic parsers. Research in semantic parsing can be broadly classified into three categories which are not necessarily mutually exclusive: 1) shallow semantic parsing; 2) statistical semantic parsing; 3) formalism-based semantic parsing. We next attempt to provide a broad overview of the three different approaches.

2.1 Shallow semantic parsing

Shallow semantic parsing is also referred to as semantic role labeling, and the task essentially involves labeling phrases of a given sentence with semantic roles (also known as semantic arguments). These semantic roles are often associated with a target word, which is usually the predicate or main verb of the sentence or clause. Essentially, the task involves identifying each entity, abstract or otherwise, and labeling it with some information specific to the role it plays in the event described by the main verb. For example, in the sentence *the boy kissed the girl*, *the boy* is the AGENT, *the girl* is the PATIENT, and the event is KISS. The choice of actual semantic role labels can often vary depending on the task and resources at hand, and while our example used generic role labels, it is also possible for the entities to be described more specifically. For example, *the boy* may be labeled as the KISSER and *the girl* as the KISSED.

Early work on automatic semantic role labeling had mostly relied on utilizing syntactic structures to derive semantic interpretations (e.g., Warren and Friedman, 1982; Hirst, 1983; Sondheimer et al., 1984; see Pradhan, 2006 for a review). Subsequent work was greatly facilitated by the availability of annotated resources such as FrameNet (Baker et al., 1998), PropBank (Palmer et al., 2005), NomBank (Meyers et al., 2004), and VerbNet (Schuler, 2005). The availability of such resources led to a shift in the approach towards semantic role labeling with the emphasis now being on using trained corpora to develop statistical systems that could perform labeling accurately. The seminal work was done by Gildea and Jurafsky (2002) in which they used about 50,000 sentences hand-annotated with roles from FrameNet to train a statistical classifier to

perform automatic semantic role labeling. Following their work, recent attempts have used a conceptually similar framework to develop high-performing systems. The idea has been to use state-of-the-art statistical techniques and clever feature-engineering to improve upon performance (e.g., Chen and Rambow, 2003; Fleischman et al., 2003; Xue and Palmer, 2004; Pradhan, 2006; Surdeanu et al., 2007; Toutanova et al., 2008; Vickrey, 2010).

Semantic role labeling has recently been shown to be a fairly useful component in the natural language processing pipeline, and is often used as a pre-processing step in many higher-level applications. For example, semantic role labeling has been used for coreference resolution (Ponzetto and Strube, 2006), question-answering systems (Narayanan and Harabagiu, 2004; Shen and Lapata, 2007), information extraction (Surdeanu et al., 2003; Barnickel et al., 2009; Christensen et al., 2010), and machine translation (Boas, 2002).

However, semantic role labeling does have some limitations. Semantic roles are sufficient to identify relations between various entities in a sentence, but they cannot fully represent the meaning of a sentence. For example, consider the following sentence *the boys like to play*, and *the boy likes to play*. A semantic role labeling system will correctly identify *the boys* and *the boy* as being the entities that LIKE *to play*. However, it would be unable to notice that *the boys* is plural form of *the boy*, or that *like* and *likes* are both forms of the same verb LIKE, their forms differing only because of grammatical agreement. Thus, for applications where a complete meaning representation of a sentence is desired, semantic role labeling systems fail to be adequate.

2.1.1 Statistical semantic parsing

Many researchers have thus undertaken significant effort to propose approaches for generating (near-) complete meaning representations for sentences. One line of work has attempted to solve this problem by using sophisticated machine learning techniques to train statistical systems for semantic parsing (e.g., Ge and Mooney, 2005; Wong and Mooney, 2007; Lu et al., 2008; Poon and Domingos, 2009; Zettlemoyer and Collins,

2009; Kwiatkowski et al., 2010). The common thread in this line of work is to use corpora containing sentences/phrases annotated with a *meaning representation language*, and to use supervised or unsupervised learning methods to derive statistical systems. These *meaning representations* are, more often than not, not integrated with any underlying grammatical (i.e. syntactic) formalism, and thus these systems often have to also learn an alignment between the meaning representational language and the grammatical formalism.

A caveat here, while these meaning representation languages tend to capture greater semantic detail than shallow semantic representations, they also tend not be a complete semantic representation. As such, these languages tend to be agnostic of linguistic research in semantics, and various aspects of meaning tied to things like the discourse, real-world ontologies, and interpretive dependencies fail to be adequately captured. However, while these representations have theoretical limitations, they tend to be well-suited for many applications, and thus offer a compromise between what ought to be and what can be.

A considerably large subset of researchers, led mostly by Raymond Mooney, have attempted to use techniques derived from statistical machine translation to develop semantic parsing systems that map natural language phrases to a logical meaning representational languages (e.g., Ge and Mooney, 2005; Wong and Mooney, 2006, 2007; Lu et al., 2008; Chen and Mooney, 2008; Liang et al., 2009; Kim and Mooney, 2012). Their work has largely considered domain-specific meaning representations. For example, they have often used a corpus derived from instructions in the RoboCup tournament. The instructions are coded in a formal language called CLang, an example of which is presented below:

```
((bpos (penalty-area our))
 (do (player-except our {4})
     (pos (half our))))
```

The example instruction corresponds to the sentence: *If the ball is in our penalty area, all our players except player 4 should stay in our half.* The common thread in

their work has been to first develop an alignment and then a mapping between the underlying syntactic structure of a sentence and its logical meaning representation. Often, the attempt is to develop a generational system that can simultaneously derive natural language phrases and meaning representations. These systems are then trained on annotated corpora to develop statistical semantic parsing systems.

For example, Ge and Mooney (2005) describe a system called SCISSOR (Semantic Composition that Integrates Syntax and Semantics to get Optimal Representations) that builds upon Collins' head-driven model 2 (Collins, 1997). The system modified Collins' model by incorporating semantic labels into non-terminals. In addition, the system was suitably modified to label each node of a sentence's derivation tree with the semantic label of the head child, along with the head word and its POS tag. Lastly, the system was trained on a corpus of manually-annotated natural language sentences to develop a probabilistic generative system that decodes a sentence's meaning by finding its most likely derivation.

The WASP (Word Alignment-based Semantic Parsing) system of Wong and Mooney (2006) is conceptually similar to SCISSOR and was constructed in two stages. First, they used GIZA++ (Och and Ney, 2003; Brown et al., 1993) to perform an alignment between natural language phrases and the corresponding meaning representation. Next, they constructed a synchronous context-free grammar whose rules are of the form:

$$A \rightarrow \langle \alpha, \beta \rangle$$

Where α is the natural language phrase and β is its corresponding meaning representational translation. Finally, WASP is essentially a maximum-entropy model over the synchronous context-free grammar.

Lu et al. (2008) used the same training set as Wong and Mooney (2006) to describe a slightly different approach. Unlike earlier work that essentially relied on using the syntactic structure of natural language sentences to define a correspondence between the sentence and its meaning representation, their work used components of the grammatical structure of the meaning representation language instead. They defined

and used *hybrid trees*, where meaning representation rules constitute internal nodes, whereas natural language words/phrases constitute the leaves. They further defined a Markovian generative process based on these hybrid trees to generate natural language sentences using meaning representation rules. Their semantic parsing model essentially used a joint probability distribution of generating hybrid trees given sentences and meaning representations to estimate the likelihood of a given sentence having some meaning representation, and chose the most likely meaning representation.

The works described earlier have all been under fully-supervised settings, where each sentence is annotated with its corresponding meaning representation. Recent work along similar lines has been attempting to extend these models to be able to learn from ambiguous supervision, where sentences are annotated with a set of potential meaning representations. For example, Chen and Mooney (2008) extended WASP using an EM-like retraining system that iteratively improves upon the selection of correct natural language and meaning representation pairs. Along similar veins, Kim and Mooney (2012) extended the hybrid tree approach of Lu et al. (2008) to handle ambiguous supervision. Their approach is inspired by that of Liang et al. (2009), and uses a two-step generative approach to map natural language to its meaning representation. Their generative model first chooses an event to be described in a given world state (comprising meaning representations), and then models the probability of getting a natural language sentence from a meaning representation specified by the event.

Another line of work has attempted to map natural language sentences to lambda-calculus encodings of their meaning (Zettlemoyer and Collins, 2007, 2009, 2012; Kwiatkowski et al., 2010). Lambda-calculus encodings allow for a domain-independent representation of meaning, and thus is more easily generalizable. For example, lambda-calculus based meaning representation of the sentence *List flights to Boston on Friday night* would be:

$$\lambda x.flight(x) \wedge to(x, Boston) \wedge day(x, Friday) \wedge during(x, night)$$

Their approach has relied on using Combinatory Categorial Grammar (Steedman, 1996,

2001). CCG is a lexicalized, mildly context-sensitive grammatical formalism, and offers an integrated treatment of syntax and semantics making the use of compositional semantics based on lambda calculus. Thus, CCG is well-suited to develop a system to map natural language to a lambda-calculus based meaning representation.

The core of CCG is the lexicon, and in its purely syntactic form, the lexicon consists of words and their syntactic types. An example lexicon is as follows:

$$\begin{aligned} \text{flights} &:= N \\ \text{to} &:= (N\backslash N)/NP \\ \text{Boston} &:= NP \end{aligned}$$

CCG also has a set of *combinatory rules* which describe how adjacent syntactic categories in a string can be recursively combined. For example, the following set of rules are called *functional application rules*:

$$\begin{aligned} A/B \quad B &\implies A \\ B \quad A\backslash B &\implies A \end{aligned}$$

Thus, an intuitive interpretation of complex syntactic categories of the form $A\backslash B$ and A/B is that these are categories of type A which are missing a string of type B to the left and right respectively. The work by Zettlemoyer and Collins extends these CCGs to incorporate lambda-calculus into the lexicon and the combinatory rules. For example, a revised lexicon is shown below:

$$\begin{aligned} \text{flights} &:= N : \lambda x.flight(x) \\ \text{to} &:= (N\backslash N)/NP : \lambda y.\lambda f.\lambda x.f(x) \wedge to(x, y) \\ \text{Boston} &:= NP : Boston \end{aligned}$$

A modified example of the functional application rules is shown below:

$$A/B : f \quad B : g \implies A : f(g)$$

$$B : g \quad A\backslash B : f \implies A : f(g)$$

Thus, whenever a complex category is combined with its missing type, its lambda function is applied to the expression of the subsumed category. For example, the derivation of the phrase *flights to Boston* is shown below:

$$\dfrac{\dfrac{\text{flights}}{N : \lambda x.flight(x)} \quad \dfrac{\dfrac{\text{to}}{(N\backslash N)/NP : \lambda y.\lambda f.\lambda x.f(x) \wedge to(x,y)} \quad \dfrac{\text{Boston}}{NP : boston}}{(N\backslash N) : \lambda f.\lambda x.f(x) \wedge to(x, Boston)}}{N : \lambda x.flight(x) \wedge to(x, Boston)}$$

The work by Zettlemoyer and Collins (2007, 2012) has essentially built upon this idea to develop a probabilistic-CCG parser that can simultaneously derive the lambda-calculus representation of words. The basic idea is to first derive a lexicon using an annotated corpus mapping sentences and their logical forms. Subsequently, a log-linear model is derived to define a conditional distribution of the form $P(L, T|S)$, where S is the sentence, L is the logical form, and T is the sequence of steps taken to derive L. The semantic parser chooses the most-likely parse to generate the logical form of a given sentence. The general idea has since been refined to be able to handle context-dependent interpretation of sentences (Zettlemoyer and Collins, 2007), and to handle fairly-large space of possible grammars (Kwiatkowski et al., 2010).

2.1.2 Formalism-based Semantic Parsing

Another approach to generating (near-) complete meaning representations of sentences has relied on using grammatical formalisms that have been explicitly developed for deep language understanding. This line of work differs from the other in that there is no divorce between the natural language grammatical structure and meaning representational language, and as such, researchers here tend to be more open to linguistic theories. The approach relies on using constraint-based grammatical formalisms which directly encode syntactico-semantic constraints as features in the syntactic grammar

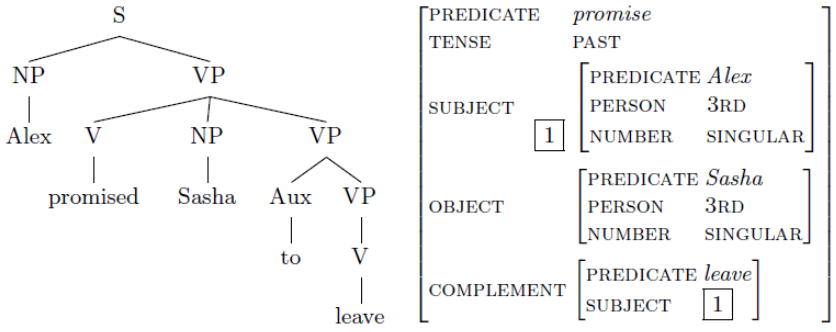


Figure 2.1: The syntactic and semantic structures from Lexical Functional Grammars for the sentence *Alex promised Sasha to leave*. Example taken from Johnson (2003)

itself. These formalisms may be based on linguistic theories, such as Lexical Functional Grammar (Bresnan, 1982, 2001), Head-driven Phrase Structure Grammar (Pollard and Sag, 1987, 1994), and Generalized Phrase Structure Grammar (Gazdar, 1985). On the other hand, it is also possible for the formalisms to be linguistic theory-neutral and be purely computational tools, such as Functional Unification Grammars (Kay, 1984), Definite Clause Grammars (Pereira and Warren, 1980), and PATR-II (Shieber et al., 1983).

All such constraint-based formalisms differ in how they actually represent the syntactico-semantic constraints. For example, Lexical Function Grammars often use several different structures to represent different aspects of a sentence's syntactic and semantic structures. In Figure 2.1, we see two example structures for the sentence *Alex promised Sasha to leave*. The structure on the left, called a c-structure, represents the syntactic structure of the sentence (and is not unlike regular phrase structure trees), whereas the one on the right, called an f-structure, represents the semantic structure of the sentence using an attribute value matrix. On the other hand, Head-driven Phrase Structure Grammars offer a very different representation, and use a lexicalist approach to represent syntactic and semantic information. Figure 2.2 shows an example lexical entry for the word *loves*. As we can see, the representation is a rich feature structure, and HPSG parsing involves unifying such feature structures across words and phrases.

Again, the idea in principle is to encode all necessary linguistic constraints into the definition of the syntactic grammar itself, where constraints could be syntactic as

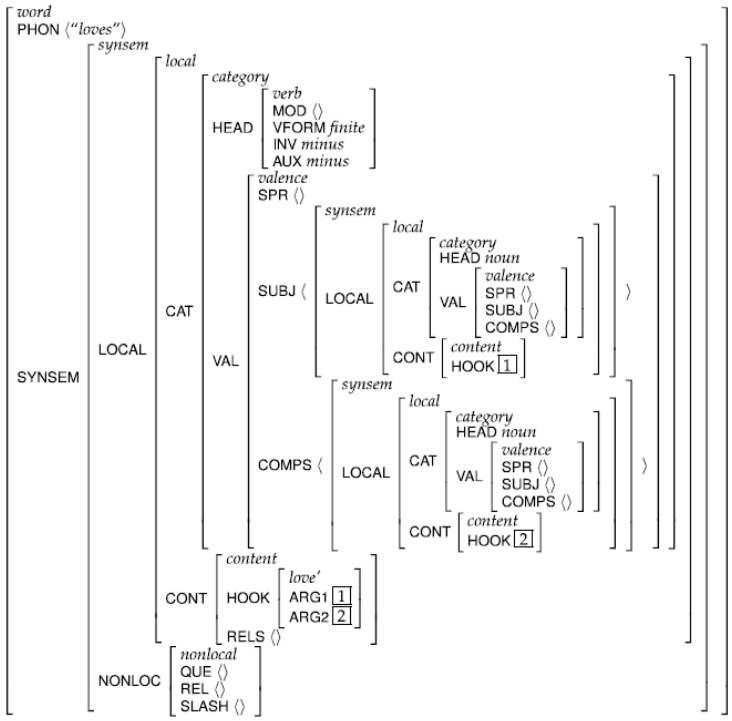


Figure 2.2: A Head-drive Phrase Structure Grammar entry for the word loves. Example taken from Miyao and Tsujii (2008)

well as semantic. Then, given a new sentence, a rule-by-rule syntactic derivation also leads to a simultaneous semantic analysis of the sentence. There is a rich history of work in developing constraint-based grammars and algorithms to parse using those. In recent history, the impetus has diverted slightly to focus on developing probabilistic routines for parsing using these grammars (e.g., Abney, 1997; Johnson et al., 1999; Geman and Johnson, 2002; Johnson, 2003; Miyao and Tsujii, 2002, 2005, etc.)

The pioneering work in the domain was by Abney (1997), where they proposed a general framework for estimating probabilities for constraint-based grammars. They noted that unlike context-free grammars which produced parse trees following syntactic derivations, constraint-based grammars produced dags. Thus, they proposed using Markov Random fields (or log-linear models) to estimate a probability distribution over the grammars, as opposed to the maximum likelihood estimation (i.e., rule counting) method often used for context-free grammars.

The idea was then carried forward by Johnson and colleagues (Johnson et al.,

1999; Geman and Johnson, 2002; Johnson, 2003) to develop stochastic versions of Lexical Functional Grammars (LFG) (Bresnan, 1982, 2001). Johnson et al. (1999) noted that defining probability distributions over all parses is computationally very expensive because it requires integrating over all parses of all sentences. However, they pointed out that for parsing purposes, it is sufficient to have a conditional distribution of parses given their yields (i.e., the sentences). They estimated this conditional distribution using log-linear models, following the idea of Abney (1997).

Their work was then extended by Geman and Johnson (2002) to handle broad-coverage parsing. They built their method on earlier work by Maxwell III and Kaplan (1995), who had developed dynamic programming routines to parse LFGs by building "packed representations" to compactly represent a set of parses of any given sentence instead of simply enumerating all of them. They then used those packed representations to estimate conditional distributions of generating parses given a sentence with the help of graphical models.

Along similar veins is the work of Miyao and Tsujii (2002, 2005) who were working with Head-driven Phrase Structure Grammars (Pollard and Sag, 1987, 1994). They also used log-linear models to estimate conditional distributions of parses given a sentence. However, their work differs in the type of data structure they used to represent multiple parses. The backbone of their approach is something they called "feature forests." These feature forests are more general, and less compact, than the packed representations of Maxwell III and Kaplan (1995). They also presented an analogue of the inside-outside algorithm (which is used for probabilistic context-free grammars) to estimate parameters given such feature forest representations.

3. Lexicalized Well-Founded Grammars

Lexicalized Well-Founded Grammars (LWFGs) are a type of constraint-based grammars introduced by (Muresan, 2006; Muresan and Rambow, 2007; Muresan, 2010, 2011), which extend traditional context-free grammars by incorporating semantic information into the grammar. LWFGs are functionally a type of Definite Clause Grammars (Pereira and Warren, 1980), and underlying LWFGs are Well-Founded Grammars, which are defined to be context-free grammars where there is a partial ordering among the non-terminals. Such an ordering allows for an ordering of the strings derived from the grammar, which in turn allows an ordering on the grammar rules. Essentially, such an ordering facilitates bottom-up induction of these grammars, and as such does not affect their generative capabilities (see Muresan and Rambow, 2007, for more information on the grammar's learnability).

Lexicalized Well-Founded Grammars build upon Well-Founded Grammars by encoding semantic composition and semantic interpretation constraints at the grammar rule level. The critical addition in LWFGs are representations called *semantic molecules*, which are used to incorporate semantic information at the string level.

3.1 Semantic Molecules

A semantic molecule associated with a natural language string w is defined as $w' = \binom{h}{b}$, where:

1. h (*head*) encodes the syntactic, or compositional, information of the string.
2. b (*body*) encodes the semantic information of the string.

These semantic molecules are further classified as *elementary semantic molecules* and *derived semantic molecules*. The former are representations associated with words and other lexical items, whereas the latter are built by a combination of other semantic molecules. Figure 3.1 depict examples of elementary and derived semantic molecules.

The *head* of a semantic molecule is a one-level (i.e. non-recursive) feature structure, which is depicted as an Attribute-Value Matrix (AVM) in Figure 3.1. Every head

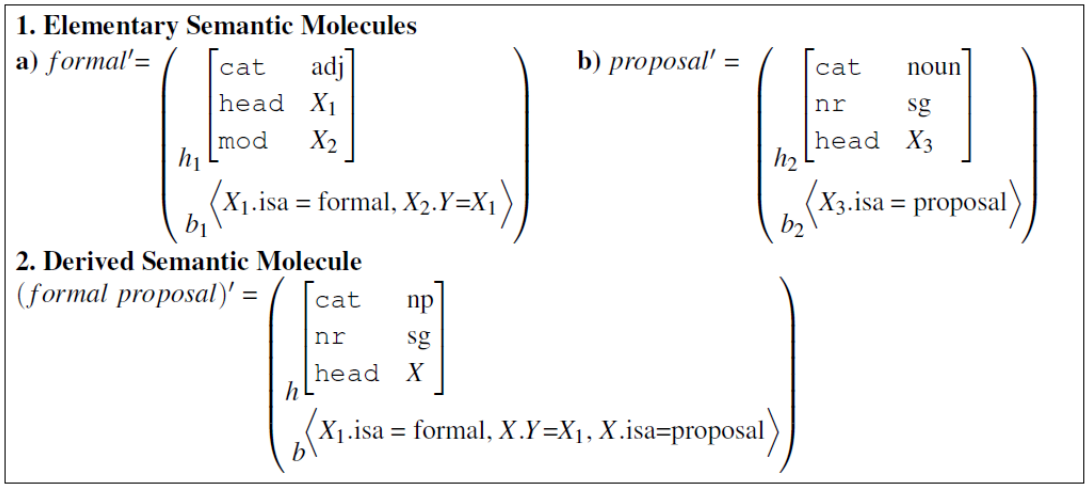


Figure 3.1: Example semantic molecules for *formal*, *proposal*, and *formal proposal*

contains at least two attributes: 1) *cat* – which encodes the syntactic category of the string; and 2) *head* – which encodes the semantic head of the string. In addition, there are features for grammatical features such as agreement, modifier-modifiee relations, etc. All these features are finite and defined a-priori for each syntactic category.

The *body* of a semantic molecule is a flat representation which Muresan (2006, 2010) have called OntoSeR (Ontology-based Semantic Representation). Essentially, OntoSeR is a logical representation built using a set of atomic predicates. The formal definition of an OntoSeR is as follows:

$$\langle OntoSeR\rangle := \langle AP\rangle | \langle OntoSeR\rangle\langle lop\rangle\langle OntoSeR\rangle$$

$$\langle AP\rangle := \langle conceptID\rangle.\langle attr\rangle = \langle concept\rangle$$

$$\langle AP\rangle := \langle conceptID\rangle\langle coord\rangle\langle conceptID\rangle$$

$$\langle concept\rangle := \langle conceptID\rangle | \langle conceptName\rangle$$

$$\langle conceptID\rangle := \langle logicalVariable\rangle$$

$$\langle conceptName\rangle := \langle lexicalWord\rangle$$

$$\langle attr\rangle := \langle attrID\rangle | \langle attrName\rangle$$

$$\langle attrID\rangle := \langle logicalVariable\rangle$$

$$\langle attrName\rangle := \langle lexicalWord\rangle$$

$$\langle coord\rangle := \langle lexicalCoord\rangle$$

$$\langle lop\rangle := \wedge$$

Here, $\langle lop\rangle$ is the logical conjunction operation $\wedge$. The $\langle coord\rangle$ operator is a linguistic coordinator, such as *and*, *or*, *but*, etc. $\langle conceptID\rangle$ and $\langle conceptName\rangle$ denote concept identifier and concept names in the semantic model. Similarly, $\langle attrID\rangle$ and $\langle attrName\rangle$ refer to attribute slots in the semantic model, where attribute slots are either properties or relations. The underlying OntoSeR representation for the grammar at hand may be changed to reflect the desired level of semantic representation that is required for the task. For example, we could encode only admissibility relations, such as thematic roles of verbs and prepositions, at the level of lexical entries, or we could have a more defined ontology with a hierarchy of concepts and roles, and relations among concepts. Thus, OntoSeR can be suitably modified to be sufficiently expressive of the desired aspects of natural language.

Figure 3.1 shows example OntoSeR representations. For example, the OntoSeR for *formal* is $\langle X_1.isa = formal, X_2.Y = X_1\rangle$. The representation encodes the meaning of *formal* as a concept ($X_1.isa = formal$), which is a value of a property of another concept X_2 in the semantic model ($X_2.Y = X_1$). In the meaning of *formal proposal*, we

see that X_2 is instantiated through composition to be the concept of the noun *proposal* ($X_2 = X_3 = X$). The variable Y will later be instantiated by the process of semantic interpretation which is based on a semantic model (e.g., Y=`manner`).

3.2 Semantic Composition and Interpretation

For Lexicalized Well-Founded Grammars, the lexicon consists of words paired with *elementary semantic molecules*. The lexicon does not specify the syntactic context in which the word is anchored, which makes the LWFG different from certain other lexicalized formal grammars, such as Combinatory Categorial Grammars (Steedman, 1996). The syntactic context for words is instead learned from examples, in the form of grammatical rules and compositional constraints.

Thus, the LWFG also has a set of constraint grammar rules, which can be recursive. The non-terminals in the grammar rules are augmented with *syntagmas* – tuples of strings and their semantic molecules $\big(w_i, \binom{h_i}{b_i}\big)$. For example, a grammar rule for a noun phrase could be:

$$NP\big(w, \binom{h}{b}\big) \rightarrow Adj\big(w_1, \binom{h_1}{b_1}\big), Noun\big(w_2, \binom{h_1}{b_2}\big) : \Phi_c(h, h_1, h_2), \Phi_i(b)$$

Grammar rules are also augmented with two types of constraints, one for *semantic composition* Φ_c, and one for *semantic interpretation* Φ_i. The composition constraints are applied to the *heads* of the semantic molecules to perform unification of the heads. The *bodies* are simply concatenated through logical conjunction along with variable substitution performed using Φ_c constraints. The variables to be substituted are discovered prior to the application of the unification operation. As shown in Figure 3.1, the body of the semantic molecule for *formal proposal* is a concatenation of the bodies of the adjective *formal* and the noun *proposal*. Furthermore, we can see that there has been a variable substitution with both X_2 and X_3 being substituted by X (or $\{X_2/X, X_3/X\}$), which we get after unifying the heads of the two semantic molecules and solving a system of equations – which are simplified versions of "path equations" (Shieber et al., 1983). For the grammar rule shown above, along with the semantic molecules shown in

Figure 3.1, the semantic composition constraints are given as: $\Phi_c(h, h_1, h_2) = \{h.cat = np, h.head = h_1.mod, h.head = h_2.head, h.nr = h_2.nr, h_1.cat = adj, h_2.cat = noun\}$. In the constraints, the part $\{h.head = h_1.mod, h.head = h_2.head\}$ indicates that the variable denoting the semantic head of the noun phrase *formal proposal* (X) should be the same as the variable denoting the semantic head of the noun *proposal* (X_2), and also the same as the variable denoting the `mod` attribute of the adjective *formal* (X_1). In turn, that information gives us the variable binding $\{X_2/X, X_3/X\}$. In Lexicalized Well-Founded Grammars, all of these constraints are learned together with the grammar rules.

Unlike semantic composition constraints, semantic interpretation constraints are not learned. Instead, they represent the validation based on some semantic model (the "world truth," in some sense). In the current implementation of the Lexicalized Well-Founded Grammars, the semantic interpretation constraints are predicates which can either succeed or fail. When the constraint succeeds, the variables of the semantic representation get instantiated with concepts/slots in the semantic model (see, Muresan, 2006, 2010). Continuing with our *formal proposal* example, the corresponding semantic interpretation constraint $\Phi_i(b)$ will succeed and return $\langle X_1.isa = formal, X.manner = X_1, X.isa = proposal\rangle$. On the other hand, if we had a noun phrase *fair-hair proposal*, the the constraint would have failed as the phrase is nonsensical. These interpretation constraints play an important role in the disambiguation of some linguistic phenomena, such as PP-attachment, coordination, etc. Furthermore, unlike some current broad-coverage grammars or statistical syntactic parsers, these constraints allow interpretation of phenomena such as noun-noun compounds and prepositions.

3.3 Formal Definition

We can now formally define the Lexicalized Well-Founded Grammars. Essentially, an LWFG is a 6-tuple, $G = \langle \Sigma, \Sigma', N_G, R_G, P_G, S\rangle$, where:

1. Σ is a finite set of terminal symbols (lexical items)

2. Σ' is a finite set of elementary semantic molecules corresponding to the set of

terminal symbols. In other words, for every $w \in \Sigma$ we have an $w' \in \Sigma'$ such that together w, w' form a syntagma (pair of string and semantic molecule).

3. N_G is a finite set of non-terminal symbols. That is, $N_G \cap \Sigma = \emptyset$.

4. R_G is a partial ordering relation, $\succeq$, among the non-terminals.

5. P_G is a set of constraint production rules. Each rule is a triplet $(A, (B_1, \ldots, B_n), \Phi)$, written as $A(\sigma) \rightarrow B_1(\sigma_1), \ldots, B_n(\sigma_n) : \Phi(\bar{\sigma})$. Here, $\bar{\sigma} = (\sigma, \sigma_1, \ldots, \sigma_n)$ such that $\sigma = (w, w')$, $\sigma_i = (w_i, w_i'), 1 \leq i \leq n, w = w_1 \ldots w_n, w' = w_1' \circ \cdots \circ w_n'$. The $\circ$ operator denotes the semantic composition operator which unifies the heads of the semantic molecules and performs variable substitution on concatenated bodies, as described above. All production rules have the following properties:

   - The rules can be classified into one of three categories: *ordered non-recursive rules*, *ordered recursive rules*, and *non-ordered rules*.
   - Every non-terminal symbol is a left-hand side of at least one ordered non-recursive rule.
   - No non-terminal symbol derives an empty string.
   - All non-terminals in production rules are augmented with generalized syntagmas.
   - All production rules are augmented with semantic composition and semantic interpretation constraints.
   - All production rules ensure grammar reversibility (this point aids grammatical learning, something which we choose not to discuss in this thesis for brevity.)

6. $S \in N_G$ is the start symbol and $\forall A \in N_G, S \succeq A$.

7. All substrings w that are derived from a non-terminal A have the same category value in the head of their semantic molecules, given by the name of the non-terminal. In other words $h.cat = A$, where $w' = \binom{h}{b}$ is the semantic molecule of w.

Item form	$[i, j, \sigma_{ij}, A \rightarrow \alpha \bullet \beta\Phi_A]$	$1 \leq i, j \leq n+1, A \in N_G, \alpha\beta \in N_G^*$, and Φ_A can be true
Axioms	$[i, i+1, \sigma^L_{ii+1}, B_i \rightarrow \bullet]$	$1 \leq i \leq n, B_i \in N_G, B_i \rightarrow \sigma_i \in P_G, \sigma_i = (w_i, w_i') \in \Sigma \times \Sigma'$
Goals	$[i, j, \sigma^L_{ij}, A \rightarrow \alpha\Phi_A\bullet]$	$1 \leq i, j \leq n+1, A \in N_G$
Inference Rules		
Prediction	$\frac{[i,j,\sigma^L_{ij}, B \rightarrow \beta\Phi_B \bullet]}{[i,i,\sigma^R_{ii}, A \rightarrow \bullet B\gamma\Phi_A]} \quad \langle A \rightarrow B\gamma : \Phi_A \rangle$	
Completion	$\frac{[i,j,\sigma^R_{ij}, A \rightarrow \alpha \bullet B\gamma\Phi_A][j,k,\sigma^L_{jk}, B \rightarrow \beta\Phi_B \bullet]}{[i,k,\sigma^R_{ik}, A \rightarrow \alpha B \bullet \gamma\Phi_A]}$	
Constraint	$\frac{[i,j,\sigma^R_{ij}, A \rightarrow \alpha \bullet \Phi_A]}{[i,j,\sigma^L_{ij}, A \rightarrow \alpha\Phi_A \bullet]} \quad \langle \vdash SLD\Phi_A \rangle$	Φ_A is satisfiable

Table 3.1: Muresan's (2006) deductive parser for LWFG

3.4 Parsing Lexicalized Well-Founded Grammars

In this section, we now describe a parser for the Lexicalized Well-Founded Grammars, as described by Muresan (2006). The parser can be viewed as a deductive process that attempts to prove claims about the grammaticality of a string from assumptions about the grammatical status of the string's constituents, as well as the linear order between them (Pereira and Warren, 1983). The view of parsing as a deductive process was put forth by Shieber et al. (1995) in which rules of inference are used to derive statements about the grammatical status of strings. These rules of inference act by deriving statements using other statements which are all represented by formulas in a suitable formal language. The grammatical deductive system consists of a set of inference rules and a set of axioms and goals which are given by an appropriate set of formula schemata, $\langle deductive\ system \rangle := \langle \langle formula\ schemata \rangle, \langle axioms \rangle, \langle inference\ rule \rangle, \langle goals \rangle \rangle$, where $\langle axioms \rangle \cup \langle goals \rangle \subseteq \langle formula\ schemata \rangle$.

The general form of a rule of inference is:

$$\frac{A_1 \ldots A_k}{B} \quad \langle side\ conditions \rangle$$

The antecedents $A_1 \ldots A_k$ and the consequent B are formula schemata, which means

they contain metavariables that get instantiated when the inference rule is used. A derivation of a formula B from assumptions $A_1, \ldots, A_k$ is a sequence of formulas the end product of which is B, i.e. $S_1, \ldots, S_n$ such that $S_n = B$. Each interim formula, S_i, is either an axiom, or there is some other rule of inference R and formulas $S_{i_1}, \ldots, S_{i_k}$ with $i_1, \ldots, i_k < i$ such that for appropriate substitutions of terms for metavariables in R, the formulas $S_{i_1}, \ldots, S_{i_k}$ match the antecedents of the rule R, S_i matches the consequent, and the side conditions are satisfied.

In the parsing framework, the side conditions refer to rules of a particular grammar, and the formulas refer to string positions in the fixed string to be parsed $w = w_1 \ldots w_n$. The goal formula states whether a string is grammatical according to the given formula. The parsing process corresponds to finding a derivation path from the start symbol to a goal formula.

The LWFG parser described by Muresan (2006) uses a robust bottom-up active chart parsing algorithm (Kay, 1973), and is presented as a deductive system (see Table 3.1) To understand the parser, consider the following LWFG $G = \langle \Sigma, \Sigma', N_G, R_G, P_G, S \rangle$. The parser starts with a string w and produces a syntagma $\sigma = \left(w, \binom{h}{b}\right)$.

The parsing formula schemata are of the form $[i, j, \sigma_{ij}, A \rightarrow \alpha \bullet \beta \Phi_A]$, where $A \rightarrow \alpha\beta : \Phi_A$ is a production rule in the grammar. The $\bullet$ shows how much of the right-hand side has been recognized so far, i points to the parent node where the rule was invoked, and j points to the position in the input that the recognition has reached. The lexical items containing elementary semantic molecules are assumed to make a true claim, and thus items of the form $[i, i+1, \sigma^L_{ii+1}, B_i \rightarrow \bullet]$ are assumed to be axiomatic. The goal items are of the form $[i, j, \sigma^L_{ij}, A \rightarrow \alpha \Phi_A \bullet]$, where σ^L_{ij} is derived from the rule $A \rightarrow \alpha : \Phi_A$. The superscript L indicates that the syntagma is ground-derived and belongs to the left-hand side non-terminal.

In chart parsing terminology, each item is taken to be an "edge." The axioms and goals are inactive edges having $\bullet$ at the end, whereas the rest are active edges. There are three inference rules used to obtain the goal items:

3.4.1 Prediction Rule

This is the bottom-up prediction rule which adds empty active edges. The rule predicts an active edge (i.e. an item) $[i, i, \sigma^R_{ii}, A \rightarrow \bullet B\gamma\Phi_A]$ from an inactive edge $[i, j, \sigma^L_{ij}, B \rightarrow \beta\Phi_B\bullet]$ and a grammar rule as a side condition $A \rightarrow B\gamma : \Phi_A$. The predicted syntagma $\sigma^R_{ii} = \left(w^R_{ii}, \binom{h^R_{ii}}{b^R_{ii}}\right)$ is empty, where $w^R_{ii} = \epsilon$ is an empty string, $b^R_{ii} = true$ is the empty semantic representation, and $h^R_{ii} = \emptyset$ is the empty head. The superscript R on the predicted syntagma indicates a partially-parsed syntagma that belongs to the right-hand side of the rule.

3.4.2 Completion Rule

This rule corresponds to shifting the $\bullet$ across a non-terminal in the right-hand side of the production rule. The rule essentially combines an active and an inactive edge to obtain a new active edge. Thus, it has two antecedents $[i, j, \sigma^R_{ij}, A \rightarrow \alpha \bullet B\gamma\Phi_A]$ and $[j, k, \sigma^L_{jk}, B \rightarrow \beta\Phi_B\bullet]$, and one consequent $[i, k, \sigma^R_{ik}, A \rightarrow \alpha B\bullet\gamma\Phi_A]$. The rule requires no side conditions. In terms of the syntagmas, shifting the $\bullet$ implies string concatenation, conjunction of the semantic molecule bodies, and a union of the semantic molecule heads. That is, $\sigma^R_{ik} = \sigma^R_{ij} \circ \sigma^L_{jk}$, where $w^R_{ik} = w^R_{ij} w^L_{jk}$ (string concatenation), $b^R_{ik} = b^R_{ij} b^L_{jk}$ (body conjunction), and $h^R_{ik} = h^R_{ij} \cup h^L_{jk}$ (head union).

3.4.3 Constraint Rule

This rule is used to obtain inactive edges from active edges by executing the grammar constraint Φ_A, thus shifting the $\bullet$ across the constraint. The execution of the constraint is in form of an SLD resolution $\vdash \Phi_A$ which acts as the side condition. The constraint rule also produces the ground derived syntagmas σ^L_{ij}, thus the goal items can only be produced on successful application of the constraint rule.

4. Revised LWFG Parser

In this chapter, we describe our implementation of the LWFG Parser. Unlike Muresan (2006), who implemented their parser in Prolog, we chose to use Python as the programming language of implementation. We start by first making the case for our choice.

4.1 The Case for Python

4.1.1 NLTK

Python has an extensive open source library dedicated to natural language processing called Natural Language Toolkit (NLTK) (Bird et al., 2009). Developing a parser for LWFG in Python allows for an easy integration with the library, and thus eases open-source development of the project. One limitation of Python over Prolog, at least for this project, is the lack of an inbuilt method for SLD resolution. As mentioned earlier, the constraint rule in the parser performs SLD resolution to verify that the semantic composition and semantic integration constraints have been satisfied. Prolog, being a declarative language for logic programming, makes performing an SLD resolution simple. Python, on the other hand, does not provide a straight-forward means of doing that. However, being an widely-used open-source project, there are other libraries that provide easy-to-use methods for performing SLD resolution. As discussed later, we used NLTK's unification methods to implement constraint satisfaction.

4.1.2 Scalability and long-term investment

While Prolog is perfectly acceptable for prototype development, Python offers a much better alternative in terms of scalability. A Python-based implementation is especially desirable in light of a future extension to stochastic parsing. Prolog, being a logic programming language, is not naturally suited for numerical operations, and even simple arithmetic operations are hard to perform in Prolog. Given this, using Prolog to

implement a stochastic LWFG parser, which would require substantial numerical computations, seems unreasonable. On the other hand, Python offers no such limitations, and a Python-based implementation of the LWFG parser should be easily extendable to a stochastic version.

4.2 Data Structures

Taking NLTK's grammar module as inspiration, we constructed a separate classes for non-terminals and production rules. We represented a Lexicalized Well-Founded Grammar as a class that stored its start symbol (a non-terminal) and a list of productions. Every non-terminal was an object of a class that stored the symbol of the non-terminal, and its corresponding syntagma, which was represented as three separate variables: the string, the body, and the head. The body is represented as an object of a class which implements the OntoSeR representation of Muresan (2006). The head is represented using NLTK's feature structure class called FeatStruct. As mentioned earlier, the feature structure is a one-level object and contains no recursive structures.

Production rule are represented by storing the left-hand side non-terminal, a list of right-hand side non-terminals, and a set of compositional constraints. These compositional constraints are also feature structures, but with one noticeable change. The feature structure object contains one feature for every non-terminal in the rule. The value of all such features is another feature structure that corresponds to the head of that non-terminal. Every feature within the sub-structures contain values which are initialized to variables to indicate linkage.

Recall the example used earlier of the following grammar rule:

$$NP\big(w, \binom{h}{b}\big) \rightarrow Adj\big(w_1, \binom{h_1}{b_1}\big), Noun\big(w_2, \binom{h_1}{b_2}\big) : \Phi_c(h, h_1, h_2), \Phi_i(b)$$

The semantic compositional constraints for the example grammar were given as:

$$\Phi_c(h, h_1, h_2) = \{h.cat = np, h.head = h_1.mod, h.head = h_2.head,$$
$$h.nr = h_2.nr, h_1.cat = adj, h_2.cat = noun\}$$

In our feature structure representation, the constraints are represented in the following way:

$$\left[\begin{array}{l} h = \left[\begin{array}{c} cat = np \\ head = ?x \\ nr = ?y \end{array}\right] \quad h_1 = \left[\begin{array}{c} cat = adj \\ mod = ?x \end{array}\right] \quad h_2 = \left[\begin{array}{c} cat = noun \\ head = ?x \\ nr = ?y \end{array}\right] \end{array}\right]$$

Here, $?x$ and $?y$ are variables, and act as links between $h.head$, $h_1.mod$, and $h_2.head$, and $h.nr$ and $h_2.nr$ respectively.

We also created a representation to store a complete grammar. In the representation, we distinguish the "lexicon" from the grammar by defining the lexicon to be consisting of pre-terminal rules. That is, rules that expand to a terminal are labeled as the lexicon of the grammar. These rules are distinguished from other rules because they contain an instantiation of the semantic body (the OntoSeR representation) corresponding to the terminal symbol along with an instantiation of the semantic head. Other rules, which expanded to non-terminals, are taken to be abstract rules, and only contain compositional constraints. Examples of the representations for the lexicon and the other production rules are shown in the Appendix.

Traditionally, the chart in a chart parser is implemented as a 2-d array where a cell at index (i, j) contains states (or "edges") whose span starts at i and ends at j. However, we implemented the chart as a 1-d array of size $n + 1$, where n is the size of the input sentence. The 0^{th} cell corresponds to the start of the sentence, and every other i^{th} cell corresponds to the i^{th} word in the sentence. Each cell i contains states whose span ends at i. A state may be active, i.e. not yet finished, or inactive, i.e. fully recognized. We implemented the state as a class with a production rule, an index of the $\bullet$, and chart indices representing the recognized span of the production rule (the sequence of words from the sentence that the production rule derives; the span is represented using two indices i and j described in Section 2.4). In addition, states also

Item form	$[i, j, \sigma_{ij}, A \rightarrow \alpha \bullet \beta(\Phi_A)]$	$1 \leq i, j \leq n+1, A \in N_G, \alpha\beta \in N_G^*$, and Φ_A can be true
Axioms	$[i, i+1, \sigma^L_{ii+1}, B_i \rightarrow \bullet]$	$1 \leq i \leq n, B_i \in N_G, B_i \rightarrow \sigma_i \in P_G, \sigma_i = (w_i, w_i') \in \Sigma \times \Sigma'$
Goals	$[i, j, \sigma^L_{ij}, A \rightarrow \alpha \bullet (\Phi_A)]$	$1 \leq i, j \leq n+1, A \in N_G$
Inference Rules		
Prediction	$\frac{[i,j,\sigma^L_{ij}, B \rightarrow \beta \bullet (\Phi_B)]}{[i,i,\sigma^R_{ii}, A \rightarrow \bullet B\gamma(\Phi_A)]}$ $\langle A \rightarrow B\gamma : \Phi_A \rangle$	
Completion	$\frac{[i,j,\sigma^R_{ij}, A \rightarrow \alpha \bullet B\gamma(\Phi_A)][j,k,\sigma^L_{jk}, B \rightarrow \beta \bullet (\Phi_B)]}{[i,k,(\sigma^R_{ik})', A \rightarrow \alpha B \bullet \gamma(\Phi'_A)]}$ $\langle \sigma^L_{jk}$ satisfies $\Phi_A \rangle$	$\Phi'_A :=$ Φ_A unified with Φ_B

Table 4.1: A revised LWFG parser where constraint and completion rules are merged together.

store the compositional constraints, again as feature structures.

4.3 Revised Parser

The parser described by Muresan (2006, 2010) performs SLD resolution only at the end of the completion of an item. That is, the antecedent of the constraint rule is always of the form: $[i, j, \sigma^R_{ij}, A \rightarrow \alpha \bullet \Phi_A]$. However, in our implementation, we chose to merge the constraint and the completion step together. Thus, every time the $\bullet$ is moved across a non-terminal, the composition constraints get applied, thus ensuring an early detection of an incorrect parse.

We justify our modification thusly. Consider the following rule, $VP \rightarrow VNP$. The rule corresponds to a verb phrase that consists of a transitive verb and an object noun phrase. For example, the rule should be applicable on the phrase, *(John...) liked apples*, however the rule should also not be applicable on phrases containing intransitive verbs, such as *(John...) slept apples*. Late application of the constraint rule means such invalid phrases are detected only when they are fully completed. However, our merging of the completion and the constraint rules makes the detection earlier. In effect, that greatly reduces the number of active edges that the parser entertains during its running, which proves invaluable when working with large grammars.

Table 4.1 shows the modified parser that we implemented. Because constraint

satisfaction is now performed every time an item is completed, we have modified the notation of the items to depict the passive nature of the compositional constraints. In simple terms, the • now only needs to cross the last non-terminal on the right-hand side of a rule for the item to be completed. Another noticeable change is the modification of the compositional constraints at the end of completion. Now, every completion modifies the set of compositional constraints on the active edge to denote unification with the syntagma of the active edge. In other words, every time the • moves across a non-terminal, the semantic information contained in the syntagma corresponding to that non-terminal in the inactive edge is merged with the constraints of the active edge. The resulting variable binding is also reflected in the body of syntagma of the active edge.

As an example of the process, let us again consider the following rule:

$$NP \rightarrow \bullet\ Adj\ Noun$$

The initial set of compositional constraints are given as:

$$\left[\begin{array}{l} h = \left[\begin{array}{l} cat = np \\ head = ?x \\ nr = ?y \end{array}\right] \quad h_1 = \left[\begin{array}{l} cat = adj \\ mod = ?x \end{array}\right] \quad h_2 = \left[\begin{array}{l} cat = noun \\ head = ?x \\ nr = ?y \end{array}\right] \end{array}\right]$$

Now, let us consider the rule:

$$Adj \rightarrow \texttt{formal}\ \bullet$$

Its syntagma is given as:

$$w = \texttt{formal}, \quad b = ?x.isa = \texttt{formal}, ?y.hasProp = ?x, \quad h = \begin{bmatrix} cat = adj \\ head = ?x \\ mod = ?y \end{bmatrix}$$

When the second rule is used to complete the first, its head is unified with the feature structure corresponding to the non-terminal Adj in compositional constraints of the first rule. That is, we unify h of the second rule with h_1 of the first rule. That gives us a modified set of compositional constraints:

$$\begin{bmatrix} h = \begin{bmatrix} cat = np \\ head = ?x \\ nr = ?y \end{bmatrix} & h_1 = \begin{bmatrix} cat = adj \\ head = ?z \\ mod = ?x \end{bmatrix} & h_2 = \begin{bmatrix} cat = noun \\ head = ?x \\ nr = ?y \end{bmatrix} \end{bmatrix}$$

The process also modifies the string of the syntagma of the left-hand side of the active edge (NP in our example), to concatenate the original string with that of the inactive edge. In our example, the original empty string is concatenated with w to get `formal`. Furthermore, the body of the left-hand side of the active edge is logically concatenated with the body of the inactive edge and appropriate variable bindings are substituted. In our example, the body of NP is modified to be: $?z.isa = \texttt{formal}, ?x.hasProp = ?z$

4.4 Implementation

The parser is detailed below:

- **Initialization**: The chart of the parser is initialized to $n+1$ lists where each list corresponds to a word in the given sentence $w = w_1 \dots w_n$ (except the first, which corresponds to the start of the sentence). States corresponding to items of the form $[i, i+1, \sigma^L_{ii+1}, w_i \rightarrow \bullet]$ are inserted to the respective n lists.

  INITIALIZE-CHART($chart, words$)

  1 **for** $i \leftarrow 1$ TO LENGTH($words$) $+ 1$

  2 **do** $chart$.APPEND([])

  3 **for** $i \leftarrow 1$ TO LENGTH($words$) $+ 1$

  4 **do** ADDTOCHART($[i, i+1, \sigma^L_{ii+1}, words[i] \rightarrow \bullet], chart[i]$)

- **Prediction**: Given a completed state, the predictor method generates a new state and adds it to the chart in the ith list where i is the same as the starting index of the completed state.

  PREDICTOR($[i, j, \sigma^L_{ij}, B \rightarrow \beta \bullet (\Phi_B)]$)

  1 **for** each $(A \rightarrow B\gamma : \Phi_A) \in Grammar$

  2 **do** ADDTOCHART($[i, i, \sigma^R_{ii}, A \rightarrow \bullet B\gamma(\Phi_A)], chart[i]$)

- **Completion**: Given a completed state, the completer method looks at all states in the jth list in the chart where j is the start index of the completed state. Of all the states in that list, any state which contains the left-hand side of the completed state in the position just to the right of the $\bullet$ is selected. The new state is generated from the selected state such that the start index is the same as that of the selected state, the end index is the same as that of the completed state, and the $\bullet$ is moved one step to the right. The semantic information is copied over the selected state to the new state. Subsequently, the constraint rule gets applied to unify the semantic information of the left-hand side of the completed state with the compositional constraints of the selected state. If the constraint rule fails, the new state is rejected, otherwise it is inserted into the chart in the kth list where k is the end index of the completed state.

COMPLETER($compState = [j, k, \sigma^L_{jk}, B \rightarrow \beta \bullet (\Phi_B)]$)

1 **for** each $[i, j, \sigma^R_{ij}, A \rightarrow \alpha \bullet B\gamma(\Phi_A)] \in chart[j]$

2 **do** $newState = [i, k, \sigma^R_{ik}, A \rightarrow \alpha B \bullet \gamma(\Phi_A)]$

3 **if** APPLYCONSTRAINT($newState, compState$) == TRUE

4 **do** ADDTOCHART($[i, k, \sigma^R_{ik}, A \rightarrow \alpha B \bullet \gamma(\Phi_A)], chart[k]$)

- **Constraint Application**: The ApplyConstraint method first unifies the head of the completed state's left-hand side with the compositional constraints of the new state. The unification is done so that the head from the completed state unifies only with the feature structure corresponding to the same non-terminal in the new state (i.e. the non-terminal across which the $\bullet$ moves). We used the unification function implemented for FeatStruct in Python's NLTK. An unsuccessful unification implies the completed state does not satisfy the compositional constraints of the new state, and thus the method returns FALSE. On the other hand, if the unification is successful, a set of variable bindings is obtained. We generate a new syntagma by storing the feature structure corresponding to the left-hand side of the new state as its head. The body of the new syntagma is obtained by performing a logical conjunction between the previous body and the body from the completed state, along with an appropriate variable substitution. The string of the new syntagma is obtained by simply concatenating the previous string of the new state with that from the completed state.

ApplyConstraint($newState, compState$)

1 $bindings \leftarrow \{\}$
2 $\Phi'_A \leftarrow$ Unify($newState.\Phi_A, compState.\sigma^L_{jk}.head, bindings$)
3 **if** $\Phi'_A =$ Null
4 **do return** False
5 $\sigma \leftarrow newState.\sigma^R_{ik}$
6 $\sigma.head \leftarrow \Phi_A[h]$
7 $\sigma.body \leftarrow$ MergeBodies($\sigma.body, compState.\sigma^L_{jk}.body, bindings$)
8 $\sigma.string \leftarrow$ StringConcat($\sigma.string, compState.\sigma^L_{jk}.string$)
9 $newState.\sigma^R_{ik} \leftarrow \sigma$
10 **return** True

- **LWFG Parser**: The parser begins by initializing the chart. Subsequently, the parser goes through the chart looks for completed states. If a state is completed, the Completer and the Predictor methods get called. The whole process is repeated until no new states are added to the chart.

LWFG-Parse($G, words$)

1 $chart \leftarrow []$
2 Initialize-Chart($chart, words$)
3 **while** no new state is added to the *chart*
4 **do for** each $state \in chart$
5 **do if** Is-Complete($state$)
6 **do** Completer($state$)
7 Predictor($state$)
8 **return** *chart*

4.5 Complexity Analysis

The underlying algorithm of the parser is a bottom-up chart parser, with the added property of performing a feature unification at every completion step. Before analyzing

the algorithm for LWFGs, consider a bottom-up chart parser for a context-free grammar. Each item in the chart is of the form $[i, j, A \rightarrow \alpha \bullet \beta], 1 \leq i < j$. For an input sentence containing n words, there are $n + 1$ chart entries. In each chart entry, the worst case maximum number of distinct states is equal to the maximum number of dotted rules times the maximum number of distinct span values, or $O(n|G|)$, where $|G|$ is the size of the grammar. The time to process a single item is given by the time to perform the predict and the completion operations on it. In the worst case, the predict operation creates $O(|G|)$ states. The completion operation advances the $\bullet$ in all states in some state set, and thus can take at most $O(n|G|)$ steps. Thus, in the worst case, the overall time per state is given by $O(n|G| + |G|) = O(n|G|)$. Factoring in all costs, we can see that the overall time complexity of the algorithm is bounded in the worst case by $O(n^3|G|^2)$.

The fact that our parser is also a bottom-up chart parser might lead us to believe that it has the same worst-case time complexity. However, because our parser represents semantic information in every state, it suffers from an exponential worst-case bound. To see why, let us first consider the following context-free grammar:

```
1. n -> nc
2. nc -> na nc
3. na -> na na
4. na -> noun
5. nc -> noun
```

These rules correspond to noun-compound cases where some nouns may act as adjectival modifiers on other nouns. Such cases correspond to examples like *government road accident research center* and *England national football team manager*. Now, consider the phrase, *football team manager*. The phrase is ambiguous given the example grammar and has two separate derivations which are shown in Figure 4.1. Thus, one interpretation has *football* be a modifier on the noun phrase *team manager*, and the other has *football team* be a compound modifier on the noun *manager*.

Another way of visualizing the ambiguity is by using parentheses. For example,

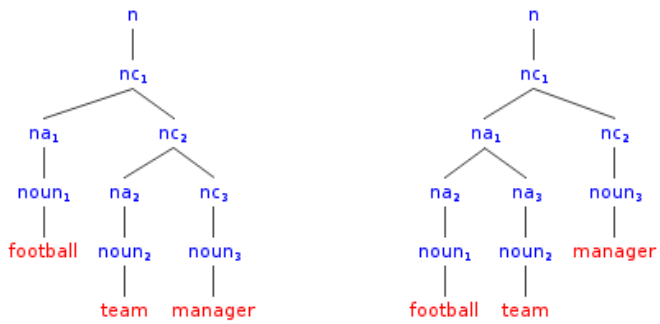


Figure 4.1: Ambiguous parse-trees corresponding to the noun compound, *football team manager*

one interpretation is *(football (team manager))*, and the other, *((football team) manager)*. Thus, in general, a noun-compound will have as many interpretations as there are ways of parenthesizing it. It is well known that number of ways of parenthesizing n factors is equal to the nth Catalan number C_n, which is given by:

$$C_n = \frac{(2n)!}{n!(n+1)!}$$

Using Stirling's approximation for $n!$, we can also show that $C_n = O(n^{-3/2}4^n)$. Thus, the number of derivations corresponding to ambiguous noun-compounds grow exponentially.

Now, one might ask why parsers for context-free grammars do not suffer from worst-case exponential performance. The reason is because enumeration of derivation trees is usually not considered a part of the parsing process. Instead, it is perfectly acceptable for a parser to only store pointers that may later be used to enumerate all possible derivation trees. For our example phrase, *football team manager*, the bottom-up chart parser will only have one state corresponding to the highest level `nc` node ($\texttt{nc}_1$ in Figure 4.1), namely $[0, 3, \texttt{nc} \rightarrow \texttt{na nc} \bullet]$. Depending on the implementation, the parser may also store pointers to two different `na` and `nc` pairs that combine to get this state. Thus, the use of pointers to eliminate redundancy allows context-free parsers to perform in tractable time.

Importantly, parsers for context-free grammars are able to eliminate redundancy

because context-free grammars do not encode any semantic information at the state-level. In our example, at the highest-level `nc` node, we do not know whether *football* is a modifier on the noun-compound *team player* or *football team* collectively modifies *player*. On the other hand, any semantic parser that encodes semantic information must differentiate between the two interpretations, and thus has to represent the two states separately. This crucial difference means that the chart size of parsers for such grammars has a worst-case exponential bound. In turn, this affects the overall time complexity of such parsers as well, making them perform at worst-case exponential time. Because our parser also represents semantic information at the state level (in the form of Φ_A), it suffers from the same theoretical upper-bound.

Crucially, the exponential bound applies to the LWFG chart parser only because all possibilities are being entertained. Currently, our parser does not include *semantic interpretation* constraints, which act as filters that select only those possibilities that are plausible (see Section 3.2). If we go back to the example phrase, *football team manager*, only one of the two possibilities is plausible: *((football team) manager)* (depicted as the right parse tree in Figure 4.1) Applying the interpretation constraints at the rule-level (i.e. after every completion step) would allow the parser to weed out implausible parses. Thus, in practice, a complete LWFG parser with semantic interpretation constraints is expected to be tractable.

5. Evaluation

In this chapter, we discuss empirical evaluations to test the correctness of our parser. To that end, we first constructed a mid-sized grammar, consisting of 477 production rules, of which 351 were pre-terminal rules (see Appendix for a sample from the grammar). It is important to keep in mind that the number of pre-terminal rules does not equal the number of words in the lexicon. As seen in the sample grammar in the Appendix, there are often pre-terminal rules that expand to the same word, differing in the semantic representation for that word.

We created a random sentence generator that randomly expanded non-terminals, starting from the start state, till a complete sentence was formed. At each non-terminal, all possible productions were equally likely to be expanded (in other words, there was a uniform distribution over them), but care had to be taken to ensure that only sentences that the grammar would allow are constructed by ensuring that compositional constraints are satisfied at every expansion. The generator also recorded the syntactic derivation used to generate a sentence, which we used that derivation as the "gold standard" to evaluate our parser against.

The random sentence generator was then used to create a list of 7500 sentences, with the constraint that the sentences were all of length at most 10. Later evaluations confirmed that the parser can in fact parse sentences of greater length. Figure 5.1 shows a histogram of the number of sentences as a function of the number of words. As can be seen, the random sentence generator tended to prefer producing sentences of shorter length. We believe this is a property of the grammar itself, and we did not investigate this further. Furthermore, 104 of the 477 productions were never used to generate a sentence, out of which 72 were pre-terminal productions. Thus, the effective size of the grammar was 373, with the number of pre-terminal productions equaling 279.

Our parser was able to parse all of the 7500 sentences. In addition, for each sentence, the parser was able to derive the parse that matched its gold standard. We recorded statistics on time taken to parse, number of derivations, and the size of the chart for each sentence.

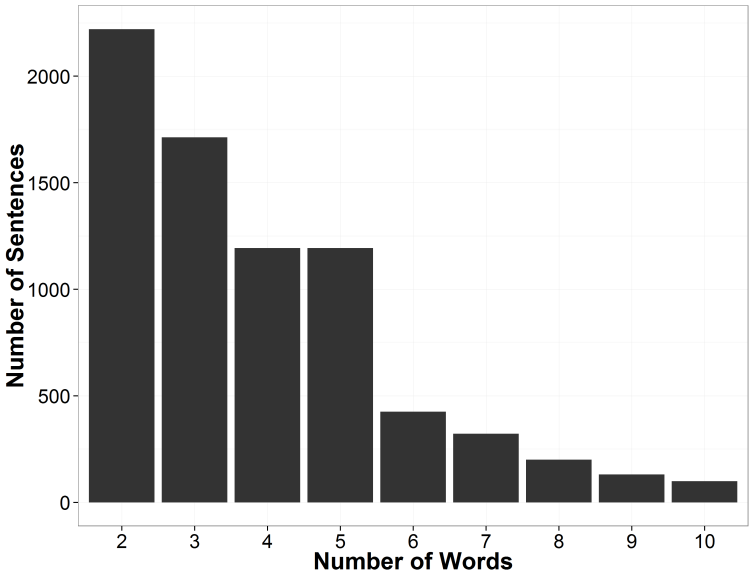


Figure 5.1: Histogram of number of sentences from the random sample used for evaluation as a function of sentence length

While our parser was able to effectively parse all sentences, there were certain cases that took a considerably large amount of time to parse. One example sentence that was troublesome for the parser was: *particle boy afternoon boy door treatment man pencil particle went*. Of course, the sentence does not make much sense, but that is simply because interpretation rules were not applied to generate these random sentences. The parse tree depicting the derivation that produced this sentence is shown in Figure 5.2. As the example shows, this sentence falls into the class of sentences that cause an exponential blowup in the chart size and parsing time for semantic parsers (see Section 4.5). Because there are 9 nouns preceding a verb, and because all of the initial 8 nouns act as adjectival modifiers, the total number of ways of interpreting this sentence is given by the 9th Catalan number, which is 1430. Indeed, our parser was able to identify all 1430 derivations, but took just over 6 hours to find them all and finished with a chart size of 50103 states.

Again, this is not a problem specific to our parser or grammar, but is a general issue that all semantic parsers face. Regardless, such cases severely skew the statistical measures of the parameters under investigation. Thus, for our analyses, we discarded sentences on which the parser took more than 1000 seconds. In total, we had to discard

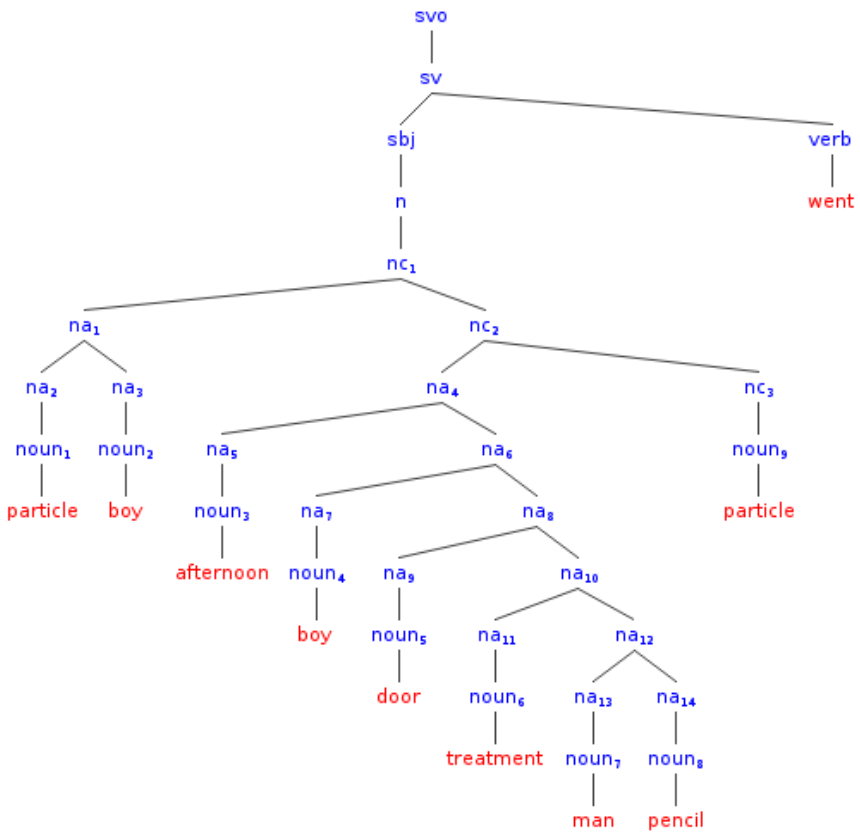


Figure 5.2: Derivation tree for the sentence *particle boy afternoon boy door treatment man pencil particle went*

13 sentences (.2% of our data). The mean values for each recorded statistic as a function of the number of words are reported in Table 5.1. In addition, Figure 5.3 shows graphical representations of the means and the standard errors for the measures.

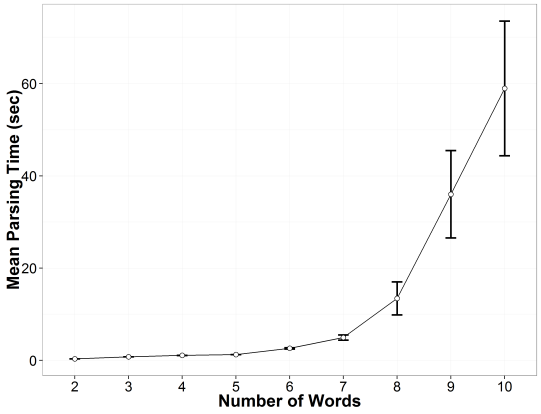
To facilitate a meaningful comparison with a baseline, we implemented a bottom-up chart parser for context-free grammars. Recall here that our parser is also essentially a bottom-up active chart parser, differing only in the type of grammar used and the incorporation of constraint-satisfaction in the derivation process. Thus, a comparison with an equivalent parser using a context-free grammar is justified. We constructed a context-free grammar by simply removing the compositional constraints from each rule, along with the semantic bodies for pre-terminal rules. The baseline parser was compared on the same set of 7500 sentences, and like earlier, we collected parsing times,

#Words	Mean Time (secs)	Mean #Derivations	Mean Chart Size
2	0.3420549	1.018910	60.3611
3	0.7653398	1.993579	104.6346
4	1.0821767	1.888610	136.2320
5	1.2620939	2.155909	158.9438
6	2.6118709	3.075117	253.1502
7	4.9626584	5.742236	366.2112
8	13.4548850	13.905000	568.1500
9	36.0161328	39.632812	1064.3359
10	58.9704111	85.544444	1564.8889

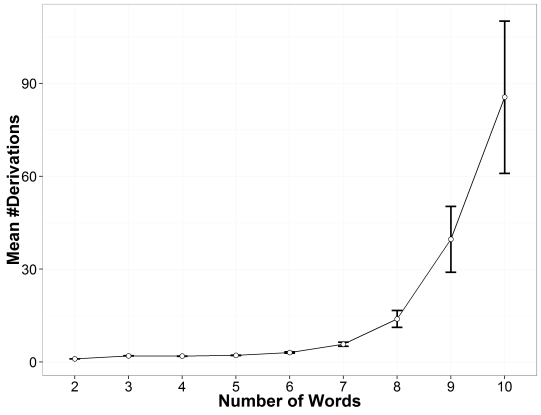
Table 5.1: Descriptive statistics of LWFG parser's performance on a random sample of sentences

#Words	Mean Time (secs)	Mean #Derivations	Mean Chart Size
2	0.01220441	1.847366	78.57587
3	0.02733859	2.583771	125.35552
4	0.06110721	4.675042	177.11307
5	0.08834870	4.321039	210.87091
6	0.21986621	9.298122	317.02113
7	0.36189752	14.565217	399.02484
8	0.55855001	21.870000	479.19500
9	0.98085937	64.890625	606.86719
10	1.39517779	93.444444	702.67778

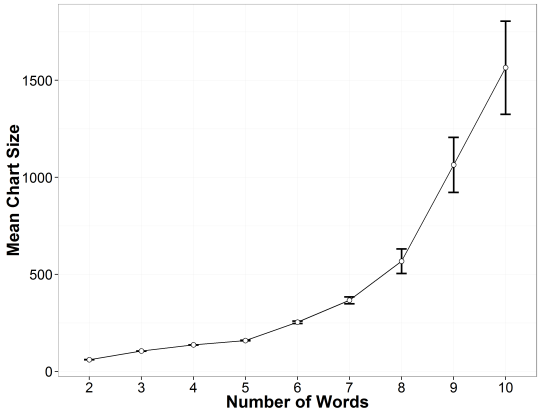
Table 5.2: Descriptive statistics of baseline parser's performance on a random sample of sentences



(a) Parsing time

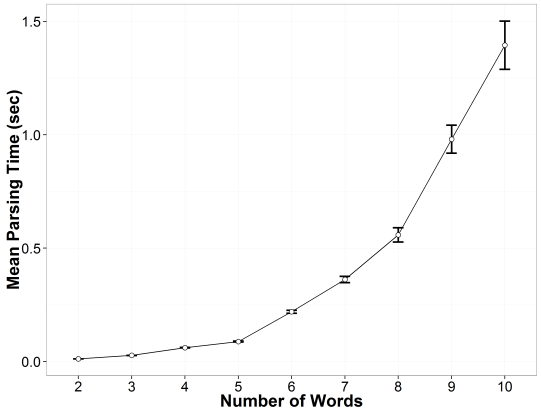


(b) Number of derivations

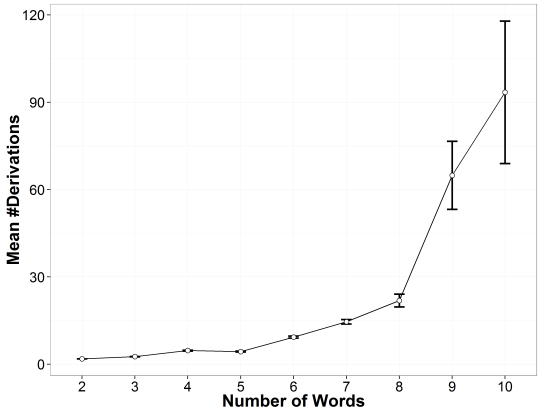


(c) Chart Size

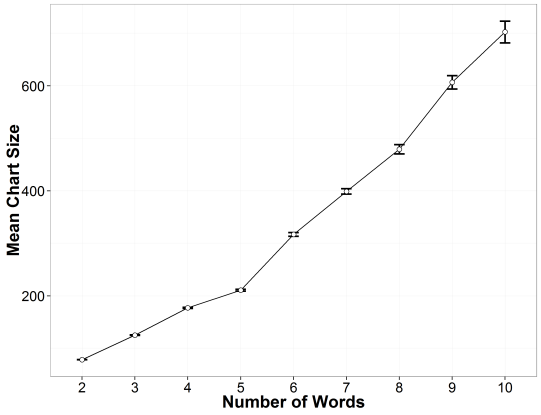
Figure 5.3: Graphical representation of the descriptive statistics for the LWFG parser



(a) Parsing time



(b) Number of derivations



(c) Chart Size

Figure 5.4: Graphical representation of the descriptive statistics for the baseline parser

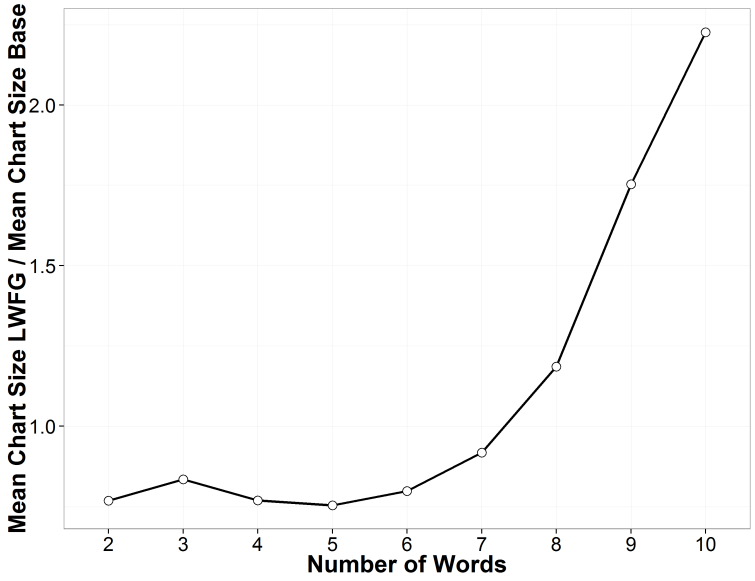


Figure 5.5: Ratio of mean chart size per sentence length from our LWFG parser and the baseline CFG parser.

number of derivations, and chart sizes per sentence. The 13 sentences that we had discarded in the evaluation of our LWFG parser were again discarded for the evaluation of the baseline. The performance of that parser is reported in Table 5.2, and shown graphically in Figure 5.4.

As we can clearly see, the use of a context-free grammar results in a greater number of derivations. This difference is expected because our parser rejects syntactic derivations that do not conform to semantic composition constraints. Additionally, the baseline parser was considerably faster than our parser. This disparity is also expected because the baseline parser does not need to store multiple copies of similar states that differ based on the underlying semantic information. Furthermore, the baseline parser does not need to perform constraint application at every completion step which also carries a lot of overhead. As expected, the growth rate of the chart size as a function of sentence length is higher for our parser than the baseline (see Figure 5.5).

To facilitate a comparison between the parsing times of our parser with the baseline, we plotted the ratio of mean parsing times per sentence length from the two parsers. The results are shown in Figure 5.6. As we can see from the graph, our parser suffers a parsing time penalty when the sentences are small and when they are long,

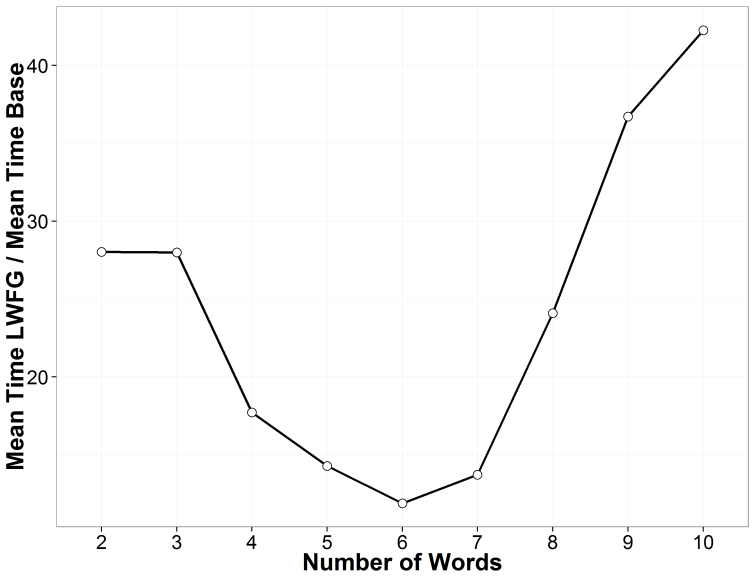


Figure 5.6: Ratio of mean parsing time per sentence length from our LWFG parser and the baseline CFG parser.

but performs relatively better for mid-length sentences. We believe these penalties correspond to the two factors that make our parser slower than an equivalent CFG parser. For short sentences, the chart sizes are in fact smaller for our parser because the grammar discards states that do not satisfy compositional constraints. In those cases, the large parsing time penalty is due to the additional constraint-application step which has a significant overhead. On the other hand, for longer sentences, the chart sizes are significantly larger for our parser because of the increased overhead of storing state-wise semantic information. This contributes to a significant penalty in parsing time which is bound to only increase further as the number of sentences are increased due to the worst-case exponential bound.

Regardless, the takeaway is still promising. Our parser was able to correctly parse all 7500 sentences, and was able to produce semantic information corresponding to them. In the future, we wish to evaluate our parser using a larger grammar and wider-range of sentences. It would also be interesting to see how our parser stacks against other semantic parsers.

6. Future Directions

Apart from implementational improvements, there are a number of possibilities to extend upon the work presented in this thesis, and in this chapter, we discuss some of them. The first goal is to implement coordinating conjunctions (such as, *and*, *or*, *but*, etc.) The current implementation is unable to handle those, largely due to the difficulty in mapping natural language conjunctions with their logical counterparts. For example, *and* is often used to imply order in sentences such as in sentences like, *they got married and had a child*. This means the semantic content of conjunctions needs to be carefully defined using contextual cues.

A more technical reason why our current implementation cannot handle conjunctions is because a unification framework does not easily conform with logical operations. For example, in the sentence *the boys and the girl ate the cake*, it is clear that the subject of the verb *ate* is *the boys and the girl*, and its semantic head is going to be a unified version of the heads of *the boys* and *the girl*. However, the way unification is implemented currently, such an operation would fail because *the boys* and *the girl* do not agree based on their grammatical number. Also, if the sentence was *the boys or the girl ate the cake*, the semantic head of the subject would need to indicate the possibility that either one of *the boys* and *the girl* could be the agent of the verb *ate*. Simple unification is unable to handle that requirement.

Another extension that we wish to incorporate is for the parser to be able to handle semantic interpretation constraints. As discussed earlier, these semantic interpretation constraints are currently predicates which can either succeed or fail, and are applied after the parser generates a complete parse. On succeeding, these constraints instantiate the semantic representations with concepts or slots in the semantic model (see Section 3.2). We wish to incorporate these constraints to the parser itself, so that we can detect nonsensical sentences at an early stage itself.

Additionally, the use of interpretation constraints will also allow the parser to perform efficiently by not storing nonsensical phrases. As discussed earlier, the LWFG parser suffers from having to store different states for all possibilities when

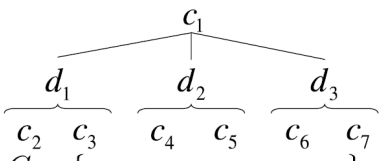
parsing an ambiguous input. Using interpretation constraints, the parser will be able to discard implausible structures early, which should significantly reduce the chart size. Consequently, this should reduce overall parsing time.

A long-term goal of our project is to be handle large grammars to enable broad-coverage parsing. The current implementation can handle small to mid-sized grammars, but more work needs to be done to enable sizes capable of parsing general text. As mentioned earlier, other work on constraint grammar parsing has attempted to solve this problem by using dynamic programming techniques and creating packed representations of parses (e.g., Maxwell III and Kaplan, 1995; Miyao and Tsujii, 2002). The packed representation of Maxwell III and Kaplan (1995) combines sets of parses by identifying "features" of the parse structures, where features are usually structures that combine to generate a complete parse structure, such as chart edges. They then defined the packed representation as $R = (\mathcal{F}', X, N, \alpha)$, where:

- $\mathcal{F}' \supseteq \mathcal{F}(y)$ is a finite set of features,
- X is a finite vector of *variables*,
- N isa finite set of conditions on X called *no-goods*, and
- α is a function that maps each feature $f \in \mathcal{F}'$ to a condition α_f on X.

Here, the "variables" X act as non-linguistic attributes defined only for the packed representations which identify which parse a set of features corresponds to. The conditions on X are functions from $\mathcal{X}$ to $\{0, 1\}$, where $\mathcal{X}$ defines the range of X. The *no-goods* are used to identify parses. A vector of values x satisfies a *no-goods* N iff $N(x) = 1$, where $N(x) = \prod_{\eta \in N} \eta(x)$. Each x that satisfies the *no-goods* identifies a parse $\omega(x) = \{f \in \mathcal{F}' | \alpha_f(x) = 1\}$. In addition, each parse is uniquely identified by a vector of values. That is: $\forall x, x' \in \mathcal{X}$ if $N(x) = N(x') = 1$ and $\omega(x) = \omega(x')$ then $x = x'$. Finally, a packed representation R represents the set of parses $\Omega(R)$ that are identified by values that satisfy the *no-goods*, i.e. $\Omega(R) = \{\omega(x) | x \in \mathcal{X}, N(x) = 1\}$.

A slightly different approach is that of Miyao and Tsujii (2002, 2005) who defined *feature forests* to represent a set of parses. A feature forest is defined as a tuple



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

Figure 6.1: An example feature forest from Miyao and Tsujii (2008)

$\langle C, D, r, \gamma, \delta \rangle$, where:

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

Essentially, a feature forest represents a set of trees of conjunctive nodes in a packed form. Each conjunctive node corresponds to an entity in a parse tree. On the other hand, disjunctive nodes are used to enumerate multiple possibilities. The two daughter functions are used to represent immediate relations of conjunctive and disjunctive nodes. As should be easy to tell from the function definitions, every conjunctive node can only have disjunctive children, and vice versa. Furthermore, only conjunctive nodes can act as leaf nodes. Additionally, with every feature forest, there is an associated feature function defined over the set of conjunctive nodes:

$$f_i : C \mapsto \mathbb{R}$$

A feature forest can be "unpacked" to produce the constituent trees, by enumerating the alternatives at disjunctive nodes. For example, Figure 6.1 shows an example feature forest, and Figure 6.2 shows its unpacked trees.

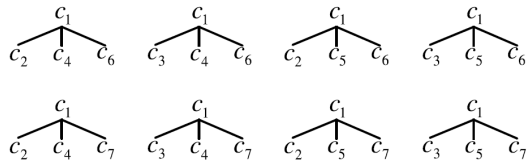


Figure 6.2: Unpacked trees corresponding to the feature forest shown in Figure 6.1

We wish to use either one of the two or a similar technique for LWFGs. On a higher-level examination, feature forests seem to be the more promising of the two and offer a slightly more general framework. Another aspect that makes feature forests promising is the fact that the authors, Miyao and Tsujii (2002, 2005), have also discussed a dynamic programming algorithm to estimate a probability distribution over the set of feature forests. Their algorithm is analogous to the inside-outside algorithm used for probabilistic context-free grammars.

This brings us to another of our long-term goals, which is to develop a stochastic framework for LWFGs. As discussed earlier, one appealing aspect of LWFGs is that their learnability is guaranteed and tractable, which makes them well-suited for resource-poor settings. However, for languages that do not suffer from a dearth of resources, it is desirable to use the available information on the relative frequencies of various constructions and usages. For ambiguous sentences, being able to determine which interpretation is most likely is particularly interesting, and currently LWFGs are unable to utilize such information.

There are many directions in which we can go to develop a stochastic LWFG. At the moment, the feature forest framework of Miyao and Tsujii (2002, 2005) seems most promising because of its general nature. The challenge is to not just use feature forests to define a probability distribution over parse fragments (which are called "features" by both Miyao and Tsujii (2002, 2005) and Maxwell III and Kaplan (1995)), but to also define a probability distribution over various semantic interpretation constraints. As mentioned earlier, in the current description of the LWFG, the semantic interpretation constraints are predicates that can either succeed or fail. A failure implies the interpretation of the given sentence is not valid in the defined semantic model ("world truth"). Instead of simply saying that a sentence is implausible, we would like the parser to be

able to quantify the degree of implausibility of various interpretations.

7. Conclusion

In this thesis, we have described a new semantic parser for natural languages that simultaneously performs a syntactic and an ontology-based semantic analysis on sentences. Our implementation was derived from that of Muresan (2006) who described a bottom-up chart parser for an constraint-based grammatical framework called Lexicalized Well-Founded Grammar (Muresan, 2006, 2010). Our work extends theirs by revising the parsing algorithm to detect semantic violations early, thus reducing the number of derivations that the parser entertains while parsing. In addition, our parser is implemented using Python's NLTK library (Bird et al., 2009) which allows for an easy integration with the open-source NLP community. Furthermore, our choice of an imperative language like Python, unlike a declarative language like Prolog, allows for a greater scope of scalability to larger-sized grammars for broad-coverage parsing. Lastly, our choice will facilitate a more straight-forward extension to a stochastic implementation of the LWFG framework.

A. Appendices

A.1 Sample Lexicon of the LWFG

```
det -> "a"
        [h=[nr=sg,cat=det,mod=?x]]
        ?x.det=a

det -> "an"
        [h=[nr=sg,cat=det,mod=?x]]
        ?x.det=an

det -> "the"
        [h=[nr=?anon1,cat=det,mod=?x]]
        ?x.det=the

verb -> "like"
        [h=[head=?x,vtype=norm,headS=?y,val=tv,vf=bse,cat=verb,headC=?z,
            aux=no,voice=act,vft=nf]]
        ?x.agt=?y,?x.pnt=?z,?x.is_a=like

verb -> "like"
        [h=[head=?x,vtype=norm,headS=?y,val=tv,vf=bse,neg=no,cat=verb,pf
            =no,pg=no,aux=no,headC=?z,voice=act,vft=nfin]]
        ?x.vft=bse,?x.pnt=?z,?x.is_a=like,?x.agt=?y

verb -> "like"
        [h=[head=?x,vtype=norm,headS=?y,val=tv,vf=no,neg=no,cat=verb,
            pers=[?anon12,n3],pf=no,pg=no,headC=?z,aux=no,tense=pr,nr=sg,
            voice=act,vft=fin]]
        ?x.tense=pr,?x.agt=?y,?x.pnt=?z,?x.is_a=like

verb -> "like"
        [h=[head=?x,vtype=norm,headS=?y,val=tv,vf=no,neg=no,cat=verb,
            pers=[?anon13,?anon13],pf=no,pg=no,headC=?z,aux=no,tense=pr,
            nr=pl,voice=act,vft=fin]]
        ?x.tense=pr,?x.agt=?y,?x.pnt=?z,?x.is_a=like

adj -> "short"
        [h=[head=?x,cat=adj,mod=?y]]
        ?x.is_a=short,?y.Has_prop=?x

adj -> "small"
        [h=[head=?x,cat=adj,mod=?y]]
        ?x.is_a=small,?y.Has_prop=?x

noun -> "boys"
        [h=[nr=pl,cat=noun,head=?x,mod=?y]]
        ?x.is_a=boys,?y.Has_prop=?x

noun -> "boys"
        [h=[case=[?anon80,ng],count=y,modr=no,hum=y,det=no,head=?x,cat=
            noun,pers=[?anon79,3],nr=pl,gen=male]]
        ?x.is_a=boys

noun -> "boy"
        [h=[nr=sg,cat=noun,head=?x,mod=?y]]
        ?x.is_a=boy,?y.Has_prop=?x
```

```
noun -> "boy"
        [h=[case=[?anon84,ng],count=y,modr=no,hum=y,det=no,head=?x,cat=
            noun,pers=[?anon83,3],nr=sg,gen=male]]
        ?x.is_a=boy
```

A.2 Sample Production rules of the LWFG

```
sbj -> n
        [h=[case=[n,ng],head=?d,hum=?h,cat=sbj,pers=[?e,?f],nr=?g,dets=y
            ,stype=s], h1=[case=[n,ng],head=?d,hum=?h,cat=n,pers=[?e,?f],
            nr=?g]]

n -> a n
        [h2=[case=[?k,?l],count=?o,head=?e,hum=?m,det=no,modr=?g,cat=n,
            pers=[?h,?i],nr=?j,gen=?n], h=[case=[?k,?l],count=?o,head=?e,
            hum=?m,det=no,modr=?g,cat=n,pers=[?h,?i],nr=?j,gen=?n], h1=[
            cat=a,mod=?e]]

a -> adj
        [h=[head=?d,cat=a,mod=?e], h1=[head=?d,cat=adj,mod=?e]]

n -> nc
        [h=[case=[?j, ?k], cat='n', count=?n, det=?f, gen=?m, head=?d,
            hum=?l, modr=?e, nr=?i, pers=[?g, ?h]], h1=[case=[?j, ?k],
            cat='nc', count=?n, det=?f, gen=?m, head=?d, hum=?l, modr=?e,
             nr=?i, pers=[?g, ?h]]]

nc -> na nc
        [h=[case=[?l, ?m], cat='nc', count=?p, det=?h, gen=?o, head=?e,
            hum=?n, modr=?g, nr=?k, pers=[?i, ?j]], h1=[cat='na', mod=?e,
             nr='sg'], h2=[case=[?l, ?m], cat='nc', count=?p, det=?h, gen
            =?o, head=?e, hum=?n, modr=?g, nr=?k, pers=[?i, ?j]]]

nc -> noun
        [h=[case=[?i, ?j], cat='nc', count=?m, det='no', gen=?l, head=?d
            , hum=?k, modr=?e, nr=?h, pers=[?f, ?g]], h1=[case=[?i, ?j],
            cat='noun', count=?m, det='no', gen=?l, head=?d, hum=?k, modr
            =?e, nr=?h, pers=[?f, ?g]]]

na -> na na
        [h=[cat='na', head=?e, mod=?g, nr='sg'], h1=[cat='na', mod=?e,
            nr='sg'], h2=[cat='na', head=?e, mod=?g, nr='sg']]

na -> noun
        [h=[cat='na', head=?d, mod=?e, nr='sg'], h1=[cat='noun', head=?d
            , mod=?e, nr='sg']]
```

Bibliography

Abney, S. (1997). Stochastic attribute-value grammars. *Computational Linguistics*, 23:597–617.

Baker, C. F., Fillmore, C. J., and Lowe, J. B. (1998). The berkeley framenet project. In *Proceedings of the 17th international conference on Computational linguistics-Volume 1*, pages 86–90. Association for Computational Linguistics.

Barnickel, T., Weston, J., Collobert, R., Mewes, H.-W., and Stümpflen, V. (2009). Large scale application of neural network based semantic role labeling for automated relation extraction from biomedical texts. *PLoS One*, 4(7):e6393.

Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media Inc.

Boas, H. C. (2002). Bilingual framenet dictionaries for machine translation. In *LREC*.

Bresnan, J. (1982). Control and complementation. In Bresnan, J., editor, *The mental representation of grammatical relations*, pages 282–390. Cambridge, MA.

Bresnan, J. (2001). *Lexical-functional syntax*. Blackwell, Malden, MA.

Brown, P. F., Pietra, V. J. D., Pietra, S. A. D., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311.

Chen, D. L. and Mooney, R. J. (2008). Learning to sportscast: a test of grounded language acquisition. In *Proceedings of the 25th international conference on Machine learning*, pages 128–135. ACM.

Chen, J. and Rambow, O. (2003). Use of deep linguistic features for the recognition and labeling of semantic arguments. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pages 41–48. Association for Computational Linguistics.

Christensen, J., Soderland, S., Etzioni, O., et al. (2010). Semantic role labeling for open information extraction. In *Proceedings of the NAACL HLT 2010 First International Workshop on Formalisms and Methodology for Learning by Reading*, pages 52–60. Association for Computational Linguistics.

Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pages 16–23.

Fleischman, M., Kwon, N., and Hovy, E. (2003). Maximum entropy models for framenet classification. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pages 49–56. Association for Computational Linguistics.

Gazdar, G. (1985). *Generalized phrase structure grammar*. Harvard University Press.

Ge, R. and Mooney, R. J. (2005). A statistical semantic parser that integrates syntax and semantics. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, pages 9–16. Association for Computational Linguistics.

Geman, S. and Johnson, M. (2002). Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 279–286. Morgan Kaufmann.

Gildea, D. and Jurafsky, D. (2002). Automatic labeling of semantic roles. *Computational linguistics*, 28(3):245–288.

Hirst, G. (1983). A foundation for semantic interpretation. In *Proceedings of the 21st annual meeting on Association for Computational Linguistics*, pages 64–73. Association for Computational Linguistics.

Johnson, M. (2003). Learning and parsing stochastic unification-based grammars. In *Proceedings of the 16th Annual Conference on Learning Theory and 7th Kernel Workshop, COLT*, pages 671–683, Washington, DC, USA.

Johnson, M., Geman, S., Canon, S., Chi, Z., and Riezler, S. (1999). Estimators for stochastic "unification-based" grammars.

Kay, M. (1973). The mind system. In Rustin, R., editor, *Natural Language Processing*, pages 155–188. Algorithmics Press, New York.

Kay, M. (1984). Functional unification grammar: A formalism for machine translation. In *Proceedings of the 10th International Conference on Computational Linguistics and 22nd annual meeting on Association for Computational Linguistics*, pages 75–78. Association for Computational Linguistics.

Kim, J. and Mooney, R. J. (2012). Unsupervised pcfg induction for grounded language learning with highly ambiguous supervision. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 433–444. Association for Computational Linguistics.

Kwiatkowski, T., Zettlemoyer, L., Goldwater, S., and Steedman, M. (2010). Inducing probabilistic ccg grammars from logical form with higher-order unification. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 1223–1233. Association for Computational Linguistics.

Liang, P., Jordan, M. I., and Klein, D. (2009). Learning semantic correspondences with less supervision. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 91–99. Association for Computational Linguistics.

Lu, W., Ng, H. T., Lee, W. S., and Zettlemoyer, L. S. (2008). A generative model for parsing natural language to meaning representations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 783–792. Association for Computational Linguistics.

Maxwell III, J. T. and Kaplan, R. M. (1995). A method for disjunctive constraint satisfaction. In Dalrymple, M., Kaplan, R. M., Maxwell III, J. T., and Zaenen, A., editors, *Formal Issues in Lexical-Functional Grammar*, number 47 in CLSI Lecture Notes Series, chapter 14, pages 381–481. CSLI Publications.

Meyers, A., Reeves, R., Macleod, C., Szekely, R., Zielinska, V., Young, B., and Grishman, R. (2004). The nombank project: An interim report. In *HLT-NAACL 2004 workshop: Frontiers in corpus annotation*, pages 24–31.

Miyao, Y. and Tsujii, J. (2002). Maximum entropy estimation for feature forests. In *Proc. of HLT*, volume 2.

Miyao, Y. and Tsujii, J. (2005). Probabilistic disambiguation models for wide-coverage hpsg parsing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 83–90. Association for Computational Linguistics.

Miyao, Y. and Tsujii, J. (2008). Feature forest models for probabilistic hpsg parsing. *Computational Linguistics*, 34(1):35–80.

Muresan, S. (2006). *Learning constraint-based grammars from representative examples: Theory and applications*. PhD thesis, Columbia University, NY.

Muresan, S. (2010). A learnable constraint-based grammar formalism. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING 2010)*.

Muresan, S. (2011). Learning for deep language understanding. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three*, pages 1858–1865. AAAI Press.

Muresan, S. and Rambow, O. (2007). Grammar approximation by representative sublanguage: A new model for language learning. In *Annual Meeting – Association for Computational Linguistics*, volume 45, page 832.

Narayanan, S. and Harabagiu, S. (2004). Question answering based on semantic structures. In *Proceedings of the 20th international conference on Computational Linguistics*, page 693. Association for Computational Linguistics.

Och, F. J. and Ney, H. (2003). A systematic comparison of various statistical alignment models. *Computational linguistics*, 29(1):19–51.

Palmer, M., Gildea, D., and Kingsbury, P. (2005). The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106.

Pereira, F. C. and Warren, D. H. (1983). Parsing as deduction. In *Proceedings of the 21st annual meeting on Association for Computational Linguistics*, pages 137–144. Association for Computational Linguistics.

Pereira, F. C. N. and Warren, D. H. D. (1980). Definite clause grammars for language analysis – a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13(3):231–278.

Pollard, C. and Sag, I. A. (1987). *Information-based syntax and semantics*. Number 13 in CSLI Lecture Note Series. Chicago University Press, Chicago.

Pollard, C. and Sag, I. A. (1994). *Head-driven phrase structure grammar*. The University of Chicago Press, Chicago.

Ponzetto, S. P. and Strube, M. (2006). Exploiting semantic role labeling, wordnet and wikipedia for coreference resolution. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 192–199. Association for Computational Linguistics.

Poon, H. and Domingos, P. (2009). Unsupervised semantic parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 1–10. Association for Computational Linguistics.

Pradhan, S. S. (2006). *Robust semantic role labeling*. PhD thesis, University of Colorado.

Schuler, K. K. (2005). *VerbNet: A broad-coverage, comprehensive verb lexicon*. PhD thesis.

Shen, D. and Lapata, M. (2007). Using semantic roles to improve question answering. In *EMNLP-CoNLL*, pages 12–21.

Shieber, S., Uszkoreit, H., Pereira, F., Robinson, J., and Tyson, M. (1983). The formalism and implementation of patr-ii. In Grosz, B. and Stickel, M., editors, *Research on Interactive Acquisition and Use of Knowledge*, pages 39–79. Menlo Park, CA.

Shieber, S. M., Schabes, Y., and Pereira, F. C. (1995). Principles and implementation of deductive parsing. *The Journal of Logic Programming*, 24(1):3–36.

Sondheimer, N. K., Weischedel, R. M., and Bobrow, R. J. (1984). Semantic interpretation using kl-one. In *Proceedings of the 10th International Conference on Computational Linguistics and 22nd annual meeting on Association for Computational Linguistics*, pages 101–107. Association for Computational Linguistics.

Steedman, M. (1996). *Surface structure and interpretation*, volume 30. MIT press Cambridge, MA.

Steedman, M. (2001). *The syntactic process*. The MIT press.

Surdeanu, M., Harabagiu, S., Williams, J., and Aarseth, P. (2003). Using predicate-argument structures for information extraction. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 8–15. Association for Computational Linguistics.

Surdeanu, M., Màrquez, L., Carreras, X., and Comas, P. (2007). Combination strategies for semantic role labeling. *J. Artif. Intell. Res.(JAIR)*, 29:105–151.

Toutanova, K., Haghighi, A., and Manning, C. D. (2008). A global joint model for semantic role labeling. *Computational Linguistics*, 34(2):161–191.

Vickrey, D. (2010). *Learning Structured Probabilistic Models for Semantic Role Labeling*. PhD thesis, Stanford University.

Warren, D. S. and Friedman, J. (1982). Using semantics in non-context-free parsing of montague grammar. *Computational Linguistics*, 8(3-4):123–138.

Wong, Y. W. and Mooney, R. J. (2006). Learning for semantic parsing with statistical machine translation. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 439–446. Association for Computational Linguistics.

Wong, Y. W. and Mooney, R. J. (2007). Learning synchronous grammars for semantic parsing with lambda calculus. In *Annual Meeting-Association for computational Linguistics*, volume 45, page 960.

Xue, N. and Palmer, M. (2004). Calibrating features for semantic role labeling. In *EMNLP*, pages 88–94.

Zettlemoyer, L. S. and Collins, M. (2007). Online learning of relaxed ccg grammars for parsing to logical form. In *In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL-2007*. Citeseer.

Zettlemoyer, L. S. and Collins, M. (2009). Learning context-dependent mappings from sentences to logical form. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 976–984. Association for Computational Linguistics.

Zettlemoyer, L. S. and Collins, M. (2012). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *arXiv preprint arXiv:1207.1420*.