

# A CCG-Based Method for Training a Semantic Role Labeler in the Absence of Explicit Syntactic Training Data

DISSERTATION

Presented in Partial Fulfillment of the Requirements for  
the Degree Doctor of Philosophy in the  
Graduate School of The Ohio State University

By

Stephen A. Boxwell, M.A., B.A.

Graduate Program in Linguistics

The Ohio State University

2011

Dissertation Committee:

Prof. Chris Brew, Co-Adviser

Prof. Michael White, Co-Adviser

Prof. Simon Dennis

Prof. William Schuler

© Copyright by  
Stephen A. Boxwell  
2011

## ABSTRACT

Treebanks are a necessary prerequisite for many NLP tasks, including, but not limited to, semantic role labeling. For many languages, however, treebanks are either nonexistent or too small to be useful. Time-critical applications may require rapid deployment of natural language software for a new critical language – much faster than the development time of a traditional treebank. This dissertation describes a method for generating a treebank and training syntactic and semantic models using only semantic training information – that is, no human-annotated syntactic training data whatsoever. This will greatly increase the speed of development of natural language tools for new critical languages in exchange for a modest drop in overall accuracy. Using Combinatory Categorical Grammar (CCG) in concert with Propbank semantic role annotations allows us to accurately predict lexical categories in combination with a partially hidden Markov model. By training the Berkeley parser on our generated syntactic data, we can achieve SRL performance of 65.5% without using a treebank, as opposed to 74% using the same feature set with gold-standard data.

## ACKNOWLEDGMENTS

I am deeply indebted to my advisors, Chris Brew and Mike White. Chris supported me for several quarters, introduced me to cryptography class which I now teach, and taught me the importance of correct evaluation and understanding exactly why something is (or isn't) working. My other advisor Mike White taught me the paramount importance of attention to detail and error analysis. I am of course indebted to the rest of my committee as well: William Schuler, whose valuable insights into parsing, semantic role labeling, and EM have contributed immensely to the development of this work, and Simon Dennis, whose extensive work on the learning of language from semantic patterns contributed immensely as well. I am also grateful to Jason Baldrige, whose expertise on unsupervised and semi-supervised CCG supertagging was of tremendous value. From the College of William and Mary, I am very grateful to Ann Reed and Jack Martin for introducing me to linguistics in the first place, and to Virginia Torczon, who gave me my love for computer science. I am also very appreciative of the help and support of my fellow students at Ohio State – I am especially grateful to DJ Hovermale, whose support and encouragement have been invaluable along the way, and to Dennis Mehay, whose technical expertise has gotten me out of many tough spots. Finally, I am deeply indebted to my wife Alison, who for many years has patiently endured my bizarre fascination with words and grammar.

## VITA

May 2006 .....B.A., Linguistics  
The College of William and Mary,  
Williamsburg, VA, USA

March 2011 .....M.A., Linguistics  
The Ohio State University,  
Columbus, OH, USA

## PUBLICATIONS

### Conference Papers

Stephen A Boxwell, Chris Brew, Jason Baldrige, Dennis N Mehay, and Sujith Ravi. *Semantic Role Labeling for CCG Without Treebanks?* In Proceedings of the 2011 International Joint Conference on Natural Language Processing (IJCNLP-11), Chiang Mai, Thailand, November 2011.

Stephen A Boxwell, Dennis N Mehay, and Chris Brew. *What a Parser can Learn from a Semantic Role Labeler and Vice Versa.* In Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP-10), Cambridge, MA, October 2010.

Stephen A Boxwell and Chris Brew. *A Pilot Arabic CCGbank.* In Proceedings of the Seventh International Language Resources and Evaluation Conference (LREC-10), Valetta, Malta, May 2010.

Stephen A. Boxwell, Dennis N. Mehay, and Chris Brew. *Brutus: A Semantic Role Labeling System Incorporating CCG, CFG, and Dependency Features.* In Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics (ACL-09), Singapore, August 2009.

Stephen A. Boxwell and Michael White. *Projecting Propbank Roles onto the CCG-bank*. In Proceedings of the Sixth International Language Resources and Evaluation Conference (LREC-08), Marrakech, Morocco, May 2008.

## **FIELDS OF STUDY**

Major Field: Computational Linguistics

# TABLE OF CONTENTS

	Page
Abstract . . . . .	ii
Acknowledgments . . . . .	iii
Vita . . . . .	iv
List of Tables . . . . .	ix
List of Figures . . . . .	xi
Chapters:	
1. Introduction . . . . .	1
2. Combinatory Categorical Grammar . . . . .	8
2.1 Introduction to CCG . . . . .	8
2.2 Function Application . . . . .	10
2.3 Function Composition . . . . .	12
2.4 Type Raising . . . . .	16
2.5 CCG Syntactic Dependencies . . . . .	20
2.6 Threading Heads and Dependencies . . . . .	23
2.6.1 Adjunct Categories . . . . .	24
2.6.2 Long-Distance Dependencies . . . . .	25
2.7 Why CCG? . . . . .	25
2.8 Conclusion . . . . .	28

3.	Semantic Roles . . . . .	29
3.1	Introduction . . . . .	29
3.2	Semantic Role Paradigms . . . . .	30
3.3	Automatic Semantic Role Labeling . . . . .	33
3.4	Previous Approaches to Semantic Role Labeling . . . . .	34
3.5	The Brutus Semantic Role Labeler . . . . .	44
3.5.1	Predicate Labeling . . . . .	44
3.5.2	Semantic Role Labeling . . . . .	51
3.6	Future Work . . . . .	54
3.7	Conclusion . . . . .	55
4.	Automatic Parsing in CCG . . . . .	56
4.1	Introduction . . . . .	56
4.2	Hypergraph-Based Chart Packing . . . . .	59
4.3	Re-Ranking Candidate Parses Using Discriminative Model . . . . .	65
4.4	Packed Chart Parsing in CCG . . . . .	67
4.5	Semantic Role Labeling and Packed Charts . . . . .	72
4.6	Other Approaches to Parsing in CCG . . . . .	76
4.7	Conclusion . . . . .	79
5.	Learning to Do Without a Treebank . . . . .	80
5.1	The Problem with Treebanks . . . . .	80
5.2	Constructing Treebanks . . . . .	81
5.3	Predicting CCG Lexical Categories . . . . .	81
5.3.1	Verbal Categories . . . . .	82
5.3.2	Auxiliary Verbs . . . . .	86
5.3.3	Prepositional categories . . . . .	86
5.3.4	Common Closed-Class Words . . . . .	89
5.3.5	Predicting Categories with a Partially Hidden Markov Model . . . . .	90
5.4	Predicting Syntactic Structure . . . . .	96
5.4.1	Experiment 1: Unguided Parsing . . . . .	96
5.4.2	Experiment 2: Constituent Guided Parsing . . . . .	98
5.4.3	Experiment 3: Training Semantic Models over a Packed Chart . . . . .	101
5.5	Future Work . . . . .	104
5.6	Conclusion . . . . .	106

6.	Generating Syntactic Analyses for Novel Text . . . . .	107
6.1	Experiment 1: Training an HMM Supertag Model . . . . .	107
6.2	Experiment 2: Using the Berkeley Parser to Generate High Quality Analyses . . . . .	108
6.2.1	The Berkeley Parser . . . . .	109
6.2.2	Using the Berkeley Parser as a CCG Parser . . . . .	110
6.3	Error Analysis . . . . .	113
6.4	Future Work . . . . .	114
6.5	Conclusion . . . . .	114
7.	Conclusion . . . . .	116
7.1	Semantic Role Labeling in CCG . . . . .	116
7.2	Generating CCG Supertags . . . . .	116
7.3	Generating a Treebank . . . . .	117
7.4	Future Work . . . . .	117
	References . . . . .	118

## LIST OF TABLES

Table	Page
2.1 Some common CCG categories and some words and phrases that they include. . . . .	9
2.2 A variety of legal type-raising operations applied to the CCG category NP. Some, like $S/(S\backslash NP)$ , are commonly used in English. Others, like $PP/(PP\backslash NP)$ , serve only to demonstrate that anything can be type-raised over, even if the result is unlikely to be useful. . . . .	19
3.1 Performance of the predicate labeler on the development set (WSJ 00). The leftmost column shows the accuracy of identifying words as predicates. The center column shows the performance of identification and stemming (ignoring verbal disambiguation). The rightmost column shows the complete system performance. . . . .	51
3.2 Our approach for disambiguating predicates outperforms two baseline approaches. The leftmost column shows the performance of a classification module that always predicts sense #1, the center column shows the performance of always predicting the most frequent sense for that predicate, and the rightmost column reproduces the performance of the classifier suite. Identification and Stemming are held constant, using the same method as in table 3.1. . . . .	52
3.3 SRL performance, compared to the system of Punyakanok et al. [2008]	54
5.1 The complete baseline model, which requires no syntactic training data. The substitution combinator is used to model parasitic gaps in English, which are so rare that we make the pragmatic decision to disallow substitution entirely. . . . .	97

5.2	The performance of the unguided parse method for training SRL models, compared to models trained from gold-standard parses, on tagging gold standard analyses from the development set. The columns marked <i>Identification</i> indicate cases where the role boundaries are correct, whether or not the role label (Arg0, Arg1, ArgM-TMP, etc) is correct. The columns marked <i>ID+Classification</i> indicate cases where the span and label are both correct. . . . .	98
5.3	The performance of the guided parse method for training SRL models, compared to the previous models. Enforcing role boundaries on the training sentences improves SRL performance by about 1%. . . . .	101
5.4	The performance of the guided parse method for training SRL models, compared to the previous models. Enforcing role boundaries on the training sentences improves SRL performance by about 1%. . . . .	104
6.1	The performance of the three baseline supertag models, using a beta value of 0.1. Even the best baseline model achieves only 60% sentence coverage, which is unacceptably low. . . . .	108
6.2	The performance of the Berkeley parser on CCG-style analyses when trained on gold-standard CCG parses, our smaller generated treebank from section 5.4.2, and the chart-based approach from section 5.4.3. . . . .	111
6.3	The performance of the models from table 6.2 on the test set (WSJ section 23). . . . .	112
6.4	The effect of the reduced training set size is demonstrated by the SUB-GOLD model. Using an induced model instead of gold-standard data accounts for about 5.5% of the difference in performance. . . . .	113

## LIST OF FIGURES

Figure	Page
1.1 Two reasonable analyses for the sentence <i>she helped students do better</i> . For our purposes, it is less important that we choose one over the other as it is that we choose one consistently. . . . .	4
1.2 An erroneous analysis for the phrase <i>the company stopped using asbestos on 1956</i> . The bracketing suggests that the temporal modifier applies to <i>using</i> , rather than <i>stopped</i> . . . . .	4
1.3 There are two possible analyses for <i>a group of workers exposed to asbestos</i> . The semantic difference between the two is very slight – the important thing is that the syntactic and semantic annotations be consistent. . . . .	6
2.1 A simple CCG derivation. . . . .	10
2.2 The general case of modus ponens, a simple argument form . . . . .	11
2.3 The general cases of the two main CCG combinators, <i>forward application</i> (a) and <i>backward application</i> (b). . . . .	11
2.4 CCG categories often select for complex arguments.. . . . .	12
2.5 The general case of a hypothetical syllogism, a simple argument form.	12
2.6 There is often more than one way to arrive at a particular conclusion.	13
2.7 A simple sentence, with the appropriate CCG categories assigned to the lexical items. . . . .	13
2.8 Two valid analyses of the phrase <i>settled in Columbus</i> . . . . .	14

2.9	The general cases of the function composition CCG combinators, <i>forward composition</i> (a) and <i>backward composition</i> (b). Backward composition is very rare in English. . . . .	15
2.10	Function composition can be used to combine phrases so that they can be coordinated without hypothesizing invisible elements. The $\Phi$ symbol indicates coordination. . . . .	16
2.11	Two equivalent analyses for <i>dogs sleep</i> . In the first analysis, the verb combines with the subject using backward application. In the second analysis, the subject is first type-raised over the verb, and then combines with the verb using forward application. Notice that the direction of function application is reversed. . . . .	18
2.12	The general cases of CCG type-raising, called <i>forward type-raising</i> and <i>backwards type-raising</i> . Note that here T denotes a metavariable over all well-formed CCG categories. . . . .	18
2.13	By using forward type-raising and forward composition in quick succession, we can assign the category S[dcl]/NP to <i>John likes</i> . This allows the relativizer <i>that</i> to select for the category S[dcl]/NP. . . . .	21
2.14	Four phrases in which the relationship between <i>read</i> and <i>papers</i> is the same. Using a context-free grammar, it is difficult to generalize these four cases. . . . .	21
2.15	Two trees representing the CCG category (S[dcl]\NP)/NP (a) and the CCG category NP (b). In the verbal category, the two NP arguments are assigned unfilled dependencies: $\langle 2, reads, (S\NP)/NP, 1 \rangle$ and $\langle 2, reads, (S\NP)/NP, 2 \rangle$ . The result category and, implicitly, all the nodes between it and the root node, are assigned a head: $\langle 2, reads \rangle$ . In the nominal category, there are no arguments, so there are no unfilled dependencies. The single node is assigned the head representing that lexical entry: $\langle 2, papers \rangle$ . . . . .	23
2.16	An derivation of <i>the big dog</i> predicting an erroneous dependency graph. Although the derivation correctly predicts that the span is a sentence, the heads and dependencies are problematic. . . . .	26

2.17	A corrected version of the derivation in figure 2.16. Notice that the superscript coindexation on the categories for <i>the</i> and <i>big</i> ensure that the correct heads are passed up to the result categories, and the noun phrase has the correct head <i>dog</i> . . . . .	26
2.18	A derivation of the phrase <i>the sandwich that the dog ate</i> , demonstrating how the object dependency of <i>ate</i> can be threaded through <i>that</i> to reach <i>the sandwich</i> . . . . .	26
2.19	A derivation of the phrase <i>the sandwich that the dog ate</i> , demonstrating how the object dependency of <i>ate</i> can be threaded through <i>that</i> to reach <i>the sandwich</i> . . . . .	27
3.1	The tree path feature between <i>win</i> and <i>you</i> is $VB\uparrow VP\uparrow VP\uparrow S\downarrow NP\downarrow PRP$	37
3.2	The CCG dependency feature between <i>win</i> and <i>you</i> is $s[b]\backslash np.1.\rightarrow$ .	37
3.3	The CCG dependency feature between <i>reads</i> and <i>girl</i> is $(s[dcl]\backslash np)/np.1.\rightarrow$ in all three sentences. This is convenient, because the semantic relationship is the same as well. . . . .	38
3.4	A side effect of tagging multiple candidate parses is that the semantic roles sometimes interfere with each other. For example, a system presented with these two parses will likely predict a role for <i>pasta with sauce</i> (based on the first parse) and <i>pasta</i> (based on the second parse). These role assignments are mutually exclusive – they cannot both be correct. Punyakanok et al. [2008] uses Integer Linear Programming to resolve the conflicts. . . . .	42
3.5	In the first stage of the semantic role labeling process, candidate semantic roles are chosen by the identifier model. We have not yet decided which role (ARG0, ARG1, etc) each word plays, only that there is a role there. . . . .	53
3.6	In the second stage of the semantic role labeling process, the classifier model sorts the roles into ARG0, ARG1, etc. . . . .	54
4.1	Two possible analyses for the phrase <i>more intelligent administrators</i> , the first meaning “a greater number of intelligent administrators” and the second meaning “administrators who are more intelligent”. . . . .	56

4.2	A parse chart for <i>more intelligent administrators</i> . Notice that edges 7 and 8 both cover the same span and have the same category. . . . .	57
4.3	A parse chart for <i>more intelligent administrators supervised</i> . Notice that there are now three pairs of identical edges (8 and 9, 10 and 11, 12 and 13). Not only does this waste space, but it requires the parser to apply the same grammatical rule twice each time, including any expensive unification computation. . . . .	58
4.4	A parse chart for <i>more intelligent administrators supervised children with telescopes</i> . There are now four spanning analyses, each of which must be handled independently. . . . .	60
4.5	A packed parse chart for <i>more intelligent administrators</i> . Edge #7 contains a set of lists of daughters, instead of a single list of daughters. This makes it clear that edge #7 can be formed by combining edges 1 and 5, or it could be formed by combining edges 6 and 4. . . . .	61
4.6	A parse chart for <i>more intelligent administrators supervised children with telescopes</i> . All four analyses are represented in the chart, but there is only one edge for the subject, one for the predicate, and one for the complete sentence. If this sentence were embedded in a complement clause ( <i>The principal said that more intelligent administrators supervised children with telescopes</i> ), then it would not be necessary to attach the clause 4 times. . . . .	62
4.7	An example of forward application. The verbal argument np[acc] can unify with the target category np[acc], so the verb can discharge an argument and produce the category s[dc] \ np to cover the span. . . .	67
4.8	The ungrammatical sentence <i>I like she</i> is blocked by the feature unification – np[acc] on the verbal argument cannot unify with np[nom] on the target category. . . . .	68
4.9	A simplified tree representation of the CCG category (S[dc] \ NP) / NP (a) and the CCG category NP (b). In the verbal category, the two NP arguments are assigned unfilled dependencies, $\langle (S \setminus NP) / NP.1 \rangle$ and $\langle (S \setminus NP) / NP.2 \rangle$ . The result category and, implicitly, all the nodes between it and the root node, are assigned a head. In the nominal category, there are no arguments, so there are no unfilled dependencies. The single node is assigned the head representing that lexical entry. .	69

4.10	There are two ways to obtain the $s[dcl]\backslash np$ category for the 1-3 span. The first corresponds to the intended reading, the second corresponds to the unlikely reading that I am storing some of the abstract noun <i>dance</i> for later in the winter. Should they be packed into a single edge?	70
4.11	The $S[dcl]\backslash NP$ category corresponding to the intended reading has two unfilled dependencies in the subject position – one for the modal verb, and one for the main verb. The $S[dcl]\backslash NP$ category for the unlikely reading has only one dependency, originating from the declarative verb <i>can</i> . Because the unfilled dependencies are not equivalent, and therefore they may produce different results later on in the derivation, these two categories cannot be packed into the same edge.	71
4.12	In the context of a packed chart, it is meaningless to speak of a treepath between <i>supervised</i> and <i>administrators</i> because multiple analyses are “packed” under a single category.	74
4.13	Two analyses for <i>the company stopped using asbestos</i> . Depending on how <i>in 1956</i> is attached, it will be labeled either as the temporal modifier of <i>stop</i> or <i>use</i> . Because <i>stop</i> is more likely to be modified by a temporal modifier than <i>use</i> , we can choose the parse that predicts this role.	77
5.1	Two possible analyses for a simple sentence. The numbers in subscript indicate the penalty assigned to that edge. The baseline model assigns a penalty of 0 to the normal form parse (on the left), and a penalty of 3 to the non-normal form parse (on the right).	97
5.2	Two possible analyses for the phrase <i>the unit of the company that makes cigarettes</i> . According to the CCGbank, the first analysis is correct, but both analyses yield reasonable semantics.	100
5.3	The NP <i>the company that makes cigarettes</i> is in violation of the role boundaries of Arg0. Therefore, it should not be included in the final analysis.	101
5.4	A supertagging error in this sentence prevents us from producing a spanning analysis. Fortunately, we can still get useful information from the edges that we can generate.	103

6.1	A selection of CFG-style rules learned by the Berkeley parser when trained on the CCG derivations. The Berkeley parser treats the categories as atomic, and therefore does not attempt to make any generalizations about forward application. . . . .	110
6.2	CCG derivations must be represented as PTB-style trees before they are given to the Berkeley parser. . . . .	111
6.3	The Berkeley parser makes the decision to assign $(S[\text{adj}]\backslash NP)/NP$ to <i>opposed</i> . Because the verbal category prediction process never assigns $S[\text{adj}]$ as the base of a verbal category, the $(S[\text{adj}]\backslash NP)/(S[\text{adj}]\backslash NP)$ category is unlikely to be chosen for a semantic role. . . . .	115

## CHAPTER 1: Introduction

Natural language software tools are useful for many practical applications, but often require extensively annotated resources that come at an enormous cost. Modern data driven approaches to semantic role labeling, for example, require at least the following steps to develop training data:

- Obtain a large amount of text in the language and genre you wish to work on. Newswire is an appealing choice, as it is cheap, plentiful, and fluent.
- Perform a first-pass annotation of the words themselves, assigning them each a grammatical category. Decisions must be made about the nature and granularity of the tagset which are not easily changed later. The process remains noisy under the best of circumstances.
- Assign each sentence a syntactic structure. Again, decisions must be made about the nature of the syntactic formalism which are not easy to revisit. This step is particularly expensive in time, talent, and treasure.
- For each predicate in the sentence, assign semantic roles to the appropriate spans. This can be done by making reference to nodes in the syntactic trees generated by the previous step, or we could simply provide span indices.

Each of these steps is required in order to have enough data to train a system that automatically predicts semantic roles on novel text. Moving through each of

these steps is extremely expensive – far beyond the reach of an individual researcher. As a result, resources like the Penn Treebank [Marcus et al., 1993] and the Proposition Bank [Palmer et al., 2005] are created at great expense and then used and re-used. These resources are of generally high quality and have seen the development of increasingly sophisticated language resources, but they still suffer from limitations. Performance of systems trained on the Penn Treebank on genres other than newswire is substantially lower [Pradhan et al., 2008]. Making changes to the grammar or tagset (say, to convert it to a different syntactic framework) requires extensive work [Hockenmaier and Steedman, 2007, Chen and Vijay-Shanker., 2004]. And if we wish to use our semantic role labeling software on another language, work on a new corpus for that language must begin from scratch. If we could eliminate just one of the annotation steps, we could greatly reduce the labor required for the development of a semantic role labeling system for a new genre or language.

Notice that the steps above require different levels of effort and expertise. Obtaining large quantities of raw text in a target genre or language is relatively straightforward – this requires no special training or attention besides avoiding duplicate text and copyright infringement. Part-of-speech tagging is nontrivial, but we can probably count on having at least a dictionary for our target language (which will provide us with the possible grammatical categories for each word). Although there is no consensus on the nature of semantic roles among linguistic theorists, once a set of Propbank-style roles have been defined it is not overly difficult for a layperson to identify them in a sentence (it requires little coaching for a typical undergraduate in an introductory linguistics class to identify the *the board* as the “entity being joined” in *Pierre Vinken will join the board as a nonexecutive director*).

The step of syntactic annotation, however, stands out as particularly difficult. Skilled syntactic annotators take months to train. They are required to make judgements on the nature of sentences that are sharply divorced from the kinds of judgements that humans are generally required to make (only someone with a background in linguistics will be able to identify the constituents in this sentence, but any literate adult can explain informally the semantic relationships between the nouns and their verbs). The precise conventions for the syntactic annotation of certain constructions must also be made known to the annotators, even if they are experienced linguists – phrases like *she helped students do better* could reasonably be analyzed in two different ways (figure 1.1) – the important thing is that they be annotated consistently. As the number of annotators grows (and for a large treebank, it most certainly will), maintaining consistency among annotators quickly becomes problematic, requiring, in the case of the Penn Treebank, an annotation manual 318 pages in length<sup>1</sup> [Bies et al., 1995]. In spite of the Herculean effort of enforcing accuracy and consistency in the Penn Treebank, identifying and correcting inconsistencies in treebanks offers plenty of research opportunities [Dickinson and Meurers, 2003, 2005, Dickinson, 2005, Boyd et al., 2007].

Errors and inconsistencies in Treebanks can be particularly devastating for semantic role labeling. Consider the annotation of the phrase *the company stopped using asbestos in 1956*. The Penn Treebank annotator provides what appears to be, at first glance, a perfectly reasonable analysis (figure 1.2). Upon closer inspection, however, we find that the temporal modifier should be under the high VP, rather than the low

<sup>1</sup>Compare this to the Propbank semantic role annotation manual [Babko-Malaya, 2005], which is 38 pages long.

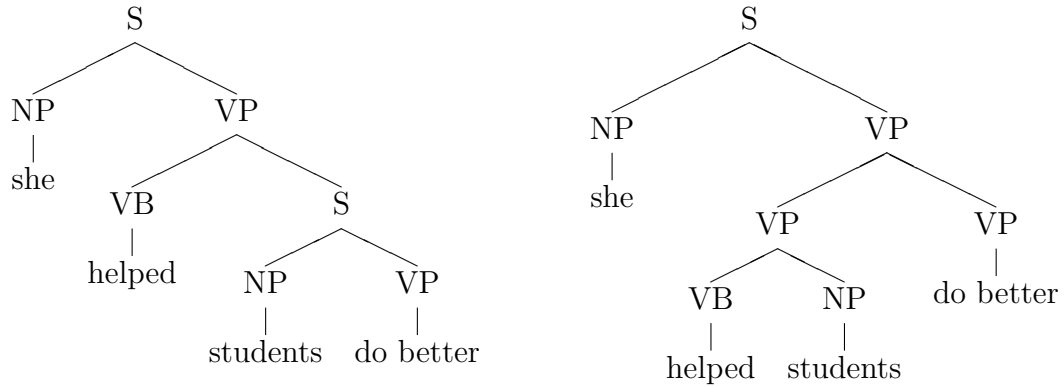


Figure 1.1: Two reasonable analyses for the sentence *she helped students do better*. For our purposes, it is less important that we choose one over the other as it is that we choose one consistently.

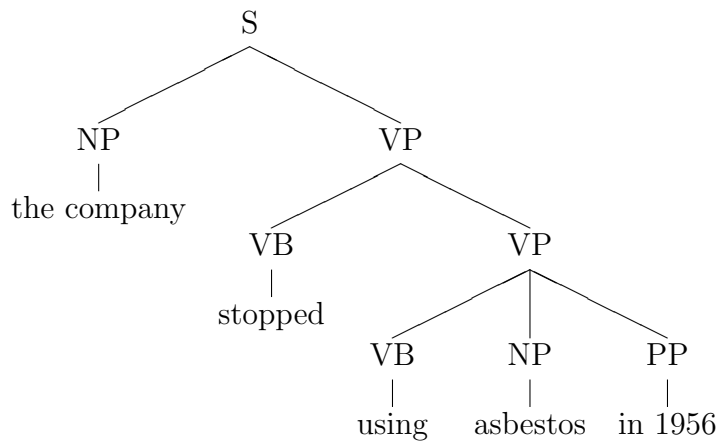


Figure 1.2: An erroneous analysis for the phrase *the company stopped using asbestos on 1956*. The bracketing suggests that the temporal modifier applies to *using*, rather than *stopped*.

one (*The company **did so in 1956***). Indeed, when provided as input to a semantic role labeling system, *in 1956* is identified as a temporal modifier of *using*, not *stopped*, leading to the somewhat nonsensical interpretation that the company made a habit of using a time machine to travel to 1956 for the purpose of using asbestos, but were eventually forced by the time-police to discontinue the practice. Surely this error by the annotator is forgivable – the prepositional phrase attachment distinction is difficult, and the annotators must make hundreds of such decisions rapidly. Surely the annotator grasps the correct meaning of the sentence, but the format of syntactic annotation is entirely unnatural. Indeed, the Propbank annotator for this sentence correctly identifies *in 1956* as the temporal modifier of *stopped* without difficulty. This mismatch of syntax and semantics almost ensures that an automatic SRL system could not possibly predict the correct role for *in 1956*.

Consider another case: the noun phrase *a group of workers exposed to asbestos*. This phrase is ambiguous – the two analyses are shown in figure 1.3. It is difficult to determine the analysis intended by the journalist – both structures make sense, and the semantic distinction is small. The Penn Treebank annotator chose the latter analysis, where *workers* is modified by *exposed to asbestos*. This would be fine, except that the Propbank annotator chose the equally acceptable former option, where *a group of workers* is modified by *exposed to asbestos*. The attachment decision will almost certainly determine the span (and headword) of the predicted role. Because of this mismatch, we are unlikely to identify the correct relationship between the syntax and the semantics in this case.

Clearly, generating a high-quality syntactic treebank is no small undertaking. Humans annotators are, by nature, notoriously inconsistent, and the difficulty in ensuring

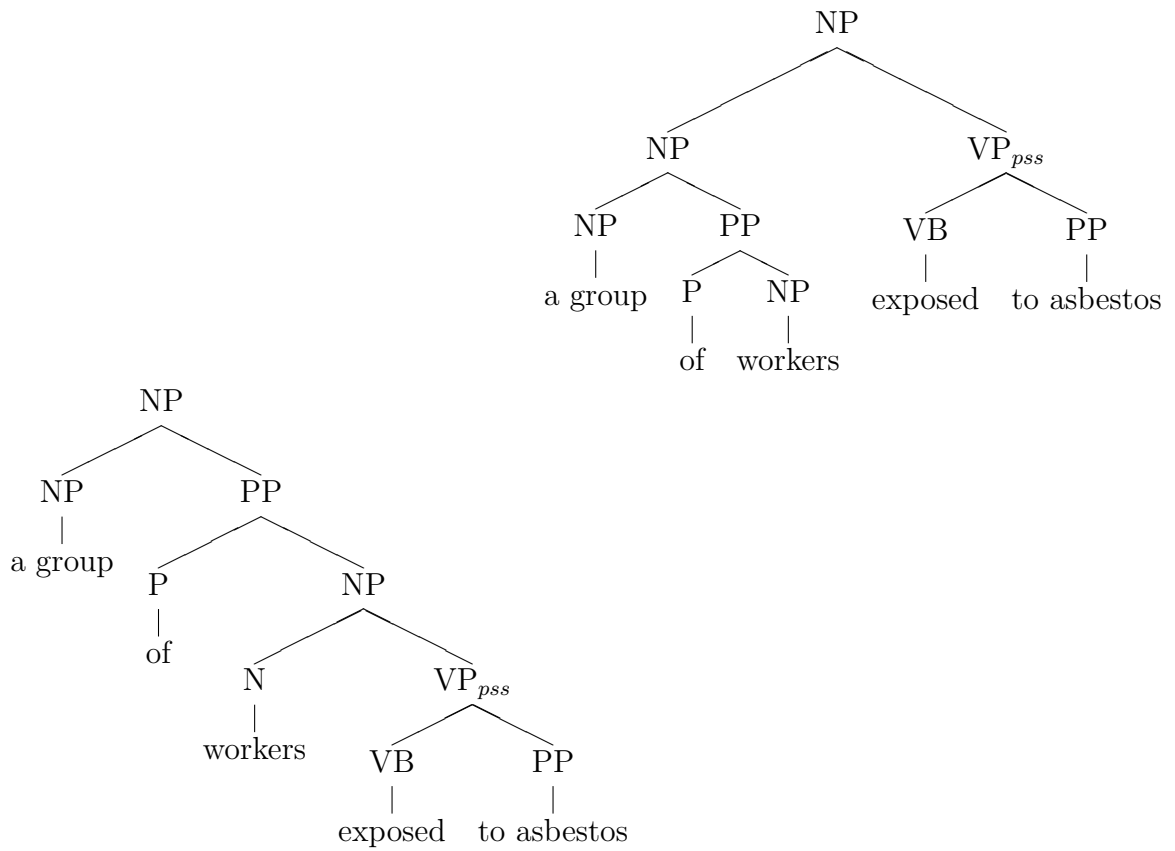


Figure 1.3: There are two possible analyses for *a group of workers exposed to asbestos*. The semantic difference between the two is very slight – the important thing is that the syntactic and semantic annotations be consistent.

consistent syntactic and semantic annotation is great. But what if we could somehow use a semantic, Propbank-style annotation to generate a syntactic, Penn Treebank-style annotation? The semantic annotation is far more natural for annotators, and, by using a Propbank-style numbered-role system, can remain more theory-neutral than any syntax annotation can. Granted, producing a corpus of semantic roles like Propbank is no small undertaking, but it is certainly less effort than producing *both* a syntactic and a semantic corpus.

In this dissertation, I will describe a method of using the English Propbank to generate, automatically, a CCGbank-like syntactic corpus, which can then be used to power a parser and semantic role labeler. I will begin in chapter 2 by describing Combinatory Categorical Grammar, my chosen syntax formalism. Next, in chapter 3, I will give a detailed description of semantic role labeling as a natural language processing task and describe our semantic role labeling system, Brutus [Boxwell et al., 2009]. I will then describe in chapter 4 some of the challenges involved in parsing with CCG, especially in the context of learning and predicting semantic roles in the context of a packed chart. In chapter 5, I will describe the problem of generating CCG lexical categories from scratch, and how the Propbank annotation can give valuable insight into how we choose our categories and syntactic structures, while highlighting the utility of a categorial formalism like CCG for such a purpose. Finally, in chapter 6, I will describe how to use the judgements of the Propbank annotators to guide the generation of a treebank that can accurately predict semantic roles on novel text. Chapter 7 will summarize the dissertation and look forward to future work.

## CHAPTER 2: Combinatory Categorical Grammar

### 2.1 Introduction to CCG

Combinatory Categorical Grammar [Steedman, 2000] is a grammar formalism that describes categories in terms of their ability to combine with other categories. Consider the category of words that includes *the*, *an*, and *my*. Rather than assigning this category an arbitrary name (like DETERMINER or ARTICLE), CCG assigns them the name NP/N, or “the category of words that will produce a noun phrase if combined with a noun to the right”. Similarly, the category that contains the words *likes*, *devours*, and *sees* is not named TRANSITIVE VERB – they belong to the category (S[dcl]\NP)/NP, or “the category of words that will produce a declarative sentence if combined with a noun phrase to the right and then a noun phrase to the left”. CCG categories are often referred to as CCG *supertags*, to reflect their greater informativeness than traditional part of speech tags. The process of automatically assigning CCG supertags to lexical items is called *supertagging* [Bangalore and Joshi, 1999]. Table 2.1 shows some common CCG categories and the words or phrases that they contain. Notice that there is no distinction between lexical categories and grammatical categories: words and phrases can share the same category if they have the same combinatoric preferences. Notice also that some words can belong to multiple categories: words like *with* and *for* can belong to the category (NP\NP)/NP when they modify noun phrases (*saw the man with brown hair*) or to the category ((S\NP)\(S\NP))/NP

Category	Part of Speech	Examples
NP/N	Determiner	<i>the, an, the mayor of Boston's</i>
N/N	Adjective	<i>red, happy, simple but effective</i>
S[dcl]\NP	Intransitive Verb	<i>sleeps, awakens, put it on the table</i>
(S[dcl]\NP)/NP	Transitive Verb	<i>likes, devours, gives the dog</i>
(NP\NP)/NP	Preposition	<i>with, for, in and around</i>
((S\NP)\(S\NP))/NP	Preposition	<i>with, for, in and around</i>

Table 2.1: Some common CCG categories and some words and phrases that they include.

when they modify verb phrases (*skipped class for three weeks*).

There are two kinds of CCG supertags: atomic and complex. Atomic CCG supertags (like S[dcl], NP, N, and PP) have no internal structure, and take no other categories as arguments. Complex CCG supertags (like S[dcl]\NP, NP/N, and PP/NP) consist of two CCG categories separated by a slash. In complex CCG supertags, the right category is the *argument* category and the left category is the *result* category. The result and argument categories can themselves be either atomic or complex. The direction of the slash of a complex category (/ or \) indicates the direction that the category combines with its arguments. Consider sentence *the man devoured a steak* (figure 2.1). In this sentence, the word *the* has the category NP/N, or “the category that will be a noun phrase if it could combine with a noun to the right”. There is a noun immediately to the right (*man*), so the two words combine and form an NP. The same logic follows for *a steak*. Next, the word *devoured* combines with *a steak*. Notice that the result category for *devoured* is itself a complex CCG category, but this makes no difference for the purpose of applying the outermost argument. *devoured a steak* is therefore of the category S[dcl]\NP, the same category of an intransitive verb

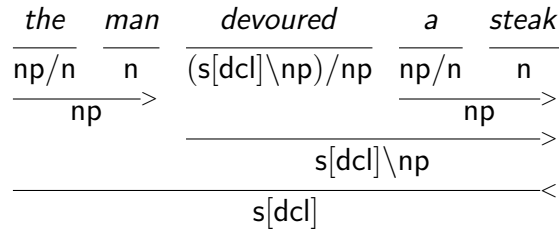


Figure 2.1: A simple CCG derivation.

standing alone. The  $S[dcl]\backslash NP$  category indicates that this category would become an  $S[dcl]$  if it could combine with a noun phrase to its left. It finds a noun phrase to its left (*the man*) and combines with it, yielding the category for declarative sentences,  $S[dcl]$ .

## 2.2 Function Application

For the sake of clarity, we have been somewhat informal in our description of how categories combine. To understand CCG properly, however, it is necessary to understand the connection between CCG and formal logic. Consider the following simple argument:

1. If it is raining, then I will get wet. (Premise)
2. It is raining. (Premise)
3. Therefore, I will get wet. (1+2, MP)

Step 3 of this simple argument can be reached by applying the simple argument form *modus ponens*, or MP. Modus ponens formalizes the familiar fact that the consequent of a conditional must be true if the antecedent is true. This is stated formally in figure 2.2.

$$\frac{A \supset B \quad A}{B} \text{MP}$$

Figure 2.2: The general case of modus ponens, a simple argument form

Modus ponens states that if A is true, and  $A \supset B$  is true, then B logically follows. We can define similar formal rules for CCG. The simplest argument forms of CCG are called *forward application* and *backward application*, and are abbreviated with  $>$  and  $<$ , respectively. These simple arguments (called *combinators* in CCG) are given in their general forms in figure 2.3, and are collectively called *function application*.

$$\text{a. } \frac{A/B \quad B}{A} > \quad \text{b. } \frac{B \quad A \setminus B}{A} <$$

Figure 2.3: The general cases of the two main CCG combinators, *forward application* (a) and *backward application* (b).

Function application works regardless of whether the arguments are atomic or complex. In some cases, a category must take a complex category as an argument.

Figure 2.4 shows two sentence analyses where CCG categories select for complex arguments.

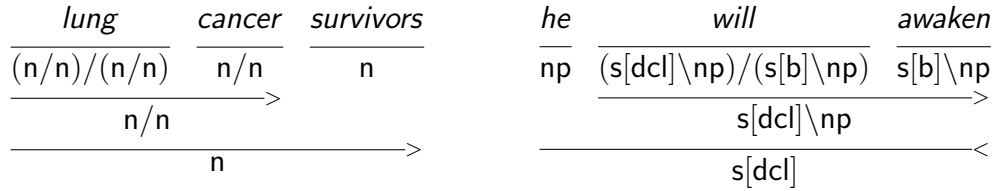


Figure 2.4: CCG categories often select for complex arguments..

### 2.3 Function Composition

Consider the following logical argument:

1. If it rains today, then I will stay home. (Premise)
2. If I stay home, then I will do the dishes. (Premise)
3. Therefore, if it rains today, then I will do the dishes. (1+2, HS)

The argument form used to arrive at step 3 is called a *hypothetical syllogism*. It can be used in logical proofs to chain together strings of conditionals. The formal definition of hypothetical syllogism is shown in figure 2.5.

$$\frac{A \supset B \quad B \supset C}{A \supset C} \text{HS}$$

Figure 2.5: The general case of a hypothetical syllogism, a simple argument form.

It is important to notice that introducing hypothetical syllogism to a system of logic may enable us to come to a particular conclusion in more than one way. Suppose that we are asked to prove the statement  $Z$  given the following premises:

(A)  $X$

(B)  $X \supset Y$

(C)  $Y \supset Z$

There are two ways to prove  $Z$ : we could apply modus ponens twice, or we could use hypothetical syllogism first and then apply modus ponens once (figure 2.6). Both are valid ways of proving the same thing.

$$\frac{\frac{X \quad X \supset Y \quad Y \supset Z}{Y} \text{MP}}{Z} \text{MP} \qquad \frac{X \quad \frac{X \supset Y \quad Y \supset Z}{X \supset Z} \text{HS}}{Z} \text{MP}$$

Figure 2.6: There is often more than one way to arrive at a particular conclusion.

Consider now a short phrase, like *settled in Columbus* (figure 2.7). The categories of the words in this phrase are uncontroversial, the traditional structure follows easily from them, and we reach the predictable conclusion that the entire phrase together is

$$\frac{\textit{settled}}{(s[dcl] \setminus np) / pp} \quad \frac{\textit{in}}{pp / np} \quad \frac{\textit{Columbus}}{np}$$

Figure 2.7: A simple sentence, with the appropriate CCG categories assigned to the lexical items.

of category  $S[dcl]\backslash NP$ . But there is another way to show that *settled in Columbus* has that category. Just as we can chain conditional statements together in traditional logic using a hypothetical syllogism, we can chain together complex categories in CCG. Think of the words *settled* and *in* as logical premises:

1. If there is an NP to the right of *in*, then *in* can make a PP.
2. If there is a PP to the right of *settled*, then *settled* can make a  $S[dcl]\backslash NP$ .
3. Therefore, if there is an NP to the right of *in*, then *settled* can make a  $S[dcl]\backslash NP$   
(1+2, HS)

Put a different way – suppose that we had to propose a CCG category for the phrase *settled in*. This requires some mental gymnastics, as this phrase does not correspond to a constituent in traditional phrase structure grammar (at least not in the non-particle sense we intend here), but careful thinking shows that it as a verb phrase that is missing a noun phrase on the end. This leads us to propose the category  $(S[dcl]\backslash NP)/NP$ , which is the category that would be a verb phrase if it could combine with a noun phrase to the right. The two ways of deriving the same category for *settled in Columbus* are shown in figure 2.8. The formal definitions of function composition are shown in figure 2.9.

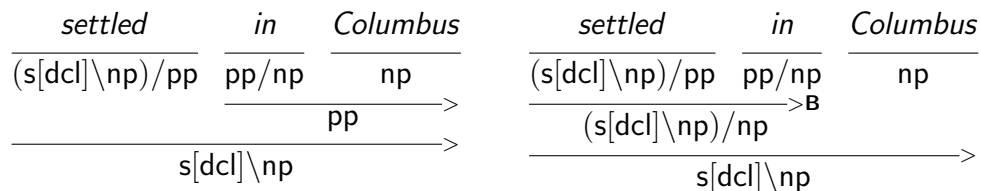


Figure 2.8: Two valid analyses of the phrase *settled in Columbus*.

$$\text{a. } \frac{A/B \quad B/C}{A/C} \rightarrow^{\mathbf{B}} \quad \text{b. } \frac{D \setminus C \quad E \setminus D}{E \setminus C} \leftarrow^{\mathbf{B}}$$

Figure 2.9: The general cases of the function composition CCG combinators, *forward composition* (a) and *backward composition* (b). Backward composition is very rare in English.

This capability of CCG introduces a new concept that is foreign to traditional phrase structure grammar: *spurious ambiguity*. Spurious ambiguity is when there are two possible analyses for a phrase resulting in equivalent spanning syntactic categories and equivalent semantics, like the two analyses in figure 2.8. This is not to be confused with *genuine ambiguity*, characterized by familiar examples like *he saw the man with a telescope*, where the two analyses have equivalent spanning syntactic categories but non-equivalent semantics. We'll have more to say about the importance of the distinction between genuine and spurious ambiguity later; for now it is enough to be aware of the phenomenon.

One of the reasons that function composition is important is that it enables us to combine certain strings of words for coordination. Consider the phrase *I want two sugar and three chocolate cookies*. Traditional phrase structure grammar struggles to account for cases like this. Perhaps we might group *two sugar* into a constituent, but we might hesitate to call this a noun phrase. Alternately, we might hypothesize an invisible element after *sugar*, or assume a multistratal approach where particular words are deleted in the surface structure, effectively pretending that the sentence really says *I want two sugar cookies and three chocolate cookies*, but this is problematic for practical computational purposes. CCG, however, has no problem with

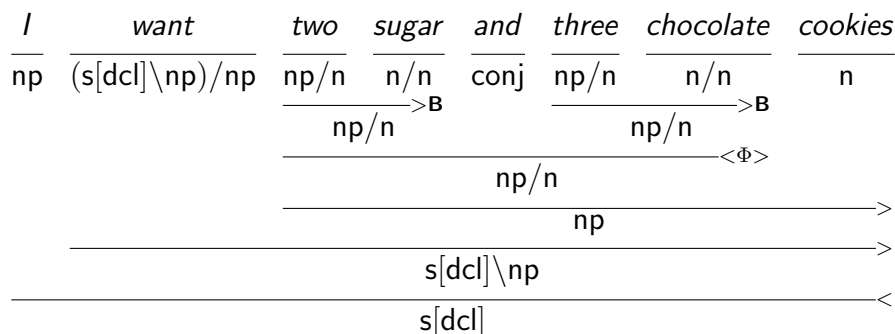


Figure 2.10: Function composition can be used to combine phrases so that they can be coordinated without hypothesizing invisible elements. The  $\Phi$  symbol indicates coordination.

this construction, as function composition can be used to assign a category to *two sugar* as well as *three chocolate* (they are both NP/N, the category that results in a noun phrase when combined with a noun to the right). These identical categories can then be coordinated in a familiar way, and then attached to *cookies*, just as if we had coordinated two regular determiners. (figure 2.10). We are unconcerned that *two sugar* and *three chocolate* are treated as units, as we are already comfortable with function composition generating categories for spans that do not correspond to traditional constituents.

## 2.4 Type Raising

Up until now, we have considered argument patterns and CCG combinators that are *binary* in nature – they require two premises or CCG categories as input. We now turn our attention to *unary* patterns, which are similar to the hypothetical proofs of traditional logic. Consider the following simple premise:

1. The congressman lied. (premise)

Taken alone, this premise will lead nowhere given the argument patterns already described. We can, however, make a series of assertions that rely only on step 1:

1. The congressman lied. (premise)
2. Suppose that “the congressman lied” implies that there was a scandal.  
(Hypothetical Assumption)
3. Then there was a scandal. (2+1,MP)
4. If “the congressman lied” implies that there was a scandal, then there was a scandal. (1, Hypothetical Proof)

Notice we are introducing no further assumptions. We are not saying that there was a scandal, or that “the congressman lied” implies that there was a scandal – we are simply saying that *if* the congressman lying implies that there was a scandal, then there was indeed a scandal. We know this because of the premise in step 1, which was given. No new premises are needed.

CCG allows a similar re-writing of categories. For example, suppose that we have the word *dogs* with the category NP. We could think of this category either as a simple NP, or we could think of it as a potential argument for another category. Suppose that the next word in the sentence is *sleep*, with the category  $S[dcl]\backslash NP$ . We could choose to think of *dogs* as “the kind of category that, if combined with  $S[dcl]\backslash NP$ , produces  $S[dcl]$ ”. Changing the CCG category in this way is called *type-raising*. The two equivalent approaches are shown in figure 2.11. The formal definitions of CCG type-raising are given in figure 2.12.

$$\begin{array}{cc}
\text{a.} & \frac{\frac{dogs}{np} \quad \frac{sleep}{s[dcl]\backslash np}}{s[dcl]} < & \text{b.} & \frac{\frac{dogs}{np} \quad \frac{sleep}{s[dcl]\backslash np}}{s[dcl]/(s[dcl]\backslash np)} >^T \\
& & & \frac{}{s[dcl]} >
\end{array}$$

Figure 2.11: Two equivalent analyses for *dogs sleep*. In the first analysis, the verb combines with the subject using backward application. In the second analysis, the subject is first type-raised over the verb, and then combines with the verb using forward application. Notice that the direction of function application is reversed.

$$\begin{array}{cc}
\text{a.} & \frac{A}{T/(T\backslash A)} >^T & \text{b.} & \frac{A}{T\backslash(T/A)} <^T
\end{array}$$

Figure 2.12: The general cases of CCG type-raising, called *forward type-raising* and *backwards type-raising*. Note that here T denotes a metavariable over all well-formed CCG categories.

One important thing to notice about type raising is that we can choose any statement we like to type-raise over, provided that the statement is a conditional based on our existing premise. Consider the following argument:

1. The congressman lied. (premise)
2. Suppose that “the congressman lied” implies that the moon is made of green cheese. (Hypothetical Assumption)
3. Then the moon is made of green cheese. (2+1,MP)
4. If “the congressman lied” implies that the moon is made of green cheese, the moon is made of green cheese. (1, Hypothetical Proof)

Forward Type Raising	Backward Type Raising
NP	NP
S/(S\NP)	S/(S/NP)
NP/(NP\NP)	NP/(NP/NP)
(S\NP)/((S\NP)\NP)	(S\NP)\((S\NP)/NP)
PP/(PP\NP)	PP/(PP/NP)

Table 2.2: A variety of legal type-raising operations applied to the CCG category NP. Some, like  $S/(S\NP)$ , are commonly used in English. Others, like  $PP/(PP\NP)$ , serve only to demonstrate that anything can be type-raised over, even if the result is unlikely to be useful.

Step 4 is a valid progression from step 1. Notice that we have not claimed that “the congressman lied” implies that the moon is made of green cheese – it probably doesn’t. We are simply claiming that if that were true, then the moon would be made of green cheese. We can choose anything as our hypothetical assumption in a hypothetical proof. In the same way, we can type-raise CCG categories over any well-formed CCG category we like, keeping in mind that the resultant category might not be useful in the derivation. Table 2.2 shows a variety of valid type-raising.

So far, type-raising does little to help us capture linguistic phenomena. Its real strength lies in combination with function composition. Consider the phrase *the books that John likes*. This sentence is problematic for traditional phrase-structure grammar, as the phrase *the books*, which serves as a semantic argument of *likes*, is in a nonstandard syntactic position. Different syntactic theories capture this relationship in different ways – transformational grammar hypothesizes that *books* started immediately after *likes*, then “moved” to the higher position, and HPSG proposes a SLASH feature that enables a verb to borrow an argument *ex-nihilo* so long as it can repay it

later in the derivation. CCG has a very elegant solution to the problem that involves applying type-raising and forward composition one after the other. The CCG analysis of *the books that John likes* is given in figure 2.13. The words *John* and *likes* are given a category  $S[dcl]/NP$  using type-raising and function composition, then that category can be selected for by the relativizer. The result category of the relativizer is a postnominal modifier, reflecting the tendency of relative clauses to attach to nouns.

## 2.5 CCG Syntactic Dependencies

It is often advantageous in tasks like semantic role labeling to have a method of describing word-word relationships. In a vanilla context-free grammar, this proves difficult – consider the relationship between the words *papers* and *reads* in four different phrases (figure 2.14). How can we generalize across these cases? In all four phrases, the semantic relationship between *read* and *papers* is the same, and syntacticians have generally assumed that the syntactic relationship is the same or related, although precisely how this is achieved varies, and few are computationally attractive. CCG’s approach is as follows: when a CCG category is assigned to a lexical item, it is given a *head*, which consists of the index of that lexical item and the word itself. For example, consider the sentence *Robin reads papers*. The word *papers*, which has the CCG category NP, would be given a head  $NP_{papers,2}$ . Next, each argument of that category (if any) are assigned an *unfilled dependency*, which consists of the index of that lexical item, the word itself, the CCG category of that lexical item, and which argument it is assigned to. For example, consider the word *reads*, which has the CCG category  $(S[dcl]\backslash NP)/NP$ . Both of the arguments are given unfilled dependencies as follows:  $(S[dcl]\backslash NP_{2,reads,(S[dcl]\backslash NP)/NP,1})/NP_{2,reads,(S[dcl]\backslash NP)/NP,2}$ . Notice that the two

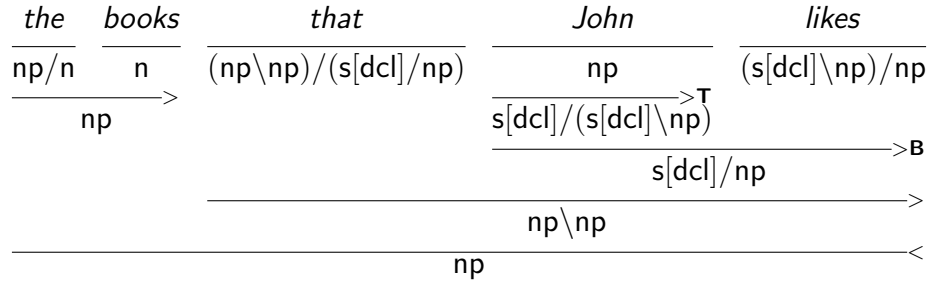


Figure 2.13: By using forward type-raising and forward composition in quick succession, we can assign the category  $S[dcl]/NP$  to *John likes*. This allows the relativizer *that* to select for the category  $S[dcl]/NP$ .

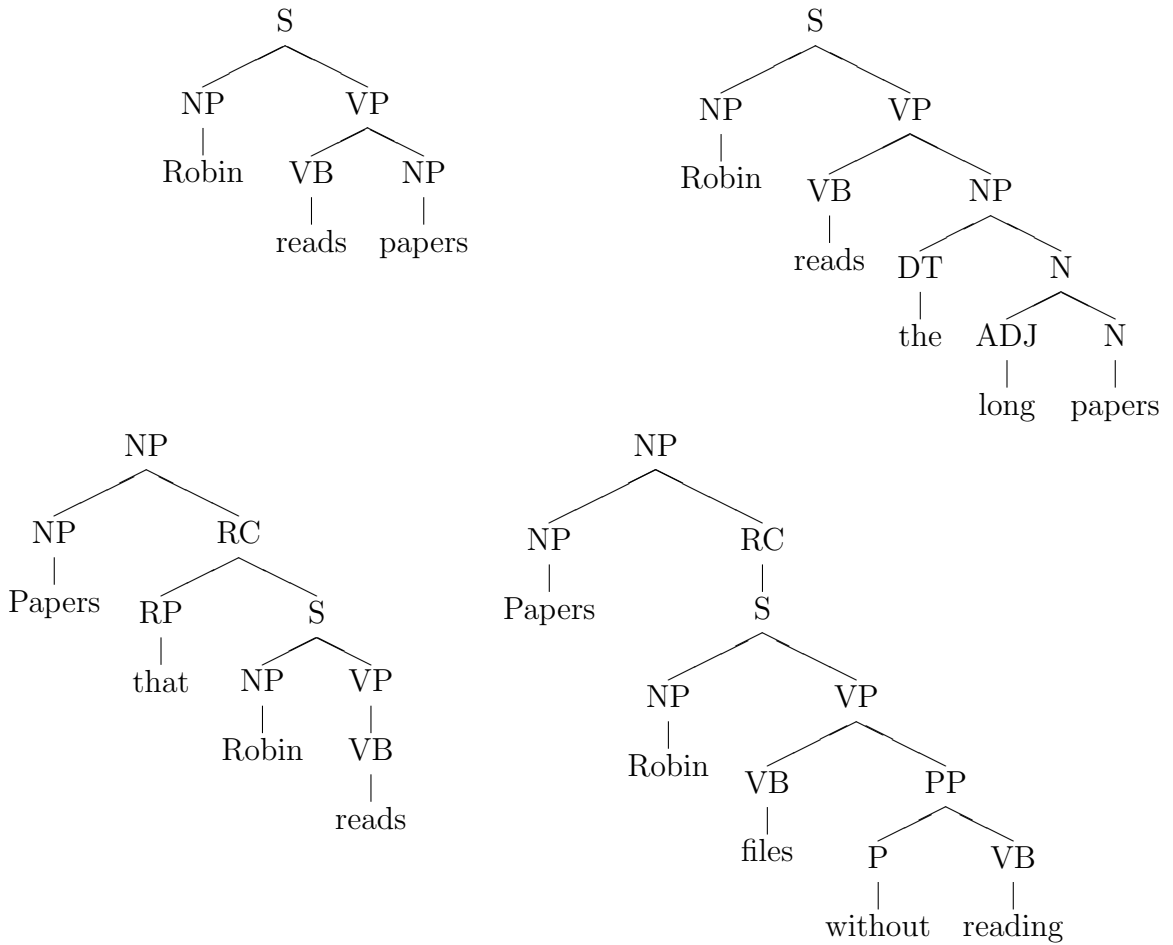


Figure 2.14: Four phrases in which the relationship between *read* and *papers* is the same. Using a context-free grammar, it is difficult to generalize these four cases.

unfilled dependencies are identical, except that the last number (the argument number) is different. The argument numbers are assigned to the syntactic arguments from left to right. The subject is argument number 1, because it is last to be discharged. The object is argument number 2, because it is the second-to-last to be discharged. If this were a ditransitive verb like *give*, there would be a third syntactic argument and its argument number would be 3. Notice that the order of the argument numbers is the opposite of the discharge order: the first argument to be discharged has the highest argument number, and the last argument to be discharged is argument number 1. These categories are shown in tree form in figure 2.15.

When a CCG category combines with another, the argument category is unified with the consumed category. That is to say, when  $reads:(S[dcl]\backslash NP)/NP$  combines with  $papers:NP$  (as in our example *Robin reads papers*), the outermost NP argument of *reads* unifies with the NP of *papers*. If the two categories fail to unify (that is, if their categories do not match), then the two categories cannot combine via forward application. But if they successfully unify, then the unfilled dependencies and heads of both categories mix. In this case, the first category contributes one unfilled dependency  $\langle 2, reads, (S[dcl]\backslash NP)/NP, 2 \rangle$  and the other category contributes one head  $(3, papers)$ . When an unfilled dependency and a head meet, they form a *filled dependency*. A filled dependency consists of the union of the information in an unfilled dependency and a head. In this case, the filled dependency consists of six fields:  $\langle 3, 2, (S[dcl]\backslash NP)/NP, 2, papers, reads \rangle$ . This can be read as: “the dependency to index 3 from index 2, which is the second argument of the category  $(S[dcl]\backslash NP)/NP$ , to the word *papers* from the word *reads*”. This is the format that can be found in the PARG dependency files distributed with the CCGbank [Hockenmaier and Steedman,

2005, 2007], and it is the format that we will assume here. The PARG dependency format is very specific – it links the identify of the dependency to the lexical category that created it. For this reason, there is a distinction between the subject dependency of *John ate* and *John ate the sandwich*, because the verbal categories are different ( $(S[dcl]\backslash NP)/NP$  and  $S[dcl]\backslash NP$ , respectively).

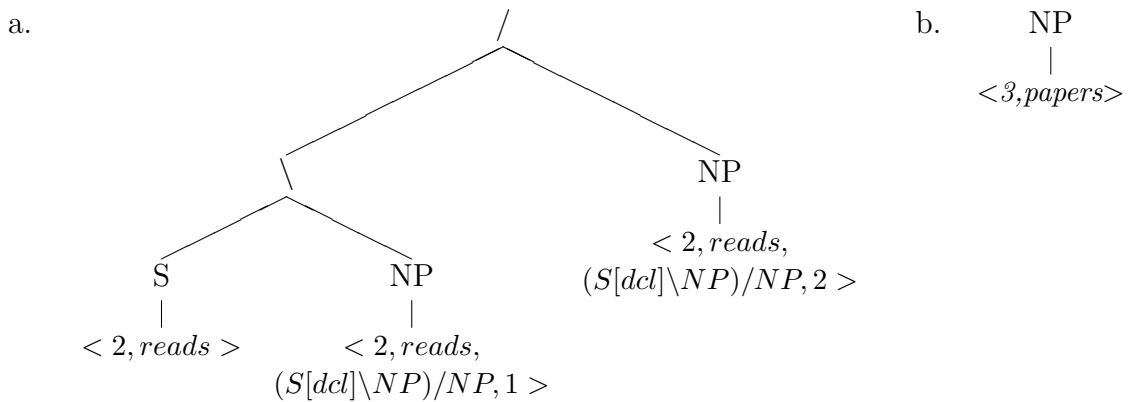


Figure 2.15: Two trees representing the CCG category  $(S[dcl]\backslash NP)/NP$  (a) and the CCG category  $NP$  (b). In the verbal category, the two  $NP$  arguments are assigned unfilled dependencies:  $\langle 2, reads, (S\backslash NP)/NP, 1 \rangle$  and  $\langle 2, reads, (S\backslash NP)/NP, 2 \rangle$ . The result category and, implicitly, all the nodes between it and the root node, are assigned a head:  $\langle 2, reads \rangle$ . In the nominal category, there are no arguments, so there are no unfilled dependencies. The single node is assigned the head representing that lexical entry:  $\langle 3, papers \rangle$ .

## 2.6 Threading Heads and Dependencies

Left as described in section 2.5, CCG heads and dependencies are only somewhat useful. Their real utility comes from their ability to be threaded through intermediary categories via coindexation to accurately account for adjunct categories and long-distance dependencies.

## 2.6.1 Adjunct Categories

A syntactic adjunct is a category that does not change the category that it combines with. For example, in English, NP-attached prepositional phrases are adjuncts.<sup>2</sup> Different syntactic theories account for this in a variety of ways, but this underlying observation is constant. CCG can easily handle syntactic adjuncts – categories of the form  $X/X$  or  $X\backslash X$  do not change the category of that which they combine with, as the result category is the same as the argument category. It is of the utmost importance, however, that we think clearly about the implications of our head-dependency rules. Consider the simple sentence *the big dog barked*. An erroneous derivation is shown in figure 2.16. There are a few problems to notice about this derivation and its accompanying dependency graph:

- The head of *big dog* is *big*, because *big* is the functor and its head  $\langle 1, big \rangle$  is passed along with the result category.
- Because *big* is the head of *big dog*, the dependency originating at *the* is erroneously attached to *big*.
- The spanning category, while correctly labeled as NP, has its head at *the*. This is unacceptable for our purposes, and must be addressed.<sup>3</sup>

The solution to this problem is to draw a distinction between lexical and functional heads. The lexical head of *big dog* should be *dog*, because this is the word that carries the primary semantic content, even if *the* is the functional head. We can

<sup>2</sup>Rare exceptions include event nominals, like *the destruction of the city*, where *of the city* is an argument [Honnibal et al., 2010].

<sup>3</sup>We are aware that many syntacticians assert the validity of determiner-headed noun phrases; we choose to follow the English CCGbank analysis.

use coindexation to pass along the heads of arguments. The category  $N/N$  is given a coindexation scheme like  $N^{10}/N^{10}$ , indicating that the head of the result category should come from the argument. The same is true of the category  $NP/N$ , where the head of the noun phrase should be the noun that the determiner consumes, not the determiner itself. A corrected version of the derivation is shown in figure 2.17.

## 2.6.2 Long-Distance Dependencies

In some cases, it is desirable to show the syntactic relationship between two words that are very distant from each other. Consider the following phrases:

- The dog ate the sandwich.
- The sandwich that the dog ate.
- The sandwich that Robin thinks the dog ate.

Intuitively, the relationship between *ate* and *sandwich* is the same in all three phrases. Consider the CCG derivation for the second phrase. By including an appropriate coindexation scheme on the relative pronoun, we can connect the object dependency of *ate* to *sandwich* in the same way we threaded dependencies through adjunct categories in section 2.6.1. The derivation is shown in figure 2.19.

## 2.7 Why CCG?

Throughout this dissertation, we will be using CCG as our syntax formalism for the following reasons:

- CCG, or a formalism like it, is required to provide the initial generated supertags in chapter 5. In that chapter we will describe an approach based on that of

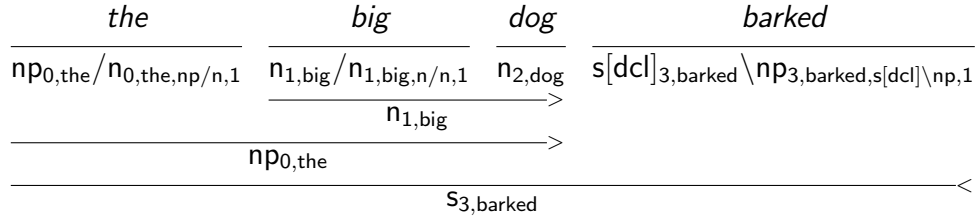


Figure 2.16: An derivation of *the big dog* predicting an erroneous dependency graph. Although the derivation correctly predicts that the span is a sentence, the heads and dependencies are problematic.

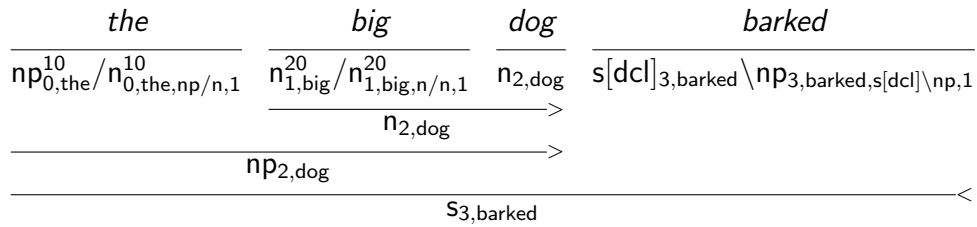


Figure 2.17: A corrected version of the derivation in figure 2.16. Notice that the superscript coindexation on the categories for *the* and *big* ensure that the correct heads are passed up to the result categories, and the noun phrase has the correct head *dog*.

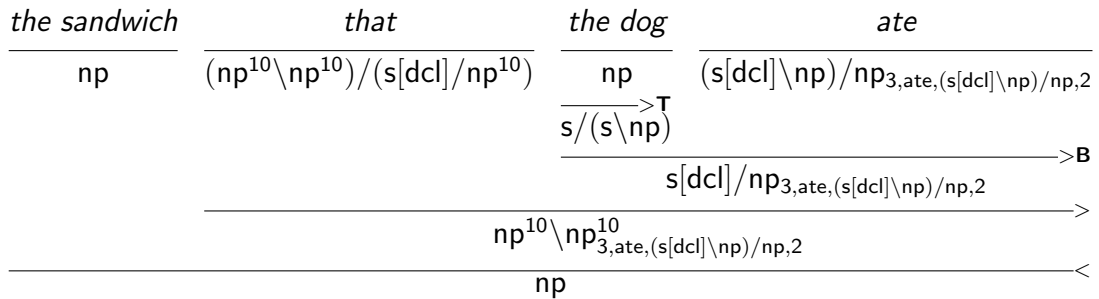


Figure 2.18: A derivation of the phrase *the sandwich that the dog ate*, demonstrating how the object dependency of *ate* can be threaded through *that* to reach *the sandwich*.

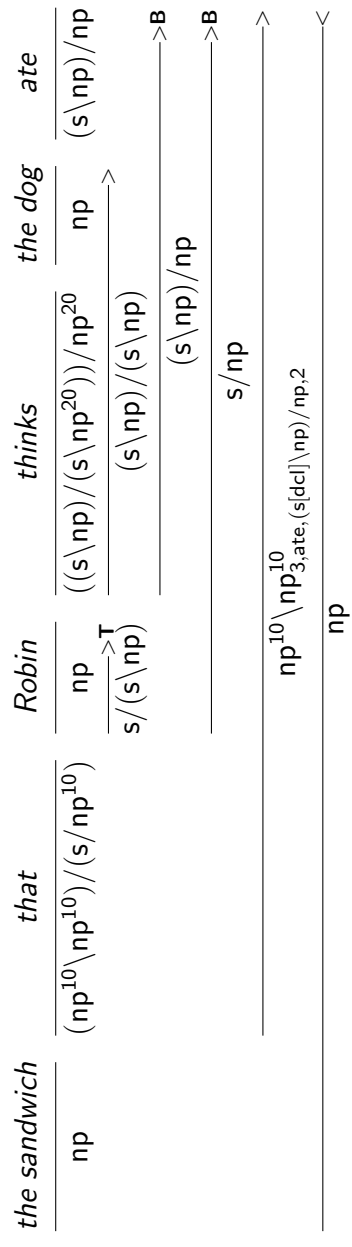


Figure 2.19: A derivation of the phrase *the sandwich that the dog ate*, demonstrating how the object dependency of *ate* can be threaded through *that* to reach *the sandwich*.

Baldrige [2008] and Ravi et al. [2010] to generate CCG supertags using only part-of-speech tagged text. A traditional Penn Treebank-style formalism cannot duplicate this process.

- CCG dependencies can be used as valuable semantic role labeling feature that are predictable inside a packed chart. This will allow us to experiment with predicting and training over semantic roles inside a packed chart in section 5.4.3. Only a formalism that predicts dependencies at parse time can do this – the Penn Treebank formalism must rely on the corresponding treepath feature (see section 3.4), which cannot be predicted inside a packed chart.
- There are many CCG experts at The Ohio State University, both graduate students and faculty, facilitating consultation and collaboration.

## 2.8 Conclusion

In this chapter, we described how Combinatory Categorical Grammar works and how it can be used to capture both local and long-distance syntactic dependencies. But so far we have not linked these syntactic dependencies to any kind of deeper meaning relationships. In the next chapter, we will explore how to link these grammatical relations into semantic ones.

## CHAPTER 3: Semantic Roles

### 3.1 Introduction

A *semantic role* represents a meaning relationship between a predicate and one of its arguments. Consider the following sentences:

- (a) Robin filed the papers.
- (b) The papers were filed by Robin.
- (c) These are the papers that Robin filed.
- (d) These are the papers that were filed by Robin.
- (e) These are the papers that Robin forgot about after filing.

In these sentences, we are particularly interested in the relationship between the predicate *file* and the argument *papers*. We can choose a name for this semantic relationship if we like (PATIENT, DOCUMENT, and THE-THING-THAT-GETS-FILED spring to mind) – if we name this relationship, we refer to it as a semantic role. But the key observation, whatever we decide to call this relationship, is that it remains the same in each of the sentences, even though the grammatical relation changes. In sentence (a), the role is in the object position. In the passive sentence (b), it's in the subject position. The semantic relation remains the same when the argument is made the

head of a relative clause (c,d), and even remains the same in a long-distance extraction in a parasitic-gap construction (e).

It is not overly difficult to define semantic roles for a handful of verbs. Once we attempt to describe semantic roles on a large scale, however, the problem quickly becomes unmanageable. Semanticists have long pursued an intuition that there must be a universal set of semantic roles from which every verb chooses a subset to apply to its arguments. The goal of producing an exhaustive list of universal semantic roles, however, proves elusive, and unfailingly generates skeptical looks from undergraduates in introductory linguistics classes. Dowty laments that “although many linguists seem to assume that linguistic theory should include a finite (and short) language-universal canon of thematic roles – including the familiar members Agent, Patient, Goal, Source, Theme, Experiencer, Instrumental, etc. – no one that I know of has ever attempted to propose a complete list” [Dowty, 1991, pg 548]. In practice, systems of nine or ten roles are often used, but this is clearly inadequate and based on an arbitrary granularity cutoff. For the purposes of computational linguists who are more interested in generating models than debating the minutiae of the PATIENT/THEME distinction, the problem of identifying a universal set of semantic roles must be either solved or avoided before work can proceed.

### **3.2 Semantic Role Paradigms**

There have been two major computationally-oriented approaches to resolve the issue of enumerating a universal set of semantic roles. The first is the Berkeley FrameNet Project [Baker et al., 1998]. This project relies heavily on frame semantics,

which states that the meaning of a predicate and its arguments depends crucially on knowledge of the particular real-world context in which it is used. This is to say that it is impossible to understand the meaning of the arguments of the words *buy*, *sell* or *lend* without some knowledge of how commerce works (imagine trying to explain these words to an individual who had lived their whole life in a money-free commune). Following the argument that it is impossible to divorce the meaning of verbal arguments from their real-world context, the Berkeley FrameNet Project divides the set of all predicates into small domains, or *frames*. For example, the TRAVEL frame includes semantic roles like MOVER, MEANS OF TRANSPORTATION, and PATH (sometimes called *frame elements*), and encompasses predicates like *journeyed*, *voyaged*, *took a trip*, and *commuted*. Clearly, there is a pattern in the kind of arguments that these verbs take – it makes sense to handle them together. Breaking down the space of predicates into conceptual chunks is a reasonable way to define sufficiently informative role labels while overcoming the problems of broad-scale semantics. The disadvantage of this approach is that it is, by nature, usable only within the constructed domains. Furthermore, automatically detecting which frame a novel piece of text should be in is problematic.

The second major approach to the problem of the lack of a universal set of semantic roles is to avoid making any sort of claim about connections between predicates whatsoever. The Propbank project [Palmer et al., 2005] makes no claim to developing an interconnected set of core semantic roles, even for a small domain – instead, it enumerates separate rolesets for each sense of each individual predicate. It even avoids assigning the familiar AGENT/PATIENT labels to the roles, opting instead to

assign predicate-specific role numbers (ARG0, ARG1, etc.), then elaborate on the individual meanings in prose in an accompanying “frame” file. For example, the frame file for the predicate *dance* consists of three numbered roles: Arg0 (*dancer*), Arg1 (*dance that is performed*), and Arg2 (*dance partner*). Even though for almost every predicate ARG0 corresponds to Dowty’s PROTO-AGENT, this is a pattern – not a rule. Propbank also includes a set of fourteen “modifier” roles, whose meaning is consistent across predicates (temporal modifiers, locative modifiers, etc) and which need not be identified on a predicate-by-predicate basis in the frame files. The meaning of the word “frame” in Propbank is similar to that of the Berkeley FrameNet Project, except that a Propbank frame encompasses only one sense of one predicate, instead of a small, multi-predicate domain. For this reason, Propbank only commits its annotation to be consistent across instances of that predicate (roles are consistent across all instances of *buy*, whether it appears in active, passive, or relative clauses, but there is no attempt to link these roles to those of *sell*). A semantic model may be able to identify an AGENT-like pattern across all instances of Arg0 or a THEME-like pattern across all instances of Arg1, but it would be difficult to draw strong connections between verbs when the numbered argument is 2 or higher. This illustrates the importance of predicate-specific features in our classifier feature set, so that the classifier can learn the specifics of each verb. Alternately, one could use the human annotated VerbNet [Schuler, 2005], which organizes English verbs by their syntactic and semantic selectional preferences. We chose not to incorporate VerbNet, as this introduces a reliance on additional human annotated resources that could interfere with efforts to reduce such reliance.

There is a related resource that focuses on annotating nouns in the same way that

Propbank does verbs. Nombank [Meyers et al., 2004] annotates nouns in the Wall Street Journal corpus that have explicit semantic arguments. Consider the phrase *a nonexecutive director*. With regard to the predicate *director*, Nombank will annotate *nonexecutive* as Arg3 (which the corresponding frame file describes as “rank”), and *director* itself as Arg0 (“job holder”). This could be used to improve our handling of noun-phrase structure, but because of the less transparent interface between CCG syntax and Nombank semantics we will set Nombank aside for now to focus on the more transparent structure of Propbank annotation.

### 3.3 Automatic Semantic Role Labeling

From here, we turn our attention to efforts to predict semantic roles on novel text automatically. Semantic Role Labeling (SRL) is related to parsing, but performs best when it is partially divorced from syntactic judgements. This is because there is not a simple one-to-one relationship between semantic roles and grammatical relations. Semantic roles are not trivially deduced from grammatical roles, but neither are they completely independent of grammatical relation. Consider the following unlikely sentences:

- (a) Robin put some bread on her butter.
- (b) This is the rabid dog that the frightened man bit.
- (c) The rabbit swallowed the wolf without chewing.

In each of these sentences, the meaning is clear, if unlikely. Even though any reasonable semantic model would predict that a rabid dog is more likely to bite a frightened man than *vice versa*, the grammatical relations in (b) make the intended

meaning clear. A model that is entirely ignorant of grammatical relations (say, a bag-of-words model) may correctly identify easily guessable relationships (*the champion won the race*), but we would expect it to perform no better than chance on sentences like *United Airlines acquired Continental*. Clearly there is some relation between semantic roles and grammatical relations, but this relationship is somewhat obscure and needs to be learned in order to accurately predict semantic roles in novel text.

### 3.4 Previous Approaches to Semantic Role Labeling

Early attempts at semantic role labeling were restricted to sharply limited domains, such as the Air Traveler Information System (ATIS) [Hemphill et al., 1990]. Early versions of these systems were largely limited to linking semantic rolesets to specific verbal subcategorization lists, trying to fit utterances into pre-arranged templates (I want a flight to TO\_CITY from FROM\_CITY on FLIGHT\_DATE). One of the first serious efforts at full scale semantic role labeling was Gildea and Jurafsky [2002], which used a statistical system to assign FrameNet roles to full CFG-parses generated automatically from the British National Corpus. This system used a variety of features to predict semantic roles on nodes in the trees. The features used in this system include:

- **Phrase Type.** The syntactic category corresponding to the targeted span of words – e.g. NP, PP, S.
- **Governing Category.** The syntactic category of the parent node of the targeted span of words.

- **Tree Path.** The list of categories along the path from the target node to the predicate, separated by arrows indicating up and down movement through the tree (figure 3.1). This is a powerful feature for representing the grammatical relation between the predicate and its argument – by including it as a feature for the statistical labeler, we can learn the patterns in the relationship between grammatical and semantic relations without overcommitting ourselves through a heuristic-based method. A weakness of this feature is that it tends to suffer from data sparsity.
- **Position.** A binary indicator feature showing whether the target word comes before or after the predicate. This is helpful for recovering from automatic parse errors, as it does not depend on the hypothesized structure.
- **Voice.** A binary indicator feature showing whether the predicate in question is in the passive or active voice. This is determined by simple heuristics based on the presence or absence of the copula or *get*.
- **Head word.** The lexical head of the target phrase. If the parser does not provide head words, Collins’ head-finding rules can be used [Collins, 1999, Appendix A].

To identify and label semantic roles, Gildea and Jurafsky estimate a probability distribution indicating how likely each constituent is to fill each role. Due to the familiar problems of data sparsity, it is necessary to organize the probability estimates so that if a particular pattern is unseen the system backs off to a more general estimate (e.g. if the headword, phrase type, and target word are, together, unseen, the system will back off to an estimate using only the headword and target word. If this

is still unavailable, it backs off further to the headword). A detailed description of this complex system of backoff probabilities is given in Gildea and Jurafsky [2002, section 4.2]. This model produces acceptable results, and shortly thereafter Gildea and Palmer [2002] demonstrate that the same architecture performs well in predicting Propbank roles instead of FrameNet roles, while simultaneously experimenting with chunking in place of full parsing. In both cases, the syntactic framework is largely based on that of the Penn Treebank.

To address the problem of the somewhat weak syntax formalism, Gildea and Hockenmaier [2003] adapt the system to work with CCG derivations. The somewhat sparse tree path feature is replaced with a CCG dependency-based feature:

- **CCG Dependency.** The concatenation of the CCG category of the predicate, the argument slot that the target word fills, and a binary indicator feature showing whether it is attached as an argument or an adjunct, shown as an arrow (figure 3.2).

The CCG dependency feature is a particularly powerful one, as a single feature can cover a wide variety of syntactic contexts that predict the same semantic role. Consider the phrases in figure 3.3. In each of these phrases, the semantic relationship between *girl* and *reads* is the same. If we were to calculate the tree path feature between this word pair in the three sentences, we would get three different results. The CCG syntactic dependency, however, remains the same – CCG recognizes that in each case *girl* serves essentially the same grammatical function (consider the person and number agreement between *girl* and *reads*), and grammatical function is a strong predictor of semantic role. This makes gives CCG (and, indeed, any formalism with a sophisticated system of handling syntactic dependencies) a powerful tool for semantic

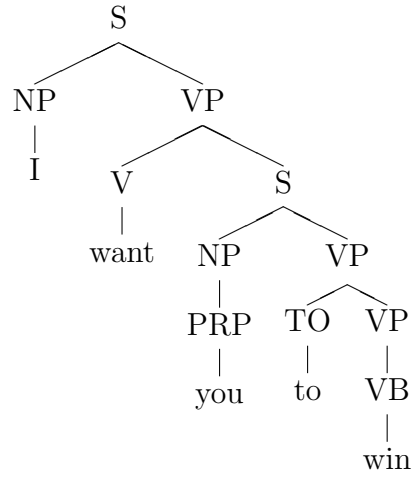


Figure 3.1: The tree path feature between *win* and *you* is  $VB\uparrow VP\uparrow VP\uparrow S\downarrow NP\downarrow PRP$

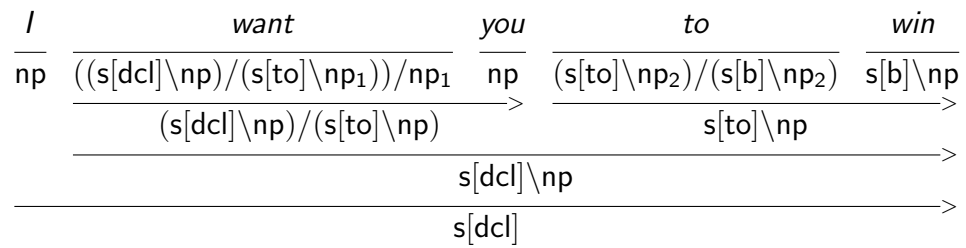


Figure 3.2: The CCG dependency feature between *win* and *you* is  $s[b]\backslash np.1.\rightarrow$

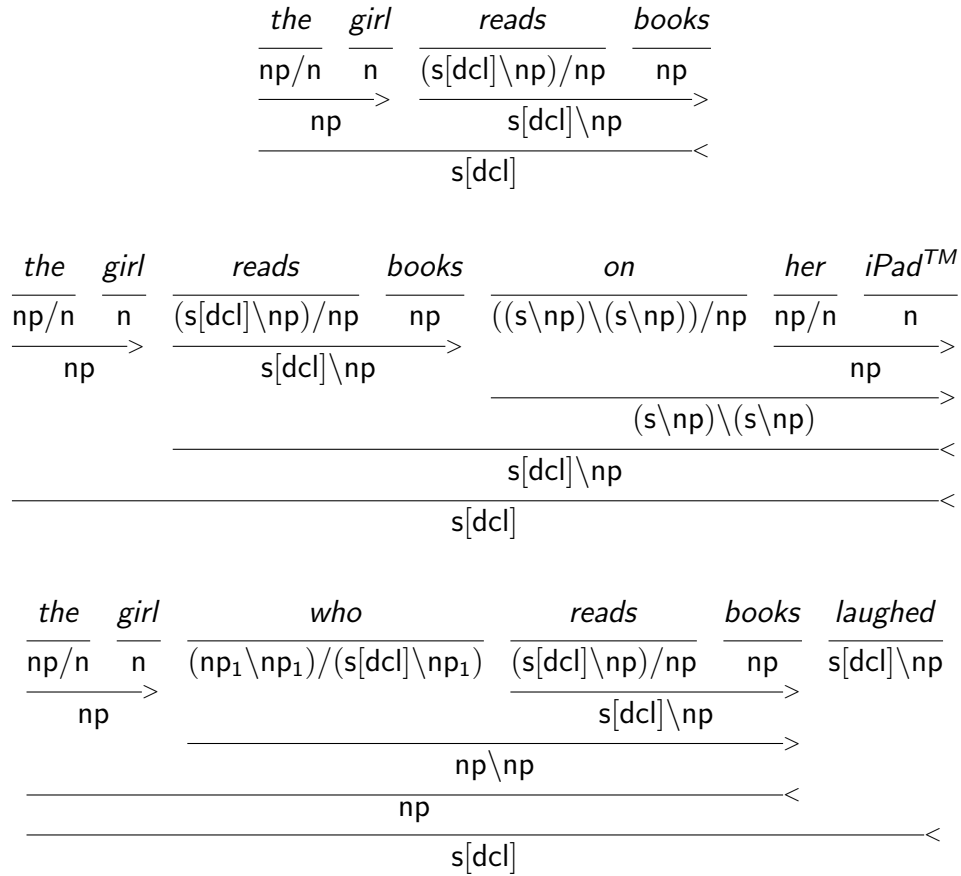


Figure 3.3: The CCG dependency feature between *reads* and *girl* is  $(s[dcl]\backslash np)/np.1.\rightarrow$  in all three sentences. This is convenient, because the semantic relationship is the same as well.

role labeling. One weakness of this feature is that the feature includes the complete CCG category of the verb, which means that subject dependencies will not be identical if the verbal categories are not identical (the subject dependency for an intransitive verb is not the same as the subject dependency for a transitive verb). This could lead to data sparsity issues for rare verb categories.

More recent work has seen numerous breakthroughs in semantic role labeling. One of the most prominent efforts has been that of Punyakanok et al. [2008]. This effort

follows previous approaches in many ways – the system is organized in a three step pipeline, passing information from parser to role identifier to role classifier. The identification step is constrained by certain linguistically-informed heuristics [Punyakanok et al., 2008, section 3.2], including the following:

- Arguments cannot overlap with the predicate.
- If a predicate is outside a clause, its arguments cannot be embedded in that clause.
- Arguments cannot exclusively overlap with the clauses.

In addition to the syntactic features introduced by Gildea and Jurafsky [2002], Punyakanok et al. [2008] also incorporate a variety of additional features that are largely chosen to protect the labeler from parser errors. These features include:

- **Subcategorization.** A complete list of categories that make up the VP in the tree. For example, the subcategorization feature for the tree in figure 3.1 is  $VP \rightarrow V-S$ .
- **Context Words.** A 5-word window of words before and after the target constituent, and their POS tags.
- **Verb Class.** The VerbNet class for the predicate. If there is more than one possible VerbNet class, then all possible classes are included.
- **Lengths.** The number of words covered by the target span.
- **Chunks.** A variety of features detailing whether a constituent is or is part of an larger constituent, as determined by a simple chunk parser.

One of the major contributions to SRL represented in Punyakanok et al. [2008] is the *pruning* stage. Instead of following Gildea and Jurafsky [2002], where every constituent is considered as a possible argument, the system uses heuristics to restrict role placement. Only sister categories of the verb or the VP are considered as possible arguments – this simultaneously filters out highly unlikely arguments and primes the system for the second major contribution.

The second major contribution of Punyakanok et al. [2008] is the use of inference to make judgements about entire sets of semantic roles, rather than judging each role independently. This step of global inference is represented as an integer linear program [Roth and Yih, 2004], where the integer program explores the solution space of possible argument labeling according to certain broad constraints. This is particularly useful for recovering from errors in the verbal category. In addition to the three heuristics applied to the identifier step, five more constraints are incorporated:

- Arguments for a particular predicate may not overlap each other or be embedded in each other.
- Each sentence shall have no more than one of each semantic argument<sup>4</sup>.
- If there is an R-*arg* argument, then there must be an *arg* argument. R-*arg* arguments are labeled on the relativizer in unreduced relative clauses, so there must be a referenced argument in the sentence.
- If there is a C-*arg* argument, then there must be an *arg* argument. C-*arg* arguments are used in discontinuous phrases, like *one troubling aspect of the*

<sup>4</sup>the authors acknowledge that natural language sometimes violates this rule, most notably in argument cluster coordination (*I left my pearls to my daughter and my gold to my son*), but because these constructions are rare they treat the constraint as hard.

*DEC's results, analysts said, was its performance in Europe*, where ARG1 of *said* is split into two parts. Furthermore, C-*arg* must occur after *arg*, according to the Propbank annotation guidelines.

- Predicates shall only be assigned valid numbered arguments, according to the possible labels enumerated in the frames files. That is, if a certain predicate has only ARG0 and ARG1 defined, it is not permitted to assign it ARG2.

This inference stage is supplied with candidate semantic roles by a pair of automatic parsers: Collins' parser [Collins, 2003] and Charniak's parser [Charniak, 2001]. Recall that the pruning step eliminates all constituents that are not sisters of the predicate – if two (or more) parses are used, the union of the sets of qualifying constituents from both parses can be tossed into the SRL system, and the constraints of the integer linear program can sort out their interactions (figure 3.4). The authors show a substantial improvement right away by incorporating roles predicted by both SRL analyses of parsers – the theoretical maximum SRL recalls of Collins' parser and Charniak's parser are 81.05% and 86.08% respectively, but the union of their roles achieves a theoretical maximum of 91.37% [Punyakanok et al., 2008, section 5.2]. In practice, SRL performance of Charniak's parser outperforms that of Collins' parser, but performing joint inference using both parsers causes the F-score to jump 2.5% over that of Charniak's parser alone. Incorporating n-best output from Charniak's parser yields even better results. This demonstrates the power of using multiple parsers to overcome parse errors and generate superior semantic role assignments.

One of the observations made by Punyakanok et al. is that Collins' parser and Charniak's parser give “noticably different outputs” [Punyakanok et al., 2008, section 5.2]. Because the SRL system is restricted to assigning roles to constituents in the

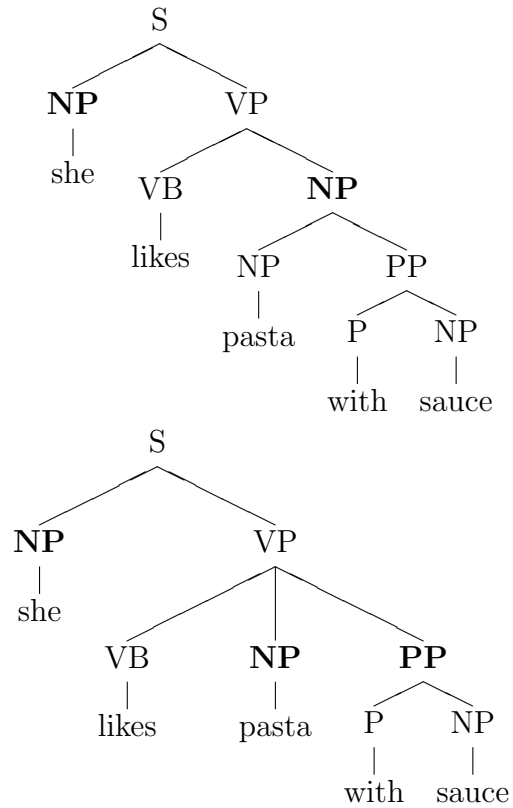


Figure 3.4: A side effect of tagging multiple candidate parses is that the semantic roles sometimes interfere with each other. For example, a system presented with these two parses will likely predict a role for *pasta with sauce* (based on the first parse) and *pasta* (based on the second parse). These role assignments are mutually exclusive – they cannot both be correct. Punyakanok et al. [2008] uses Integer Linear Programming to resolve the conflicts.

parse provided, a simple PP-attachment error could easily prove fatal to correct role placement. It seems plausible, therefore, that the labeler would benefit from input from a variety of different parses, especially if these parsers had widely divergent methods of ranking parses, or if the n-best list represented differences in prepositional phrase attachment (which is likely to have an effect on SRL), rather than internal NP-structure (which is not).

One of the properties of the SRL assignments given by this architecture is that they need not be consistent with any one of the input trees. Suppose that the system is given two parses (one by Charniak’s parser, another by Collins’). Each of the parses is erroneous in a different way. The system can, in theory, overcome this by selecting Charniak’s analysis for one role assignment and Collins’ analysis for another. This may be problematic depending on what the next stage in the NLP pipeline is, but it is simultaneously appealing to those who view syntactic parsing as little more than a means to an end (where semantic role labeling is that end).

On the topic of unsupervised approaches to semantic role labeling, Lang and Lapata [2010, 2011] show success at clustering semantic roles without the benefit of semantic training data. Their approach works on the idea that certain verbal arguments can be clustered based on semantic similarity with respect to the predicate, and that this can be improved by automatically detecting cases of non-standard word order. These experiments are primarily concerned with role classification (as opposed to role identification), and therefore use the human annotated Propbank to identify roles. This is the opposite of our approach, which uses semantic training data to infer syntactic structure.

Fürstenaу and Lapata [2009] use a different approach to induce new training data

for semantic role labeling. Using the Framenet corpus of semantic roles, they use a graph alignment method to identify instances of similar verbs to those that are already annotated, then use these to reduce error caused by otherwise unknown lexical items. Experimental results show that this improves performance on unknown lexical items. Unfortunately, it relies on having a manually labeled seed corpus with which to expand into the large, unlabeled expansion corpus. This too is insufficient for our purposes – we would like to do without syntactic training data entirely.

### **3.5 The Brutus Semantic Role Labeler**

The semantic role labeling system used in the remainder of this dissertation will be the Brutus CCG Semantic Role Labeler, adapted from the multi-formalism approach of Boxwell et al. [2009]. It takes complete CCG derivations as input, either from the gold-standard CCGbank Hockenmaier and Steedman [2007] or from an automatic CCG parser. We use a version of Propbank that has been adapted for use with the CCGbank [Boxwell and White, 2008].

#### **3.5.1 Predicate Labeling**

The first step in semantic role labeling is predicate labeling. Many of the reported results on semantic role labeling assume access to gold-standard predicates in order to better evaluate the semantic role labeling task independent of other factors. In a real-world application, however, these predicates must be predicted. We include a description here of our predicate identification system.

Predicate Labeling is divided into three steps:

- Identifying which words are predicates

- Stemming the word into its base form (e.g. the base form of “continued” is “continue”)
- Classifying the word into a specific sense (e.g. continue.01 vs continue.02, etc)

We will address each of these steps in turn.

## Predicate Identification

Predicate identification is accomplished using a maximum entropy classifier. Only verbs are considered by the classifier;<sup>5</sup> non-verbs are heuristically rejected as predicate candidates. The maximum entropy classifier is trained not to identify instances of the copula (*are*, *is*, etc) and certain auxiliary verbs (like *did* and *has*). The features used to train this classifier are as follows:

- **Word.** The candidate word, as it appears in the input. Many aspects of SRL systems use windows of words around the target word, but it was determined empirically on the development set that a window size of 1 (the word only) gave the best results.
- **Part-of-speech.** The Penn-Treebank-style tag assigned to the target word. This feature could have the following values: VBD, VBG, VBN, VBP, VBZ, and VB. For a complete description of the distinctions between these tags, see [Marcus et al., 1993].
- **CCG Category.** The CCG category assigned to the target word. These can come from the gold-standard CCGbank, or could be predicted.

<sup>5</sup>A verb, in this case, is defined as a word whose gold-standard part-of-speech tag begins with VB

- **Word endings.** A set of three features, indicating the last letter of the target word, the last two letters of the target word, and the last three letters of the target word. The purpose of these features is to recover from out-of-vocabulary errors with verbs that do not appear in Wall Street Journal text from the nineties. Consider the sentence *the reporter was blogging all month*. Even if the word *blogging* is unattested in the training data, the feature `LAST_3=ING` is strongly indicative of a progressive verb or a gerund, which are strong candidates for predicates. Including this class of features improves performance on the development set.
- **Category-Word.** A feature formed by concatenating the CCG Category feature with the word feature. This prevents certain CCG categories used by the copula from unduly influencing the classifier away from valid predicates. The copula is tagged as a verb, but is not considered a predicate for Propbank’s purposes

The maximum entropy classifier is trained to 300 iterations, reaching 98.766% accuracy on the training data. Performance on development data is given in table 3.1.

### Predicate Stemming

After a word has been identified as a predicate, we need to determine the base form of the verb. This is accomplished as follows:

- First, perform a dictionary lookup on the wordforms found in the training data. If this particular wordform has been seen in training, then its predicate form is listed in propbank. For example, the wordform *publishing* is attested in the training set (wsj\_0349.15), and it is paired with the predicate *publish*. Therefore,

when we see the wordform *publishing* in the development or test data, we can safely predict *publish* as the predicate.

- If the wordform is not found in the training data, then heuristics are applied to create a short list of candidate base forms:
  - If the wordform has the suffix “-ing”, then remove it.
  - If the wordform has the suffix “-ing”, then remove it and add “e”.
  - If the wordform has the suffix “-ied”, then remove it and add “y”.
  - If the wordform has the suffix “-ies”, then remove it and add “y”.
  - If the wordform has the suffix “-ed”, then remove it.
  - If the wordform has the suffix “-ed”, then remove it and add “e”.
  - If the wordform has the suffix “-s”, then remove it.
- In addition to the candidates generated by the heuristics above, also consider the strings generated by taking the candidate base-forms and re-inflecting them. For example, if our generated base form is *faint* (which is unattested in the training set), we will generate additional candidate words *fainting*, *fainted*, and *faints*, and add these to our candidate list as well.
- Check each candidate to see if it is attested in our training data, either as a wordform or a stem. If so, then use that predicate. If none of the candidates fit the training data, then one of the candidates is chosen arbitrarily.
- If the candidate list is empty (that is, if it has no relevant morphology to break down, and adding verbal morphology yields nothing), then the wordform itself is

predicted as the predicate. For example, the word “recede” is unattested in the training set, but appears in the development set. Guessing that the predicate is the same as the wordform produces the correct result.

The predictions of these heuristics are very accurate, predicting incorrect stems for only 21 of the 4615 predicates in the development set. Although stemming is an entire sub-field of natural language processing, and much effort has been devoted to sophisticated approaches to it, for our purposes a simple heuristic-based approach is sufficient, especially when working within the restricted domain of verbs in newswire. This does, however, remain an area with potential for improvement, especially if the system were to be adapted to work with a broader domain of text where the brute-force method of memorizing a stem dictionary is less likely to be successful. In order to focus on the broader issues of semantic role labeling, however, we choose to set aside the issue of stemming for now, perhaps to return to it at a later date.

### **Predicate Classification**

Once a word has been identified as a predicate and its stem has been found, we must determine the sense of the verb. Propbank divides each verb into one or more senses. Often, there are many complex senses of verbs that must be disambiguated. Consider the case of the predicate *shoot*, which, according to Propbank, has seven senses:

- **shoot.01:** “propel projectile”: *“the hurricane downed electric and telephone lines, **shot** coconuts through cottage rooftops, shattered windows and uprooted thousands of lives.”*

- **shoot.02:** “kill with gun”: “*one agent was stabbed and another was **shot** and killed.*”
- **shoot.03:** “record on film”: “*Mr. Lane **shot** ‘A Place in Time,’ a 36 minute black-and-white film.*”
- **shoot.04:** “shoot off”: “*John is forever **shooting** off his mouth.*”
- **shoot.05:** “shoot down”: “*Ireland didn’t spurn the Soviets after they **shot** down a Korean Air Lines jetliner over the sea of Japan in 1983.*”
- **shoot.06:** “move rapidly”: “*The Dow Industrials **shot** up 23 points in the opening hour.*”
- **shoot.07:** “shoot back”: “*And when he’s told ‘Try a little tenderness’, he **shoots** back ‘I’m going to try a little linguine’.*”<sup>6</sup>

The problem of word-sense disambiguation is particularly difficult – systems often struggle to beat the “most frequent sense” baseline. The annotator of the Propbank *shoot* file even includes a comment that he or she would not have thought that the word *shoot* would “be so complicated” and that “with all the straight transitive uses”, he or she feels “sorry for any automatic disambiguator. Yeesh”.

Our approach to the verb sense disambiguation problem is to train a separate classifier for each ambiguous verb. For example, the verb *abandon* has two attested senses in the training set, so we will train a classifier to distinguish between the two. The verb *abate*, however, has only one attested sense in the training set, so there is no need to train a classifier – we will classify every instance of *abate* that we find in the

<sup>6</sup>wsj-1397.28

development or test data as sense #1.

The features used in the suite of predicate classifiers are familiar:

- **5 word window.** The candidate word and the four words surrounding it. For example, if we were considering the predicate *shoot* in *The Dow Industrials shot up 23 points in the opening hour*, then the features would be WORD\_-2=DOW, WORD\_-1=INDUSTRIALS, WORD\_0=SHOT, WORD\_1=UP, WORD\_2=23. The 5-word window size was chosen empirically on the development set.
- **Part-of-speech.** The Penn-Treebank-style tag assigned to the target word.
- **CCG Category.** The CCG category assigned to the target word. These can come from the gold-standard CCGbank, or could be predicted.

Training this many individual classifiers (one for every ambiguous verb) introduces some implementation obstacles – storing all these models in memory simultaneously is beyond the reach of some desktop systems. For this reason, the predicate labeling system only loads a few models at a time, performing multiple passes over the data. For example, the first pass labels *abandon* through *condition*, the second pass labels *confer* through *happen*, and so on. This has no effect on the accuracy of the system, but is an important detail to be aware of when working with the accompanying software or reproducing the results.

## Predicate Labeling Performance

Performance on the predicate labeling task alone is shown in table 3.1. We show the performance of the full system on predicate identification (where full credit is given for identifying the presence of a predicate), predicate stemming (where full credit is

	ID	ID+Stem	ID+Stem+CL
Precision	98.8%	98.4%	92.4%
Recall	99.1%	98.7%	92.7%
F-score	99.0%	98.5%	92.5%

Table 3.1: Performance of the predicate labeler on the development set (WSJ 00). The leftmost column shows the accuracy of identifying words as predicates. The center column shows the performance of identification and stemming (ignoring verbal disambiguation). The rightmost column shows the complete system performance.

given for identifying and stemming a predicate correctly), and predicate classification (where full credit is given for correctly identifying, stemming, and disambiguating a predicate). Table 3.1 shows that most of the error comes from the classification step. Table 3.2 shows the performance of the full system compared to two baseline predicate classification modules. The first column, labeled ID+Stem+.01, shows the trivial baseline of assigning all predicates sense #1. The second column, labeled ID+Stem+MFS, shows the performance of the predicate classifier that predicts whatever sense is most frequent in the training data. The final column, labeled ID+Stem+CL, shows the full system performance.

### 3.5.2 Semantic Role Labeling

Once we have a set of predicates, either taken from the gold standard or predicted by the predicate labeling step, we are ready to begin labeling semantic roles. The semantic role labeling task is broken down into two parts – identification and classification.

	ID+Stem+.01	ID+Stem+MFS	ID+Stem+CL
Precision	80.1%	86.2%	92.4%
Recall	80.3%	86.5%	92.7%
F-score	80.2%	86.3%	92.5%

Table 3.2: Our approach for disambiguating predicates outperforms two baseline approaches. The leftmost column shows the performance of a classification module that always predicts sense #1, the center column shows the performance of always predicting the most frequent sense for that predicate, and the rightmost column reproduces the performance of the classifier suite. Identification and Stemming are held constant, using the same method as in table 3.1.

### Semantic Role Identification

The first step in the pipeline is to identify words that are likely candidates for semantic roles. This is accomplished by training a classifier over all words that are joined to a predicate by a syntactic dependency<sup>7</sup>. The identification model gives a binary result – either the word is identified as a candidate for a semantic role, or it is rejected.

The identification model uses the following features:

- **3 word window.** The candidate word and the words to its immediate left and right. The 3-word window size was chosen empirically on the development set.
- **CCG category.** The CCG category of the target word.
- **Predicate.** The numbered predicate, either taken from the gold-standard or predicted by the predicate disambiguation step.

<sup>7</sup>This differs slightly from the approach as described in Boxwell et al. [2009], where every word in the sentence is considered.

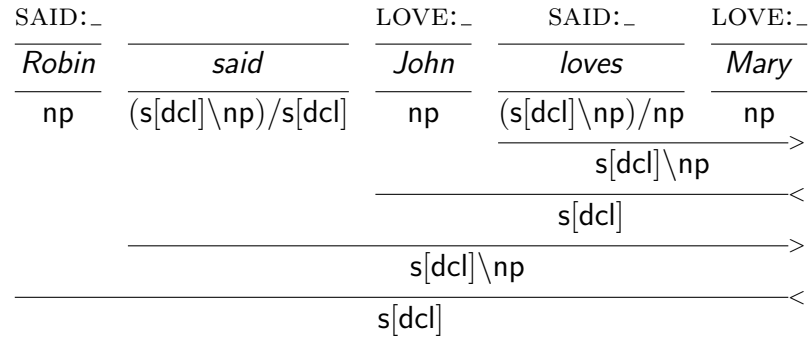


Figure 3.5: In the first stage of the semantic role labeling process, candidate semantic roles are chosen by the identifier model. We have not yet decided which role (ARG0, ARG1, etc) each word plays, only that there is a role there.

- **Before-After.** A binary indicator feature indicating whether the target word appears before or after the predicate.
- **Dependency.** A feature originally introduced by Gildea and Hockenmaier [2003], consisting of the predicate category, the argument slot the dependency fills, and whether the words are related as arguments or adjuncts, represented as an arrow. For example, the subject dependency for a transitive verb would be represented as (S[dcl]\NP)/NP.1.→. The dependency between a verb and a syntactic adjunct (say, a temporal modifier) would be ((S\NP)\(S\NP))/NP.2.←.

For each predicate in the sentence, the identifier model picks out individual words to pass on to the classifier. This is illustrated in figure 3.5, which shows a two-predicate sentence where candidate words have been identified, but not yet classified.

### Semantic Role Classification

Once a word has been chosen as semantic role candidate, the classifier predicts the best role for it. The features used for the semantic role classifier are identical to

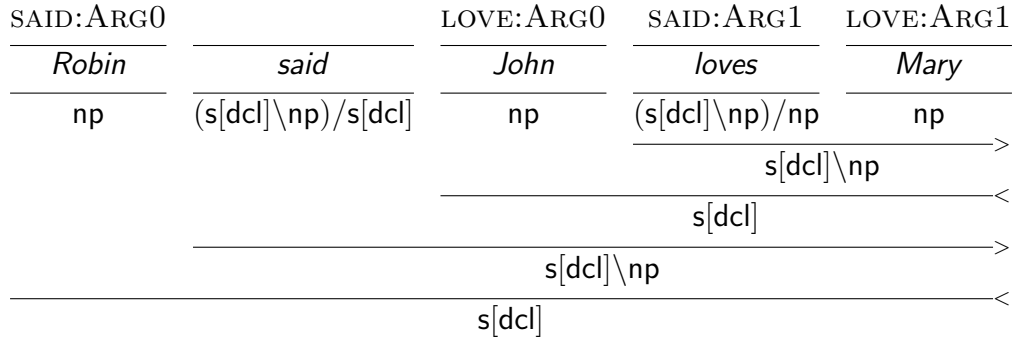


Figure 3.6: In the second stage of the semantic role labeling process, the classifier model sorts the roles into ARG0, ARG1, etc.

	ID	ID+CL	Punyakanok et al.
Precision	95.1%	88.0%	86.2%
Recall	87.7%	81.2%	87.4%
F-score	91.3%	84.5%	86.8%

Table 3.3: SRL performance, compared to the system of Punyakanok et al. [2008]

that of the identifier. This process is illustrated in figure 3.6. The performance of the semantic role labeler is shown in table 3.3.

### 3.6 Future Work

Punyakanok et al. [2008] introduced Integer Linear Programming to allow their system to incorporate information from a variety of syntactic parses instead of just one. Because parse errors are the cause of many semantic role labeling errors, this is an obvious advantage. Future work could incorporate a similar approach for n-best CCG parses. This would be of particular use to the approach described here, which

uses a rough-and-ready parse model (chapter 6) to generate the parses for semantic role labeling. Because we would have less faith in our parser model, it makes sense to consider the runner-up candidates as well.

### **3.7 Conclusion**

We showed in this chapter that we can link semantic and syntactic relations, using syntax to predict semantics. In chapter 5, we will examine the opposite process, using semantics to predict syntactic categories. But first, in chapter 4, we will describe chart parsing for CCG and its implications for semantic role labeling.

## CHAPTER 4: Automatic Parsing in CCG

### 4.1 Introduction

Chart parsing involves populating a large data structure with possible constituents until one or more spanning analyses are found [Kay, 1967, Younger, 1967, Kasami, 1965]. One potential pitfall of chart parsing is the high level of ambiguity in natural language – a wide-coverage grammar can cause a simple parse chart to rapidly grow to an unmanageable size, making it difficult to find a spanning analysis for sentences of more than about ten words. The reason for this is that each edge in a simple chart represents one and only one analysis for the span of words that it covers. In order to make chart parsing work with a wide-coverage grammar, we have to combine equivalent entries in the chart so they can be dealt with together.

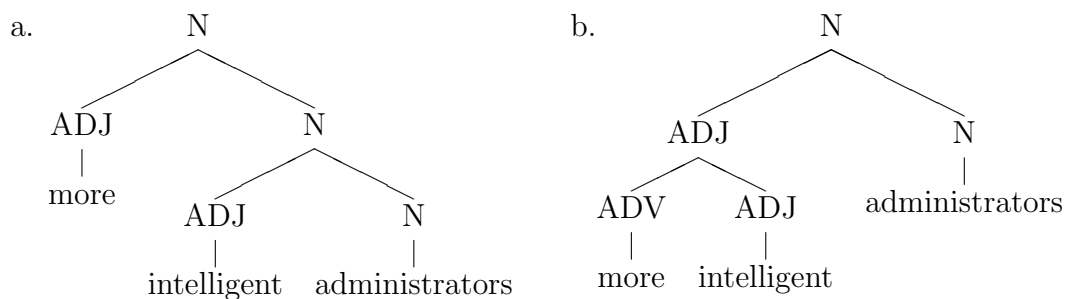


Figure 4.1: Two possible analyses for the phrase *more intelligent administrators*, the first meaning “a greater number of intelligent administrators” and the second meaning “administrators who are more intelligent”.

Consider the noun phrase *more intelligent administrators*. There are two possible analyses for this phrase, shown in figure 4.1. If we were parsing this sentence using, say, the CKY algorithm, the chart for this span of words would be like the one in figure 4.2. Each edge in the chart contains several pieces of data. Each edge has an identification number, a syntactic category, left and right indexes showing the span of the edge, and an ordered list of daughter edges (representing the edges that combine to make this edge), which we will refer to here as *memories*. In case of leaf nodes, the memory list may be empty, which we will show with the word corresponding to that leaf node. Notice that because there are two hypothesized categories for *more*, there are two separate edges for the 0-1 span.

Edge ID	Category	Left Index	Right Index	Memory
1	ADJ	0	1	<i>more</i>
2	ADV	0	1	<i>more</i>
3	ADJ	1	2	<i>intelligent</i>
4	N	2	3	<i>administrators</i>
5	N	1	3	3,4
6	ADJ	0	2	2,3
<b>7</b>	<b>N</b>	<b>0</b>	<b>3</b>	<b>1,5</b>
<b>8</b>	<b>N</b>	<b>0</b>	<b>3</b>	<b>6,4</b>

Figure 4.2: A parse chart for *more intelligent administrators*. Notice that edges 7 and 8 both cover the same span and have the same category.

Notice the boldface edges in figure 4.2 (edges 7 and 8). These two edges have the same syntactic category, and they cover the same span. This isn't immediately recognizable as a problem – in fact, it is algorithmically convenient when we go to enumerate the possible analyses for this sentence. If we wish to enumerate the possible analyses, we need only take each spanning analysis in turn (in this case,

there are two), and recursively follow the daughter indexes until we reach leaf nodes. If we represent this graphically, we will see two familiar syntax trees like the ones in figure 4.1. But what happens if we lengthen the phrase? Consider the sentence *more intelligent administrators supervised*. Figure 4.3 shows an expanded version of the chart in figure 4.2. Notice that for each analysis of the phrase *more intelligent administrators*, we need to predict a separate NP node and a separate S node.

Edge ID	Category	Left Index	Right Index	Memory
1	ADJ	0	1	<i>more</i>
2	ADV	0	1	<i>more</i>
3	ADJ	1	2	<i>intelligent</i>
4	N	2	3	<i>administrators</i>
5	VB	3	4	<i>supervise</i>
6	N	1	3	3,4
7	ADJ	0	2	2,3
8	N	0	3	1,6
9	N	0	3	7,4
<b>10</b>	<b>NP</b>	<b>0</b>	<b>3</b>	<b>8</b>
<b>11</b>	<b>NP</b>	<b>0</b>	<b>3</b>	<b>9</b>
<b>12</b>	<b>S</b>	<b>0</b>	<b>4</b>	<b>10,5</b>
<b>13</b>	<b>S</b>	<b>0</b>	<b>4</b>	<b>11,5</b>

Figure 4.3: A parse chart for *more intelligent administrators supervised*. Notice that there are now three pairs of identical edges (8 and 9, 10 and 11, 12 and 13). Not only does this waste space, but it requires the parser to apply the same grammatical rule twice each time, including any expensive unification computation.

Of course, at this point, at least the problem isn't getting much worse – we're only duplicating a few edges. But natural language is highly ambiguous. Consider the sentence *more intelligent administrators supervised children with telescopes*. There are multiple ambiguous phrases in this sentence, causing the size of the chart to expand greatly (figure 4.4). Notice now that the problem of redundant computations

is rapidly becoming more serious – if there are two analyses of the subject and two analyses of the predicate, this produces four analyses of the entire sentence, each of which must be computed independently. But each of these four computations is really the same:  $NP + VP = S$ . Four times this rule has to be retrieved from the grammar, four times an expensive unification process must be performed, and four times this same data must be stored in the chart. With a wide coverage grammar, this is unacceptable – a sentence of 30 or 40 words is impossible to parse in a realistic time frame.

## 4.2 Hypergraph-Based Chart Packing

To combat this explosion of chart entries, we need to recognize when computations are being repeated. If we can identify sets of edges that are combinatorially equivalent, then we can group them together, and then compute how they combine with their neighbors only once.

An edge A is combinatorially equivalent to another edge B if and only if:

- It has the same left index.
- It has the same right index.
- It has the same syntactic category.

Suppose we are parsing the original phrase *more intelligent administrators* using the CKY algorithm. We've added all of the edges of size 2 to the chart, and now we're about to add the edges of size 3. We identify the `[[more][intelligent administrators]]` edge and add it to the chart. Next, we identify the `[[more intelligent][administrators]]`

Edge ID	Category	Left Index	Right Index	Memory
1	ADJ	0	1	<i>more</i>
2	ADV	0	1	<i>more</i>
3	ADJ	1	2	<i>intelligent</i>
4	N	2	3	<i>administrators</i>
5	VB	3	4	<i>supervise</i>
6	N	4	5	<i>children</i>
7	P	5	6	<i>with</i>
8	N	6	7	<i>telescopes</i>
9	NP	2	3	4
10	NP	4	5	6
11	NP	6	7	8
12	PP	5	7	7,11
13	N	1	3	3,4
14	NP	1	3	13
15	ADJ	0	2	2,3
16	VP	3	5	5, 10
17	NP	4	7	10,12
18	N	0	3	1,14
19	N	0	3	15,4
20	NP	0	3	18
21	NP	0	3	19
22	VP	3	7	5,17
23	VP	3	7	16,11
<b>23</b>	<b>S</b>	<b>0</b>	<b>7</b>	<b>20,22</b>
<b>24</b>	<b>S</b>	<b>0</b>	<b>7</b>	<b>20,23</b>
<b>25</b>	<b>S</b>	<b>0</b>	<b>7</b>	<b>21,22</b>
<b>26</b>	<b>S</b>	<b>0</b>	<b>7</b>	<b>21,23</b>

Figure 4.4: A parse chart for *more intelligent administrators supervised children with telescopes*. There are now four spanning analyses, each of which must be handled independently.

edge. When we attempt to insert this edge into the chart, we notice that it is combinatorially equivalent to the `[[more][intelligent administrators]]` edge. For that reason, we “pack” the new edge into the old one, recording for future reference that there are two ways of creating this edge. The resultant chart is shown in figure 4.5.

Edge ID	Category	Left Index	Right Index	Memory
1	ADJ	0	1	<i>more</i>
2	ADV	0	1	<i>more</i>
3	ADJ	1	2	<i>intelligent</i>
4	N	2	3	<i>administrators</i>
5	N	1	3	3,4
6	ADJ	0	2	2,3
<b>7</b>	<b>N</b>	<b>0</b>	<b>3</b>	<b>{1,5; 6,4}</b>

Figure 4.5: A packed parse chart for *more intelligent administrators*. Edge #7 contains a set of lists of daughters, instead of a single list of daughters. This makes it clear that edge #7 can be formed by combining edges 1 and 5, or it could be formed by combining edges 6 and 4.

When the complete sentence *more intelligent administrators supervise children with telescopes* is represented in a packed chart, it becomes much more compact and requires many fewer computations (shown in figure 4.6). The improvement is easy to see – in the event that this sentence combines with another highly ambiguous edge (say, an edge with four of its own internal analyses), a total of 16 analyses will result. For sentences of 30 to 40 words (common in the newswire domain), the total number of edges in a simple, unpacked chart will dwarf the size of a packed chart.

This simple example demonstrates how a packed chart representation can be used to make wide coverage parsing possible. But how do we *unpack* the chart? Ultimately, we are interested in the individual analyses, not the knowledge that there is some

Edge ID	Category	Left Index	Right Index	Memory
1	ADJ	0	1	<i>more</i>
2	ADV	0	1	<i>more</i>
3	ADJ	1	2	<i>intelligent</i>
4	N	2	3	<i>administrators</i>
5	VB	3	4	<i>supervised</i>
6	N	4	5	<i>children</i>
7	P	5	6	<i>with</i>
8	N	6	7	<i>telescopes</i>
9	NP	2	3	4
10	NP	4	5	6
11	NP	6	7	8
12	PP	5	7	7,11
13	N	1	3	3,4
14	NP	1	3	13
15	ADJ	0	2	2,3
16	VP	3	5	5, 10
17	NP	4	7	10,12
18	N	0	3	{1,14; 15,4}
19	NP	0	3	18
20	VP	3	7	{5,17;16,11}
<b>21</b>	<b>S</b>	<b>0</b>	<b>7</b>	<b>{19,20}</b>

Figure 4.6: A parse chart for *more intelligent administrators supervised children with telescopes*. All four analyses are represented in the chart, but there is only one edge for the subject, one for the predicate, and one for the complete sentence. If this sentence were embedded in a complement clause (*The principal said that more intelligent administrators supervised children with telescopes*), then it would not be necessary to attach the clause 4 times.

analysis. To obtain the complete list of analyses, it is necessary to start at the spanning analysis for the category you are interested in (probably S), then walk backwards through the memory lists. The pseudocode for this is given in algorithm 1.

Using chart packing and algorithm 1 greatly improves on our ability to parse long sentences with wide coverage grammars. There is a remaining problem, though –

---

**Algorithm 1** The TREEGEN procedure, which enumerates all the analyses in a packed chart. This procedure is called on the spanning edge of the desired category (probably S). The procedure then calls itself recursively as it explores the chart exhaustively.

---

```
for all edge c in span do
  for all memory m in c do
    if m isLexical then
      result.add(new derivationnode(m))
    else if m isUnary then
      Collection daughts = treegen(m.unarydaught)
      for all derivationnode d in daughts do
        derivationnode addthis = new derivationnode(m)
        addthis.setdaughters(d)
        result.add(addthis)
      end for
    else if m isBinary then
      Collection leftDaughts = treegen(m.leftdaught)
      Collection rightDaughts = treegen(m.rightdaught)
      for all derivationnode l in leftDaughts do
        for all derivationnode r in rightDaughts do
          derivationnode addthis = new derivationnode(m)
          addthis.setdaughters(l,r)
          result.add(addthis)
        end for
      end for
    end if
  end for
end for
```

---

wide coverage grammars routinely produce thousands of analyses for average-length sentences. In all likelihood, though, most of these analyses will be erroneous – unlike the toy examples presented earlier, we would be hard-pressed to assign even dubious semantics to most these analyses. We are not interested in the worst of these analyses, only the best one, or perhaps the  $n$ -best analyses for some small value of  $n$ . We could enumerate all the analyses using algorithm 1, then sort the list according to some parse model. But this still requires us to unpack all possible analyses (a very inefficient process), most of which will be thrown away. Couldn't we just remove only the best analysis from the chart, leaving the others behind?

Doing so requires us to think clearly about how we identify desirable parses. One simple way to select a desirable parse is to assign probability scores to each grammar rule, then define the score of the complete analysis as the product of all the probability scores of each grammar rule. This is trivial to do for complete parses – we can simply walk the tree and keep a running total of the log probabilities of each step in the derivation. Doing this inside a regular, unpacked chart is similarly trivial – each edge simply adds the log probability of its own grammar rule to the log probabilities of each of its daughter nodes. This also makes it trivial to identify which spanning edge has the best score. But what about when multiple analyses are combined in a packed chart? The key to representing probability scores inside a packed chart is to recognize that the scores must be associated not with the *edges*, but with the *memories*. The edge is responsible for remembering the complete set of memories and its single-best memory. When an edge is added to the chart, the chart must determine if it can be packed into an existing edge, and if so, whether the new edge has a better score than the existing one. Each memory is responsible for remembering its “local” score (the

score for that single grammar rule) and its “global” score (the product of the local score and the best global scores of each of that memory’s daughters). The practical upshot of this is that it is possible to determine, for any given edge in the chart, which path to follow to find the single-best analysis for that span without enumerating every single parse. The pseudocode for the single-best unpacking algorithm is shown in algorithm 2.

---

**Algorithm 2** The SINGLE-BEST procedure, returns the single-best analysis in a chart, similar to how a Viterbi search would return the single-best path through a sequence of hidden states. This procedure is called on the spanning edge of the desired category (probably S). The procedure then calls itself recursively, but does not “branch” the way algorithm 1 does. If there is a tie for the single-best parse, the algorithm simply chooses one.

---

```
if thisEdge.bestMemory.isLexical then
    return new derivationnode(edge.bestMemory)
else
    daughters = new List
    for all edge daughter in thisEdge.bestMemory do
        daughters.add(single-best(daughter))
    end for
    return new derivationnode(thisEdge.bestMemory, daughters)
end if
```

---

### 4.3 Re-Ranking Candidate Parses Using Discriminative Model

At this point we must ask the question: where do these probability scores come from? We have a very large number of choices when it comes to ranking parses. We could use a simple maximum-likelihood method by taking the product of all rule applications, or we could identify likely patterns among daughter-parent-grandparent node groups, or we could even assign probabilities to large subtrees corresponding to lengthy phrases (like *devouring the steak with grilled onions*). In the context of a

packed chart, however, we are sharply limited in the nature of our ranking method. Because each edge contains a potentially large number of sub-analyses, certain features like the daughter-parent-grandparent node groups are inaccessible. Only information that is based on a single step in the analysis is admissible. If we use a simple generative model (like, say a maximum likelihood method), we can define the score of the entire derivation as the product of the scores of the individual steps, but the scores of the individual steps must be computed using only information local to that step. Clearly, this is problematic. Certain global features like the daughter-parent-grandparent node group feature are very informative, powerful features, particularly for problematic phrases like *the company stopped using asbestos in 1953*, where *in 1953* can attach to either *using asbestos* or *stopped using asbestos*, and a simple MLE approach is unlikely to distinguish between the two. Is it wise to sacrifice such strong predictors for the sake of wide-coverage bottom-up parsing?

One solution is to use a simple generative baseline to produce a short list of high-scoring candidate parses, then use a discriminative model to re-rank the resultant parses. This is appealing for several reasons. First, we need only compare a few parses, rather than all of them. The initial n-best list of analyses produced by the generative baseline model can be made smaller (ideally much smaller) than the complete list of possible analyses that would be produced by algorithm 1. Not only is it unnecessary to enumerate all parses (recall that we can enumerate only the single-best or n-best parses from the chart), but we need only consider a small number of parses for scoring and sorting. Secondly, the discriminative re-ranking model will have access to the full range of global features, and will not be limited to the restricted set

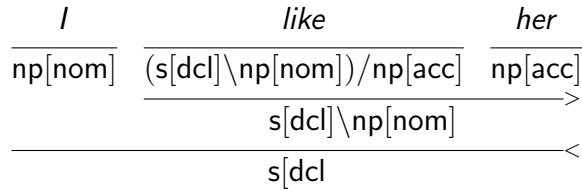


Figure 4.7: An example of forward application. The verbal argument `np[acc]` can unify with the target category `np[acc]`, so the verb can discharge an argument and produce the category `s[dcl]\np` to cover the span.

of features available inside the packed chart. Finally, having an n-best list of complete parses enables us to use other off-the-shelf tools (like CFG-style semantic role labelers, coreference resolvers, etc) before re-ranking, potentially providing additional information to help identify accurate parses.

#### 4.4 Packed Chart Parsing in CCG

Combinatory Categorical Grammar (CCG) presents some pitfalls to chart packing, but has many related benefits as well. CCG forms new categories by unifying argument categories with target categories. For example, consider the simple sentence in figure 4.7, which shows an analysis using a version of CCG that annotates nouns and verbal arguments for case. The verb *like* will become a sentence if it combines with an accusative object and a nominative subject. The pronouns *I* and *her* are given features corresponding to their cases.

When the verb *like* combines with the noun *her*, the category for *her* and the verbal argument must unify. In this case, both are `np[acc]`, so the two categories unify and the verb can discharge an argument to produce a category covering the span of both the verb and the argument. Now consider the ungrammatical sentence

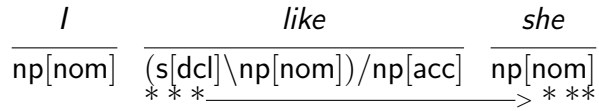


Figure 4.8: The ungrammatical sentence *I like she* is blocked by the feature unification – np[acc] on the verbal argument cannot unify with np[nom] on the target category.

in figure 4.8. In this sentence, the verb attempts to unify its argument np[acc] with the target category np[nom]. Because these categories do not unify, the derivation is blocked and the parser correctly determines that there is no spanning analysis.

Unification of argument categories can serve another purpose. One particularly exciting feature of CCG is its handling of syntactic dependencies. When lexical items are formed, they are assigned *heads* and *unfilled dependencies*. Each result category is given a head, and each argument is given an unfilled dependency. Figure 4.9 shows two CCG categories (represented as trees) with arguments and heads shown underneath terminal nodes in italics. Now, when these categories combine, the target category and the outermost argument of the verbal category attempt to unify. When the categories are unified, the unfilled dependency from the verbal argument  $\langle (S \setminus NP) / NP.2 \rangle$  is combined with the head  $\langle NP \rangle$  to generate a dependency between these two lexical entries. This represents a great improvement over a vanilla PTB-style syntactic representation, for which it is very difficult to represent syntactic dependencies. This is doubly true for *long range dependencies* that are mediated through intermediate categories, like in relative clauses. By clever co-indexation schemes, the standard syntactic category for *like* can be used in the construction *the girl that I like* to recover the object dependency between *like* and *girl*.

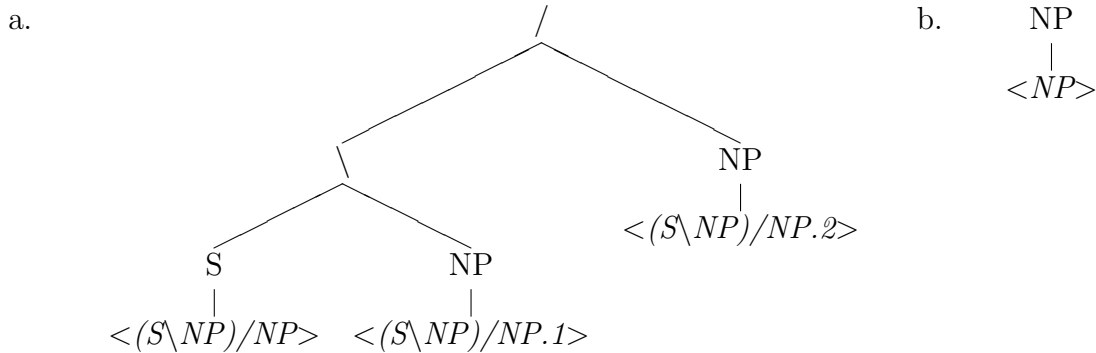


Figure 4.9: A simplified tree representation of the CCG category  $(S[decl]\backslash NP)/NP$  (a) and the CCG category  $NP$  (b). In the verbal category, the two  $NP$  arguments are assigned unfilled dependencies,  $\langle (S\backslash NP)/NP.1 \rangle$  and  $\langle (S\backslash NP)/NP.2 \rangle$ . The result category and, implicitly, all the nodes between it and the root node, are assigned a head. In the nominal category, there are no arguments, so there are no unfilled dependencies. The single node is assigned the head representing that lexical entry.

But what does this mean for packed charts when parsing in CCG? The crucial observation behind chart packing is that equivalent syntactic categories can be lumped together, because they will always have the same combinatoric properties later in the analysis. In a vanilla PTB style formalism, this is easy – the heuristics given earlier ensure that two edges are the same and will always have the same combinatoric properties. With CCG, for a new edge to be in an equivalence class with an existing edge, it must satisfy a stricter set of requirements. The original three rules stand, but two more must be added:

- It has the same left index.
- It has the same right index.
- It has the same syntactic category.

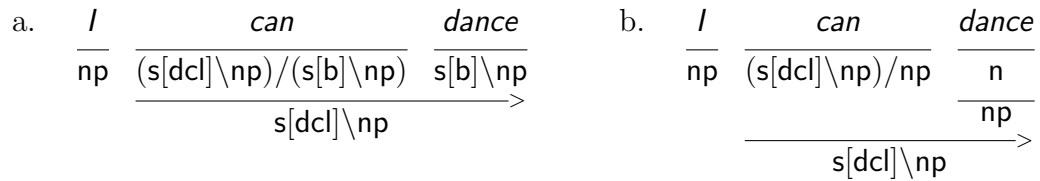


Figure 4.10: There are two ways to obtain the  $s[dcl]\backslash np$  category for the 1-3 span. The first corresponds to the intended reading, the second corresponds to the unlikely reading that I am storing some of the abstract noun *dance* for later in the winter. Should they be packed into a single edge?

- Its syntactic category must have the same pattern of unfilled dependencies on its arguments.
- Its syntactic category must have the same pattern of heads on its result categories.

The reason for including the last two requirements is that they influence the combinatory potential of the edge. In the event that two edges satisfy the first three requirements but not the second two, they will produce the same syntactic category (and the same set of syntactic derivations), but the dependency graphs will be different (which could lead to different semantic role assignments). Consider the sentence *I can dance*. The word *can* could have two possible categories:  $(S[dcl]\backslash NP_1) / (S[b]\backslash NP_1)$  (*I can swim, I can bend spoons with my thoughts*) and  $(S[dcl]\backslash NP) / NP$  (*I can tuna*). The word *dance* also has two possible categories:  $S[b]\backslash NP$  (*I like to dance, I think that he might dance*) and  $N$  (*the numa-numa dance went viral, the cha-cha slide is a childish dance*).

The parser dutifully begins combining categories, and determines that there are two possible analyses for the span 1-3 (figure 4.10). The parser further notes that the

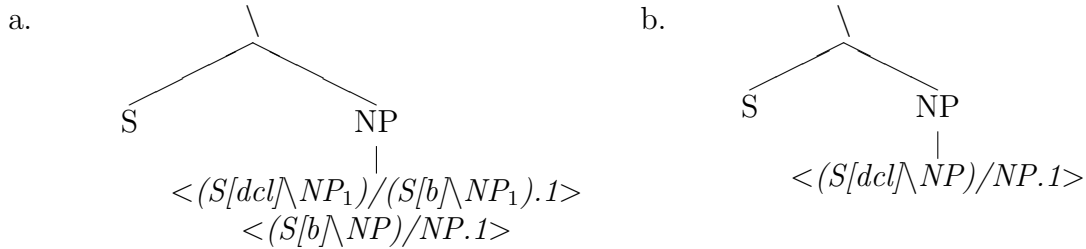


Figure 4.11: The  $S[dcl]\backslash NP$  category corresponding to the intended reading has two unfilled dependencies in the subject position – one for the modal verb, and one for the main verb. The  $S[dcl]\backslash NP$  category for the unlikely reading has only one dependency, originating from the declarative verb *can*. Because the unfilled dependencies are not equivalent, and therefore they may produce different results later on in the derivation, these two categories cannot be packed into the same edge.

categories look alike. So should they be packed into a single edge? The answer lies in the dependencies. As mentioned earlier, CCG dependencies are sometimes passed from their original categories to the heads they combine with by an intermediate category. This happens with modal verbs, relative pronouns, and verbs like *promise* and *persuade*. In this case, the modal sense of *can* passes the subject dependency of the verb it combines with to its own subject. This causes the subject dependency of *dance* to be linked to the subject *I*. So, the category for the verb phrase in the intended reading has two dependencies waiting to be filled by the subject of the sentence, while the verb phrase of the second analysis has only one dependency waiting to combine with the subject of the sentence (figure 4.11). For this reason, we cannot pack these two analyses into a single edge, as they produce different dependency graphs later in the derivation, and are therefore not strictly combinatorially equivalent. We could deal with this after unpacking, but this would prevent us from doing experiments on semantic role labeling inside the packed charts (section 4.5).

## 4.5 Semantic Role Labeling and Packed Charts

In the past, semantic role labeling and parsing have, for the most part, been considered a two-step process. Each sentence is given a syntactic analysis by an automatic parser, then these analyses are fed into a semantic role labeler. In some cases, a semantic role labeler might take input from more than one parse. Punyakanok et al. [2008] demonstrate improved SRL performance by taking input from both Collins’ parser [Collins, 2003] and Charniak’s parser [Charniak, 2001], in some cases taking n-best parses as well. In this section, we will describe a process to combine parsing and semantic role labeling, performing SRL decisions inside a packed chart [Boxwell et al., 2010].

Many efforts have been made to combine syntactic parsing with semantic role labeling into a single process, in particular the CoNLL 2008 shared task [Surdeanu et al., 2008], but in practice most of the systems organize this joint effort as a pipeline, with the parser informing the semantic role labeler. The systems in the 2008 shared task that do integrate parsing and semantic role labeling will introduce it as an intermediate step in a shift-reduce dependency parser [Henderson et al., 2008] or as part of an ensemble of parsers [Samuelsson et al., 2008], or by adapting the dependency relation tags to bear semantic as well as syntactic information [Sun et al., 2008]. The system in the CoNLL 2008 shared task that might have the best claim to a truly joint effort is that of Lluís and Màrquez [2008], which adapts the Eisner algorithm [Eisner, 1996] to predict both syntactic and semantic dependencies, though they somewhat cryptically admit that “[semantic] features extracted from the syntactic tree...are not available or expensive to compute within the Eisner algorithm” and that they “overcome this problem again with a very simple (though not elegant) solution, consisting

of introducing a previous syntactic parsing step” [Lluís and Màrquez, 2008, section 1, right column, line 12]. The paper does not elaborate, and the system performs poorly compared to other systems in the shared task [Lluís and Màrquez, 2008, section 4] [Surdeanu et al., 2008, table 16]. None of these approaches use a bottom up CKY-style packed-chart approach to parsing. The Eisner algorithm is similar to CKY in many ways [Eisner, 1996, section 3]; it acknowledges the reality of wide-coverage chart-based parsing by stating that “multiple analyses have the same **signature** if they are indistinguishable in their ability to combine with other analyses” (boldface original), but declines to attempt a chart-packing approach by determining that if there are multiple analyses, “the parser discards *all but the highest scoring one*” (italics mine). Effectively, this prevents the parser from providing multiple candidate parses, eliminating the possibility of integrating a discriminative re-ranking process like the one discussed in section 4.3 later on in the pipeline. The Eisner algorithm could be adapted to produce n-best dependency graphs, though the potential pitfalls of this remain unclear. The CoNLL 2009 shared task is similarly patterned – the systems rely heavily on a parser-SRL pipeline.

So is truly simultaneous parsing and semantic role labeling possible, and if it is, is it worthwhile? Notice that the systems from the CoNLL shared task overwhelmingly prefer dependency parsing. When we consider the information needed to pursue semantic role labeling, it becomes clear why. One of the most informative features in semantic role labeling is the treepath feature. The treepath feature, described in detail by Gildea and Jurafsky [2002] and borrowed by many since, is a very rich feature with strong predictive power. But in the context of a packed chart, it is clear that this feature is unusable. Consider the case shown in figure 4.12. Because we claim

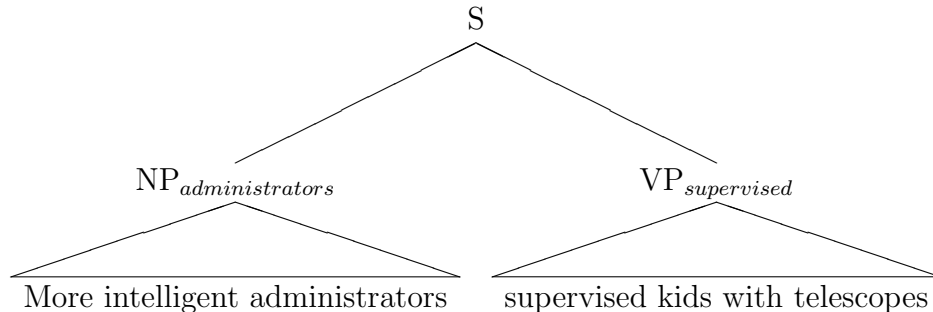


Figure 4.12: In the context of a packed chart, it is meaningless to speak of a treepath between *supervised* and *administrators* because multiple analyses are “packed” under a single category.

to know nothing about the internal structure of edges in the chart, we are unable to follow the treepath from predicate to argument. Certainly, we could use the treepath feature in a semantic role labeling system after the parse has been produced, but at that point we have committed ourselves to an analysis, and the system is organized in a pipeline fashion again. If there were some way to identify a detailed syntactic dependency between the predicate and its argument at the time of their combination high in the tree, like in a dependency grammar, this could be used to predict a semantic role – in a vanilla PTB-style formalism, this is not possible.

Fortunately, Combinatory Categorical Grammar, provides just such a tool. Recall that when performing packed-chart parsing in CCG, it is necessary represent the unfilled dependencies on the arguments themselves, and to only “pack” categories that have the same unfilled dependencies. For this reason, a filled syntactic dependency is produced *at exactly the moment that we wish to perform semantic role labeling*. That is, suppose we are combining the phrases *devoured* and *the big, juicy, mouth-watering piece of steak that the waitress brought me*. The treepath feature is unavailable. Even

if we simply assumed that a semantic role belongs on every span that the predicate combines with (a dubious claim, doubly so for a formalism like CCG), this gives us little information on the nature of their relationship (Direct object? Indirect object? Subject?). The nature of the grammatical relation between the predicate and its object is found in the syntactic dependency. This dependency, combined with other “local” features, can make up a rich feature set which can be used to predict semantic roles or train a semantic role labeling model.

There are various weaknesses to this approach, though none are particularly fatal. One persistent problem is that of memory usage. A parse chart, even a packed one, can take up very large amounts of space. For some lengthy sentences in the WSJ corpus, the parser cannot cope with the very large number of edges, even when they are combined into equivalence classes. Fortunately, much parsing research limits the length of sentences to 40 words or less, which is well within our means – the sentences that continue to cause problems in the WSJ corpus are 100+ word sentences, mostly uninteresting semicolon-separated lists of attorneys who have been disbarred or assigned various penalties.

Another potential weakness is the need to store SRL models in memory, during the parsing process, which makes the memory problem worse. There is a solution to this, though perhaps an inelegant one. Because we do not include a syntactic pruning step, it is not necessary to assign scores to the rule applications at parse time. Unlike Eisner’s algorithm (which, recall, discards all but the highest scoring edge for each equivalence class), we keep all memories in case we are asked for a second, third, or n-best analysis. Therefore, the chart could easily be written to disk in its packed form (imagine a format like that of figure 4.6), and each edge in the chart could be

evaluated independently from all others. Once the roles were predicted, the chart could be re-scored according to whatever parse model (or models) we decide upon before it is unpacked.

The major weakness to this approach is that it does not permit the use of non-local features. Boxwell et al. [2009] show that certain global features like the treepath feature of Gildea and Jurafsky [2002] give a modest improvement for CCG semantic role labeling. By the nature of a packed-chart parser, however, it is impossible to integrate these features at parse time. They are, of course, available after parsing in a discriminative re-ranking step, and this may or may not be sufficient for our purposes. One option is to use local features to perform SRL to generate features to select a strong parse, then re-label the derivation once it has been chosen [Boxwell et al., 2010]. To do this, we would predict semantic roles inside the packed chart, using the local features that are available to us. Then, we use those semantic roles to guide the parse model in selecting an n-best list from the parse chart, choosing parses that predict likely semantic roles (figure 4.13). Then, we re-label the n-best list of candidate parses using the full, global feature set (which includes features like the treepath feature, grandparent features, etc) to get even better results.

## 4.6 Other Approaches to Parsing in CCG

Another approach to parsing in CCG which could have positive implications for semantic role labeling is using a shift-reduce parser. Using a shift-reduce parser like that of Zhang and Clark [2011] would allow us to avoid the problem of a packed chart by employing a small beam and greedy-search decoding step. This has the further advantage of storing a complete analysis at parse time, which would enable



us to enabling us to perform SRL without the clunky machinery of packed-chart SRL described in section 4.5. Furthermore, Zhang and Clark [2011] show that their shift-reduce parser is competitive with the state-of-the-art CCG chart parser of Clark and Curran [Clark and Curran, 2004b].

So why not use the shift-reduce method instead of a chart parser? The answer lies in the different kinds of errors that the two parsers make. Consider that dependency recall errors are particularly devastating to semantic role labeling – failure to link *books* and *read* in the phrase *the books that John read* with a syntactic dependency would make it very difficult for the Brutus system [Boxwell et al., 2009] to predict a role on *the books*. Zhang and Clark [2011] give an excellent comparison of the performance of the two CCG parsers on the most common dependency types. The shift-reduce parser outperforms the chart parser on dependency recall for dependencies originating from N/N, NP/N, and (NP\NP)/NP. The chart parser, however, outperforms the shift-reduce parser for dependency recall for dependencies originating from ((S\NP)\(S\NP))/NP, (S\NP)\(S\NP), and (S[*dcl*]\NP)/NP. Notice that the second set of categories is certainly more important for semantic role labeling than the first set – the second set represents syntactic relations between verbs and their arguments, whereas the first set centers around noun-phrase internal structure. It would be interesting to see what effect this would have on semantic role labeling in concert with our approach, for now we will set this aside for future work.

## 4.7 Conclusion

In this chapter, we described the pitfalls of chart-parsing with a highly ambiguous grammar, and a way to overcome them. We also described the problems and limitations of performing semantic role labeling at parse time inside a packed chart. These problems and their solutions, however, are meaningless without the lexical categories that are used to begin the parsing process. In the next chapter, we will describe a method of using semantic roles to generate syntactic categories for lexical items when we have no treebank to work with.

## CHAPTER 5: Learning to Do Without a Treebank

### 5.1 The Problem with Treebanks

Human-annotated syntactic corpora are phenomenally useful resources – the use of large-scale syntactic corpora brought in a new era of data-driven linguistics. However, their greatest strength (the fact that they represent human judgements) is also a great weakness. Syntactic corpora are costly to develop, both in time and resources. They require a small army of annotators who must be familiar with the syntax formalism. These same annotators must be fluent and literate in the target language (not a problem for English, but certainly an obstacle for an exotic language). These obstacles are particularly relevant to the development of NLP systems for government applications – world events sometimes require the rapid development of new language technology for languages that have not previously been the focus of NLP research. But what if we were able to do semantic role labeling without the expense of a treebank? What if we could do away with the human annotated syntactic treebank entirely, relying only on a semantic Propbank-like resource? If achievable, this would greatly reduce the up-front cost to NLP research in new languages and expedite the development of new NLP systems for new critical languages. In this chapter, we will describe and expand on a method described in Boxwell et al. [2011]. We will also show the importance of CCG to the overall approach, as the lexical predictions made here can not be simulated using a traditional POS tagset and context free grammar.

## 5.2 Constructing Treebanks

Unsupervised approaches to inducing syntactic structure are noisy. EM-based syntax-inducement approaches like Paskin [2002] perform slightly worse than chance. Only when some additional information is incorporated do EM-based approaches become successful – Klein and Manning [2004] show that syntactic structure can be induced by incorporating a notion of valence into the grammar, and Zettlemoyer and Collins [2005] show success with guiding a derivation based on detailed semantic information given in the form of the lambda calculus. In this and the following chapters, we advocate an approach similar in spirit to that of Zettlemoyer and Collins [2005], but very different in implementation.

## 5.3 Predicting CCG Lexical Categories

Now, assume we have a POS-tagged corpus. In order to do semantic role labeling, we will probably need some kind of syntactic information, and to generate any kind of CCG parse we will need CCG supertags. There are ways to generate CCG supertag sequences using the forward-backward algorithm if we have a tag dictionary of words to CCG categories [Baldrige, 2008, Ravi et al., 2010]. But generating a word-to-CCG supertag dictionary is probably prohibitively difficult in the absence of a CCGbank. A POS-to-supertag dictionary might be feasible to construct by hand (Determiner  $\rightarrow$  NP/N), but it would be difficult to accurately account for highly ambiguous parts of speech (in sections 2-21, there are 619 distinct categories assigned to verbs). A finite-state approach would be hard pressed to choose the correct one in a particular context. What if we could narrow down the choices of the most ambiguous tags using

Propbank data?

### 5.3.1 Verbal Categories

We can use propbank data to accurately predict verbal categories. Consider the following sentence, which has been annotated with span-based semantic roles:

the	man	gave	a	flower	to	the	policeman	yesterday
Arg0		rel	Arg1		Arg2-to		ArgM-TMP	

From this annotation, we know a few things about the verbal category that we did not know before:

- The verbal category has three syntactic arguments, one for each numbered argument.
- The syntactic argument corresponding to *to the policeman* is a prepositional phrase with the head *to*.
- One of the syntactic arguments should appear to the left of the verb, and the other two to the right.<sup>8</sup>

If we begin adding syntactic arguments to the base of S[dcl], starting from the beginning of the sentence and moving in to the verb, then starting at the end of the sentence and moving in to the verb, we construct the complete verbal category:

- Start with the base verbal category, S[dcl].

<sup>8</sup>The non-standard word order of a relative clause can be detected by the presence of an R tag on one of the roles (Arg0R, Arg1R, etc). In this case, we pretend that non-subject relativized roles appear to the right of the verb, even if they actually appear to the left (eg. books that Jon likes). This also covers Wh-movement (eg. what books does Jon like). Note that including the “R” modifier on the tag could make annotation more difficult.

- The leftmost numbered semantic argument is Arg0. So, add an NP argument to the left, resulting in  $S[\text{dcl}]\backslash\text{NP}$ .
- The rightmost numbered semantic argument is Arg2-to, indicating a prepositional phrase. So, add a PP argument to the right, resulting in  $(S[\text{dcl}]\backslash\text{NP})/\text{PP}$ .
- The next numbered semantic argument is Arg1, indicating an NP. So, add a NP argument to the right, resulting in  $((S[\text{dcl}]\backslash\text{NP})/\text{PP})/\text{NP}$ .

Alternate features for the S category (declarative, base form, passive, perfect, etc) can be determined either by heuristics or by the assumed gold-standard part-of-speech tag. For example, if the preceding word is *was*, this strongly suggests that the verbal category should start with  $S[\text{pss}]$  instead of  $S[\text{dcl}]$  (the category of *seen* in *Robin was seen* is  $S[\text{pss}]\backslash\text{NP}$ ). Also, we include the special case of participles (verbs with tags VBG and VBN) with only a single argument that immediately follows it, which are assigned the category  $N/N$  (*dutch **publishing** group, a **managing** director, the **continuing** statist, restrictive, nondemocratic programs*). We could choose to assign these traditional verbal categories and then use a unary type-changing rule to convert them into the appropriate adjectival category (e.g.  $(S[\text{ng}]\backslash\text{NP}_1)/\text{NP} \rightarrow N_1/N_1$ ), but we choose not to because this would disrupt the Markov Model tagging process described in section 5.3.5 and because it would complicate comparison to the original CCGbank.

This basic approach agrees with the CCGbank at a rate of 71.4% on the development set (WSJ section 00). The most common errors are those of prepositional phrase attachment. Sometimes the Penn Treebank and the CCGbank disagree about whether a prepositional phrase should be annotated as an argument or an adjunct

[Gildea and Hockenmaier, 2003, Honnibal and Curran, 2007, Boxwell and White, 2008]. Errors of this kind are uninteresting to us, as the problems with applying Propbank argument-adjunct distinctions to CCGbank for SRL purposes are well recognized by its creator: “the performance... is lowered by the fact that the syntactic analyses in the training corpus differ from those that underlie PropBank in important ways (in particular in the notion of heads and the complement-adjunct distinction)” [Gildea and Hockenmaier, 2003, S 7]. For our purposes, whether or not the verbal category is expecting to combine with an argument or an adjunct is less important than the verbal category and the prepositional category being consistent. If we compare our predicted verbal categories to those in the modified CCGbank of Boxwell and White [2008], which modifies the CCGbank verbal categories to bring them in line with the Propbank judgements (or, equivalently, if we simply ignore errors of this kind), the previously described approach achieves 78.2% agreement with the CCGbank on the development set. We will have more to say on the mismatching of syntactic and semantic arguments later.

The remaining errors are primarily a mix of the following types:

- **Incorrect Sentential Feature.** The most common cause of this error is an incorrect POS tag. For example, in wsj\_0093.34, *the guards retained their pistols*, the verb *retained* is erroneously given the VBN (past participle) tag in the gold standard, leading to an erroneous (S[pt]\NP)/NP category assignment. If the correct VBD (past tense) tag had been assigned, then the correct CCG supertag (S[dcl]\NP)/NP would have been predicted.
- **Incorrect Argument Category.** Consider wsj\_0090.5, *I don't think it's a consideration*. In this sentence, the correct CCG supertag for *think* is (S[dcl]\NP)/S[dcl].

From the propbank annotation, though, there is no distinction between sentential and noun phrase arguments (prepositional phrase arguments are conveniently labeled as ARG1-FOR, ARG2-WITH, etc, but there is no such convention to indicate sentential arguments). Some common verbs like *said* and *reported* appear almost exclusively with sentential objects, but others appear with both nominal and sentential objects (*feel*, *note*, *find*, *learn*, *discover*, and *suggest*, to name a few). Choosing the verbal category (S[dcl]\NP)/NP instead of (S[dcl]\NP)/S[dcl] is a common error.

- **Adjective-like Predicates.** In many cases, progressive verbs are treated like noun modifiers in the CCGbank. For example, in wsj\_0088.31, *jurisdictional problems in regulating program trading*. The CCGbank correctly assigns the category (S[ng]\NP)/NP. The process described here, however, predicts the category N/N, just like *publishing* in *Dutch publishing group*. The semantic difference is real: in *publishing group*, the group is the thing doing the publishing – in *regulating program trading*, the program trading is not the thing doing the regulating. This distinction is difficult for a set of heuristics to grasp, and remains error-prone.

Additional work could no doubt reduce these errors, but for now we shall push ahead with the primary thrust of the work, leaving further refinement of these heuristics to future research.

### 5.3.2 Auxiliary Verbs

Oftentimes, predicates are preceded by some kind of auxiliary verb<sup>9</sup>. These auxiliary verbs precede verbs in progressive, base, and passive participle forms, and almost invariably take the form  $(S[dcl]\backslash NP)/(S[*]\backslash NP)$ , where  $*$  indicates some feature marked on the verbal category that follows. For example, in the sentence *he was running*, *running* has the category  $(S[ng]\backslash NP)/NP$  and *was* has the category  $(S[dcl]\backslash NP)/(S[ng]\backslash NP)$ . In the sentence *he was fired*, *fired* has the category  $(S[pss]\backslash NP)/NP$  and *was* as the category  $(S[dcl]\backslash NP)/(S[pss]\backslash NP)$ . The crucial observation is that these auxiliary verbs almost always yield a declarative result<sup>10</sup>. The only consistent exception to this is the word *to*, which behaves much like a modal verb. The word *to* consistently produces a category of  $(S[to]\backslash NP)/(S[*]\backslash NP)$ . By conditioning the categories on the verb that follows it, we can achieve 93% agreement with the CCGbank on auxiliary verbs. Almost all of the errors are the result of errors in the previous step – for example, predicting  $(S[dcl]\backslash NP)/(S[pss]\backslash NP)$  instead of  $(S[dcl]\backslash NP)/(S[pt]\backslash NP)$  because the verb following the auxiliary verb was predicted as  $S[pss]\backslash NP$  instead of  $S[pt]\backslash NP$ . Altogether, the agreement rate including predicates and auxiliary verbs is 75.6%.

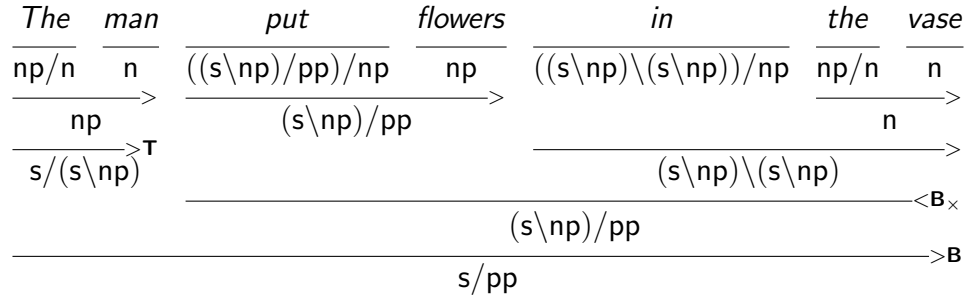
### 5.3.3 Prepositional categories

Propbank can also help us determine the categories of prepositions. Prepositions are particularly problematic for CCG. There are four main categories for CCG prepositions:

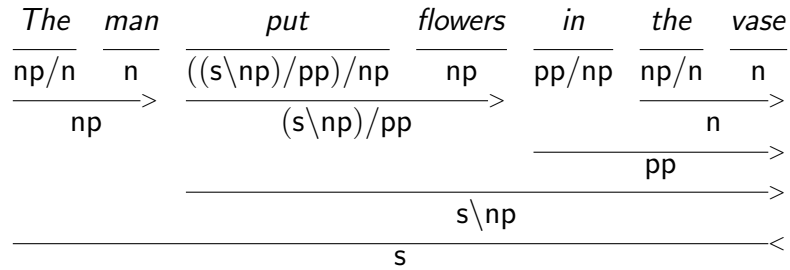
<sup>9</sup>The auxiliary verbs included in this step are *will*, *was*, *were*, *'re*, *are*, *is*, *has*, *have*, *can*, *could*, *would*, and *had*.

<sup>10</sup>complex modal constructions are extremely rare – the classic *will have had* construction appears only once in the entire treebank.





The preposition should have been assigned the argument-head category PP/NP:



The mismatch between the categories of prepositions and their corresponding verbs is particularly deadly to this approach to supertagging. It is possible to learn the correct state sequence for phrases like “*belong to...*” and “*infringed upon...*” with a hidden markov model (as the hidden states are next to each other, and so the verbal category can inform the prepositional category and *vice versa*). But the correct state sequence for sequences like “*put flowers in...*” and “*devoured steak on...*” are impossible for this Bayesian approach to learn because there is an intermediate state between the verbal category and the preposition.

Fortunately, just as Propbank provides us with guidance for constructing a verbal category, it also provides us with guidance about the prepositional phrases. Prepositional heads of numbered roles can be deterministically assigned the category PP/NP, and prepositional heads of modifier roles can be assigned the category  $((\text{S}\backslash\text{NP})\backslash(\text{S}\backslash\text{NP}))/\text{NP}$  or  $(\text{S}/\text{S})/\text{NP}$ , when they appear after or before the predicate, respectively. At the same time, one-word modifier roles can be assigned

$(S\backslash NP)\backslash(S\backslash NP)$  or S/S. The remaining prepositions are likely  $(NP\backslash NP)/NP$ . The approach described here predicts prepositional categories that agree with the CCGbank at a rate of 69.3%. Many of the discrepancies between the predicted categories and the CCGbank are consistent with the problematic argument-adjunct mapping – in about 13% of the cases, the system predicts the category PP/NP when the CCGbank has  $((S\backslash NP)\backslash(S\backslash NP))/NP$ , and in about 2% of cases the system predicts the category  $((S\backslash NP)\backslash(S\backslash NP))/NP$  when CCGbank has PP/NP. Ignoring both of these classes of mismatches yields an overall 83.7% agreement with CCGbank on prepositions. As mentioned earlier, errors of this type are unimportant to us, so long as the preposition category is consistent with the verbal category. This brings our total rate of agreement with CCGbank at 74.3%. If we ignore all errors related to argument/adjunct mismatches, our rate of agreement is 82.4%.

### 5.3.4 Common Closed-Class Words

There is a short list of closed-class words that are not covered by the previous steps. The system allows us to, at this point, enumerate a list of words and their categories, and tag words that were missed by the previous steps. This list can be of an arbitrary length. The list that we use is limited to common determiners, auxiliary verbs, and the word *of*<sup>11</sup>. The complete list is shown in Appendix A. This approach is especially effective for words like *the*, *a*, and *an*, which are very common but highly uniform in their categories. It is also effective for filling in the gaps left by the previously described auxiliary verb predictor. Altogether, the previous steps yield a

<sup>11</sup>*of* almost always takes the category  $(NP\backslash NP)/NP$ , and therefore is not covered by the preposition prediction heuristics discussed earlier.

rate of agreement with CCGbank of 81.1% (86.1% if we ignore the argument-adjunct errors).

### 5.3.5 Predicting Categories with a Partially Hidden Markov Model

The previous steps generate at least one category for about 31% of the words in the development set. In order to fill in the gaps, we will borrow and adapt a technique used by Baldridge et. al (ibid) and use a hidden markov model to label the remaining words. We will specify an initial model as follows. The start probability of the category  $c \in C$  is defined as

$$S\{c\} = \frac{hasNoLeft(c)}{\sum_{c_n \in C} hasNoLeft(c_n)} \quad (5.1)$$

And the end probability of the category  $c \in C$  is defined as

$$E\{c\} = \frac{hasNoRight(c)}{\sum_{c_n \in C} hasNoRight(c_n)} \quad (5.2)$$

where  $hasNoLeft(c)$  and  $hasNoRight(c)$  return 1 if the category  $c$  has no left or right looking arguments, respectively, and 0.01 otherwise. Effectively, this increases the likelihood that categories with no left-looking arguments (like NP/N) start sequences, and lowers the likelihood that categories with left-looking arguments like (S[dcl]\NP)/NP start sequences. Similarly, it increases the likelihood that categories with no right-looking arguments (like S[dcl]\NP) end sequences, and lowers the likelihood that categories with right-looking arguments like (S[dcl]\NP)/NP end sequences.

The transition probability between states  $c_i$  and  $c_j$  where  $c_i, c_j \in C$  is defined as follows:

$$T\{c_i, c_j\} = \frac{\text{bin}(c_i, c_j)}{\sum_{c_n \in C} \text{bin}(c_i, c_n)} \quad (5.3)$$

where  $\text{bin}(c_i, c_j)$  returns 2 if the categories  $c_i$  and  $c_j$  can combine using function application, 1 if the categories  $c_i$  and  $c_j$  can combine using any other CCG combinator, and 0.01 otherwise. Effectively, this increases the likelihood that categories that can combine according to the combinatory rules of CCG will appear next to each other in the sequence.

The emission probability of observation  $o_i$  from state  $c_j$  is defined as follows:

$$M\{o_i, c_j\} = \frac{\text{dic}(o_i, c_j)}{\sum_{o_n \in O} \text{dic}(o_n, c_j)} \quad (5.4)$$

where  $\text{dic}(o_i, c_j)$  returns 1 if the observation  $o_i$  appears in the tag dictionary for state  $c_j$ , and 0.01 otherwise.

Now that we have our initial model, we can train it using a modified version of the Baum-Welch algorithm [Rabiner, 1989] that takes into account the words that we have already tagged. To properly visualize the modified Baum-Welch algorithm, imagine the familiar soda machine example [Jurafsky et al., 2000, Manning et al., 1999]. Imagine a malfunctioning vending machine that dispenses two different beverages (Dr. Pepper and Sprite) semi-randomly, but has been equipped with a lever by two graduate students, Steve and DJ, to prefer Dr. Pepper over Sprite and *vice versa*. When the machine is in state ST, it will dispense primarily Dr. Pepper (70% of the time), but when the machine is in state DJ, it will dispense primarily Sprite (60% of the time). Now, suppose that lightning strikes the machine, causing the lever to become inoperative and the machine to begin switching semi-randomly between the two states every time a soda is dispensed (staying in the same state 60% of the time

and switching to the other state 40% of the time). If we are presented with a stream of observations, we can use the forward-backward algorithm to estimate the likelihood of being in any one state at any given time, and we can use the Viterbi algorithm to calculate the most likely underlying state sequence. For example, if the state sequence is as follows, the forward-backward algorithm and the Viterbi algorithm make the following predictions:

OBS	dp	sprite	sprite	dp	dp	dp	sprite
ST	.72	.20	.20	.75	.78	.75	.22
DJ	.28	.80	.80	.25	.22	.25	.78
Viterbi	ST	DJ	DJ	ST	ST	ST	DJ

Now, let us depart from the traditional notions of hidden markov models, where the state sequence is entirely unobservable. Suppose that we have some limited information about the underlying state sequence. Perhaps we managed to install a rootkit in the vending machine operating system, or perhaps we hid behind the machine and observed the underlying state, or perhaps we consulted the Delphic Oracle. Our source informs us that at time index 3, the machine was actually in state ST, the Dr. Pepper preferring state. Armed with this additional information, we set out to revise our estimates of the underlying state sequence:

OBS	dp	sprite	sprite	dp	dp	dp	sprite
ST	.74	.29	<b>1.0</b>	.84	.80	.76	.22
DJ	.26	.71	<b>0.0</b>	.16	.20	.24	.78
Viterbi	ST	ST	ST	ST	ST	ST	DJ

Notice that this impacts not only the state at time index 3 – having certainty that the machine is in state ST influences the likelihood of the states around time index 3. The implications of knowing that the machine was in state ST at time index 3 spread throughout the lattice. It’s even enough to change the Viterbi algorithm’s decision about the underlying state at time index 2.

Consider now the implications of the oracle predicting state DJ at time index 5:

OBS	dp	sprite	sprite	dp	dp	dp	sprite
ST	.72	.20	.19	.64	<b>0.0</b>	.64	.20
DJ	.28	.80	.81	.36	<b>1.0</b>	.36	.80
Viterbi	ST	DJ	DJ	DJ	DJ	DJ	DJ

Clearly, having oracle information about a particular time index can influence the entire neighborhood of the observations. This has the potential to greatly impact our efforts to fill in the blanks of our CCG category sequence. If we treat the Propbank-powered predictions of the verbal categories and prepositions as oracle predictions, we can ask the Partially Hidden Markov Model (PHMM) to fill in the gaps. Consider the impact of a PHMM on the simple sentence described earlier. Suppose that we give the PHMM model no oracle information (making it equivalent to a regular HMM). The following state sequence is produced by the Viterbi algorithm:

Words	Predicted Category
the	NP/N
man	N
gave	((S[ <i>dcl</i> ]\NP)/(S[ <i>ng</i> ]\NP))/NP
a	NP/N
flower	N
to	(NP\NP)/NP
the	NP/N
policeman	N/N
yesterday	N

This is the method that is used in Boxwell et al. [2011]. But this is clearly unacceptable – the system chooses the wrong category for *gave*, an incorrect prepositional category, and predicts a nominal category for *yesterday*, causing *policeman* to try to rescue the sequence by choosing N/N<sup>12</sup>. It’s not difficult to see that these tags offer no hope of finding a spanning analysis, much less a sentential one. But what if we

<sup>12</sup>If you think that the problem is that *yesterday* is incorrectly annotated because its part-of-speech tag is NN, consider two things: first, the word *tomorrow* is consistently annotated as NN in the Penn Treebank, and second, the arguably more appropriate adverbial tag RB results in the system predicting (N/N)\(N/N) (which combines with the preceding N/N via function application).

included the information suggested by propbank? We could force the PHMM into the given states for *gave*, *to*, and *yesterday*, then let it fill in the gaps as best it can:

Words	Predicted Category
the	NP/N
man	N
gave	<b>((S[dcl]\NP)/PP)/NP</b>
a	NP/N
flower	N/N
to	<b>PP/NP</b>
the	NP/N
policeman	N/N
yesterday	<b>(S\NP)\(S\NP)</b>

This is a big improvement – the tags that are most ambiguous (and therefore most difficult to predict) are exactly the tags that Propbank can help us predict. There is a problem, however. The word *policeman* is still incorrect, and the word *flower* is incorrect now but was correctly predicted before. Why is this?

The reason for the errors comes from the way that the transition probabilities are initialized. Tags that combine with each other via CCG combinatorics are encouraged, and tags that fail to combine are all discouraged. The side effect of this is that categories like PP/NP are equally unhappy to be preceded by N as they are N/N. That is, there is no way that the word *flower* can find a good way to transition to *to*, so the two are roughly equally likely. From a linguistic perspective, this should not surprise us terribly – there are a few places in the tag sequence where we would not expect the tags to combine (specifically, between verbal arguments). Fortunately, Propbank also tells us where to expect these rifts in the tag sequence.<sup>13</sup> Recall that we are assuming that Propbank gives us spans that cover groups of words. We can determine, then, the indexes where one argument meets another. For example, in the

<sup>13</sup>These rifts are similar to, but not the same as, those of Berger et al. [1996], who use a similar idea to assist in machine translation.

sentence above, there is one numbered argument that covers *a flower* and another that covers *to the policeman*. To allow the PHMM to incorporate these judgements, we can insert a dummy observation in the spaces between the semantic arguments. We initialize the transition probabilities into and out of this dummy state (which we will refer to as in the same way that we initialize the starting and ending probabilities. For each category  $c_n \in C$ , we define the transition probabilities for incoming and outgoing arcs as follows:

$$T\{RIFT, c_n\} = \frac{hasNoLeft(c)}{\sum_{c_n \in C} hasNoLeft(c_n)} \quad (5.5)$$

$$T\{c_n, RIFT\} = \frac{hasNoRight(c)}{\sum_{c_n \in C} hasNoRight(c_n)} \quad (5.6)$$

With the dummy observations incorporated for the tagging, we get the following result:

Words	Predicted Category
the	NP/N
man	N
gave	((S[dcl]\NP)/PP)/NP
a	NP/N
flower	N
RIFT	<b>RIFT</b>
to	PP/NP
the	NP/N
policeman	N
RIFT	<b>RIFT</b>
yesterday	(S\NP)\(S\NP)

Before passing the tag sequence to the parser, of course, the dummy RIFT tags can be removed. We can see now that the tag sequence generated by the model will result in a spanning analysis. Using this approach on section 00 of the WSJ corpus generates supertag sequences that result in a spanning analysis in about 37% of sentences.

## 5.4 Predicting Syntactic Structure

The next step is to predict some kind of syntactic structure on the supertags. In Boxwell et al. [2011, section 4], we showed that, given a set of high-quality supertags, we can oftentimes produce a very reasonable syntactic analysis using a very simple parse model (table 5.1). Once we have syntactic structure, then we can proceed to develop our semantic role labeling models. The simplest option is to take the supertags that we generate using the previously described method and parse the sentences automatically. Much of the work has been done for us by the previous step – the attachment decisions are, in many cases, already made. Doing so produces a spanning analysis in the training data about 20% of the time<sup>14</sup>. Nonetheless, we will require an automatic parser to generate syntactic analyses. A standard parsing algorithm (CKY) can be used to combine the CCG categories, but how will we choose which analysis to use in the event of ambiguity? Recall that we have no treebank on which to train any kind of parse model. All we have are the lexical categories generated in the previous section.

### 5.4.1 Experiment 1: Unguided Parsing

Boxwell et al. [2010] uses a baseline CCG parse model that is based on the idea that non-application operations (like type raising, composition, and substitution) should be used only when absolutely necessary. Whenever a new edge is added to the chart, it is assigned a score by the parse model. If the new edge was created by function application, coordination, or punctuation attachment, then it is assigned no penalty. If it was created by composition, type raising, or a special unary type-changing rule,

<sup>14</sup>Using the supertags to train a maximum entropy supertagging model can improve coverage to around 40%, but at the cost of accuracy.

Combinator	Penalty
Function Application	0
Function Composition	1
Crossing Composition	1
Type Raising	1
Null Coordination	2
Full Coordination	0
Substitution	$\infty$

Table 5.1: The complete baseline model, which requires no syntactic training data. The substitution combinator is used to model parasitic gaps in English, which are so rare that we make the pragmatic decision to disallow substitution entirely.

then it receives a penalty according to table 5.1. The penalties are cumulative – if a particular subtree uses type-raising once and composition twice, then it has a penalty of three. The spanning analysis that has the lowest penalty is the single-best analysis. In the event of a tie, the more right-branching analysis is chosen. Although the performance of this model is below the state of the art, it has the upshot of requiring no syntactic training data.

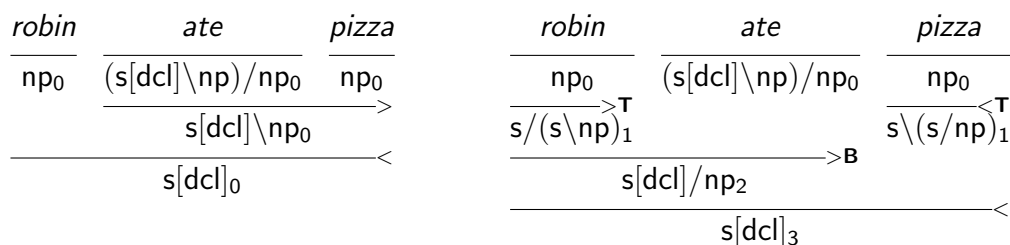


Figure 5.1: Two possible analyses for a simple sentence. The numbers in subscript indicate the penalty assigned to that edge. The baseline model assigns a penalty of 0 to the normal form parse (on the left), and a penalty of 3 to the non-normal form parse (on the right).

	Identification			ID + Classification		
	P	R	F	P	R	F
Gold Parse	96.0%	87.1%	91.3%	89.8%	81.4%	85.4%
Unguided Parse	94.4%	82.4%	88.0%	84.5%	73.8%	78.8%

Table 5.2: The performance of the unguided parse method for training SRL models, compared to models trained from gold-standard parses, on tagging gold standard analyses from the development set. The columns marked *Identification* indicate cases where the role boundaries are correct, whether or not the role label (Arg0, Arg1, ArgM-TMP, etc) is correct. The columns marked *ID+Classification* indicate cases where the span and label are both correct.

Now, once we have parsed the sentences of the training set (02-19) using the categories generated in section 5.3.5, we train the semantic role labeler over the sentences which have complete analyses (we will examine other options in section 5.4.3). In order to evaluate the SRL models, we will test them (for now) on the gold standard derivations in section 00 of the CCGbank. This will help us evaluate the quality of the SRL models while temporarily setting aside issues of the quality and coverage of the analyses of the candidate sentences. The performance of this method, which we will call the *unguided parse* method, is shown compared to SRL models trained on gold-standard parses in table 5.2.

## 5.4.2 Experiment 2: Constituent Guided Parsing

The unguided parsing method leaves much to be desired, both in performance and elegance. One of its major deficiencies is that it ignores the information that enabled us to generate high quality supertags – the Propbank annotation. Recall that we were able to identify rifts in the sequence where we do not expect the categories to

combine by any means other than highly aggressive type-raising. The same information could be used to identify likely constituents. Consider the phrase *the unit of the company that makes cigarettes* (wsj\_0003.3). This sentence is syntactically and semantically ambiguous – the two possible analyses are shown in figure 5.2. Both are valid analyses – the first analysis suggests that there is one unit that makes cigarettes in the company, and there are other units devoted to making other products (perhaps cigars or pipes). The second analysis suggests that the company as a whole makes cigarettes, and that the unit we are speaking of is only part of that company (perhaps HR or marketing). How will we choose which analysis to use? Fortunately, Propbank is designed to give us insight into the meaning of the sentence. In this case, Propbank annotates *the unit of the company* as Arg0 (Arg0 is glossed as the *creator* role in the Propbank frames files ). This clearly indicates that the first analysis is preferable.

In almost every case, the roles marked by Propbank correspond to a traditional constituent. In order for our semantic role labeler to correctly identify a role in a given analysis, that analysis must contain a constituent covering that span exactly. That is to say, if a derivation does not contain a constituent for the span 0-2, then it will never predict a role covering 0-2. It is therefore crucial that we generate analyses that accurately reflect the true meaning of the sentence, as these analyses will be used to train both a parse models and SRL models. To accomplish this, we will define a function to determine if a candidate constituent is in violation of known role boundaries. In simple terms, a constituent is in violation if it overlaps partially (but not entirely) with a semantic role. Constituents may completely cover or be covered completely by roles, but they may not partially overlap them. Figure 5.3 shows how the constituent *the company that makes cigarettes* is incompatible with the role

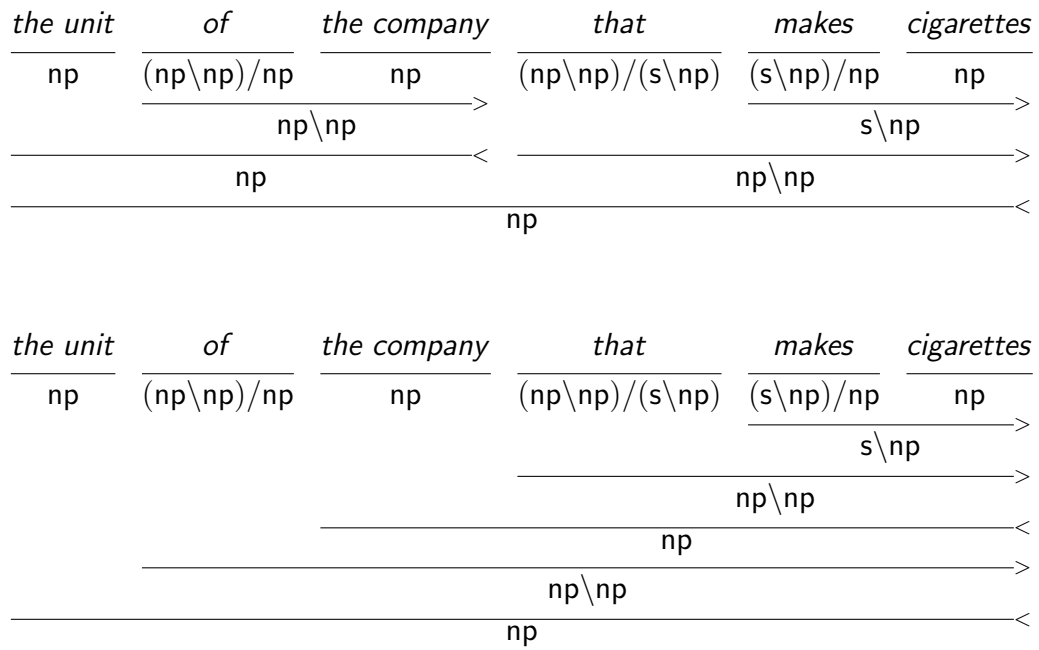


Figure 5.2: Two possible analyses for the phrase *the unit of the company that makes cigarettes*. According to the CCGbank, the first analysis is correct, but both analyses yield reasonable semantics.

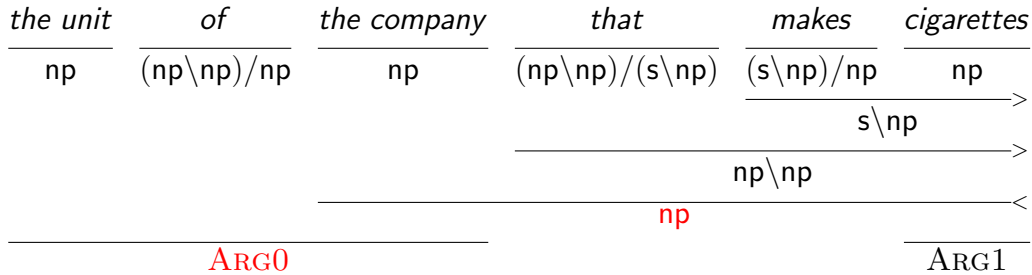


Figure 5.3: The NP *the company that makes cigarettes* is in violation of the role boundaries of Arg0. Therefore, it should not be included in the final analysis.

	Identification			ID + Classification		
	P	R	F	P	R	F
Gold Parse	96.0%	87.1%	91.3%	89.8%	81.4%	85.4%
Unguided Parse	94.4%	82.4%	88.0%	84.5%	73.8%	78.8%
Constituent Guided Parse	94.3%	84.2%	88.9%	84.6%	75.4%	<b>79.7%</b>

Table 5.3: The performance of the guided parse method for training SRL models, compared to the previous models. Enforcing role boundaries on the training sentences improves SRL performance by about 1%.

covering *the unit of the company*. By introducing a parse model that penalizes parses for violating role boundaries, we can generate better training data, which in turn can be used to generate SRL models that improve identification and classification by about 1%. The performance of these SRL models is compared with that of the previous models in table 5.3.

### 5.4.3 Experiment 3: Training Semantic Models over a Packed Chart

The SRL training approaches described in the previous sections have the advantage that they fit relatively closely with traditional notions of how SRL models should

be trained – first we take some syntax trees (whether they are gold-standard or generated somehow), we train our semantic model over these complete trees, then we use that semantic model to make predictions on previously unseen trees. The disadvantage of this approach is that we must commit ourselves to a single analysis for each training sentence. This is not a problem if we have gold-standard data, but we wish to generate the trees that we are training over. The guidance of the model in section 5.4.2 certainly helps, but the loose matrix that our trees grow on still leaves room for error, and we have to throw away any training sentence for which we do not find a spanning analysis.

To address these problems, we borrow and adapt an approach from Boxwell et al. [2010] to train a semantic model not over a single analysis of a sentence, but over many possible analyses simultaneously in the form of a packed parse chart. For each sentence, we allow the categories to fill the chart as if we were parsing, but instead of choosing a single-best parse, we train over every possible constituent in the chart that is consistent with the constraints from section 5.4.2. We prevent the grammar from using any unary rules besides the most common ( $N \rightarrow NP$ ,  $S[\text{pss}] \setminus NP \rightarrow NP \setminus NP$ ), as they will overgenerate and pollute the data. We’re unconcerned if we cannot find a spanning analysis – we don’t need one to train over the edges we can generate in the chart. It’s not hard to see a potential advantage to this approach, especially if we expect our supertag data to be noisy. Consider the sentence *Robin gave Leslie the books, right?* A small supertagging error can prevent us from generating a complete analysis, which in turn prevents the method from section 5.4.2 from obtaining any training data whatsoever from the sentence. With this method, however, we can

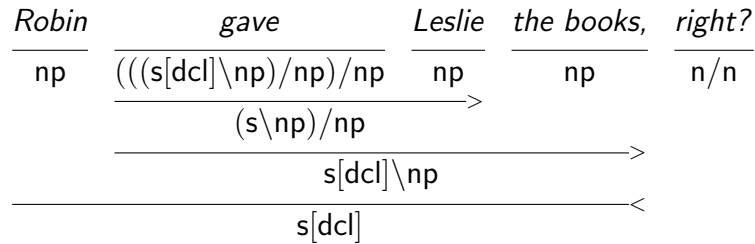


Figure 5.4: A supertagging error in this sentence prevents us from producing a spanning analysis. Fortunately, we can still get useful information from the edges that we can generate.

recover from a poorly assigned supertag and still extract useful data, especially if the erroneous supertag is somewhat peripheral to the overall meaning of the sentence.

Unfortunately, we do not find the same improved performance that was reported in Boxwell et al. [2010] with our Propbank-generated supertag data. In fact, the single-best methods both outperform the chart-based method. Weighting the features that are fed to the semantic role labeler does not seem to help – using the inside-outside algorithm to re-estimate the weights only hurts performance, and using weights based on a PCFG generated from the CCGbank (that is, cheating) still generates lower accuracy rates than weighting all features equally.

We suspect that the reduced performance compared to the model from section 5.4.2 is because the data in Boxwell et al. [2010] is substantially noisier because it is not based on the Propbank-supported supertag assignment (section 5.3). Because we are better able to predict supertags in our experiment (especially for the all-important verbal categories), the advantage of using a chart-based approach is greatly reduced. We suspect that if we were using training data of a different genre, or perhaps data with a higher incidence of disfluencies, then this approach would produce better results

	Identification			ID + Classification		
	P	R	F	P	R	F
Gold Parse	96.0%	87.1%	91.3%	89.8%	81.4%	85.4%
Unguided Parse	94.4%	82.4%	88.0%	84.5%	73.8%	78.8%
Constituent Guided Parse	94.3%	84.2%	88.9%	84.6%	75.4%	79.7%
Packed Chart Parse	96%	81.9%	88.7%	84.1%	71.2%	<b>77.1%</b>

Table 5.4: The performance of the guided parse method for training SRL models, compared to the previous models. Enforcing role boundaries on the training sentences improves SRL performance by about 1%.

than the approach described in section 5.4.2. For this reason, we will set the chart-based approach aside for a moment, and proceed assuming a single-best parse training structure. We hope to investigate the chart-based approach for disfluent texts further in future work.

## 5.5 Future Work

Ravi et al. [2010] use Integer Linear Programming to improve the selection of supertags given only a tag dictionary. We do not use any kind of ILP approach here – we rely on Propbank-inspired categories to generate high accuracy rates on supertag prediction. Combining these approaches, however, has the potential to improve performance even further. Improving the generation of verbal tags (perhaps relying on the verb classes in Levin [1993]) is another likely avenue of research. Finally, further experimentation with the Baum-Welch algorithm may yield even better supertagging results – experimenting with random start states may help avoid local maxima. Another avenue of future work is improving the primitive parser-supertagger interface (state-of-the-art parsers often adapt their lexical categories at parse time, see Clark

and Curran [2004a]). We could also realize better performance though loopy belief propagation for parsing and supertagging, a technique that has been shown to be successful for overcoming problems of coverage with CCG by preventing the supertagger from pruning analyses that will be necessary to form a spanning analysis [Auli and Lopez, 2011]. This could help improve the coverage of the parser over the training data.

Another area for further investigation is the training of semantic resources on less-structured texts. Newswire is very well structured – it is written by professional writers and checked by professional editors. Less structured text (say, text messages or posts to social networking sites) is plentiful. Natural language resources for such text are in high demand. The packed-chart method of training semantic role labeling models described in section 5.4.3 was less effective than the full-parse method of section 5.4.2 with newswire, but it may outperform the full-parse model on less structured text where sentence segmentation is guesswork and the likelihood of finding a spanning analysis is low.

One final area of potential improvement lies in using the inside-outside algorithm to do expectation-maximization on the parse model in order to generate a better corpus to train over. We found that running the inside-outside algorithm with a simple maximum-likelihood PCFG did not ultimately improve SRL performance on gold standard or automatically generated parses – (even though the likelihood of the training data increased, performance very gradually declined on unseen data at every iteration). EM is often most effective when it is given a good starting point and some restrictions on its movements. One possible explanation is that our oracle Propbank information is restricting the EM too much, and that the only changes that the model

can make are inconsequential. Another possibility is that a more sophisticated EM approach could be beneficial – incorporating lexical head features has the potential to improve EM performance. For now, however, we will set the EM approach aside, and instead rely on the split-merge approach of the Berkeley parser to generate high quality test parses. We will describe that method in chapter 6.

## 5.6 Conclusion

In this chapter, we show how human-annotated semantic roles can guide the selection of CCG lexical categories. We show how these selections can be used to guide a Partially Hidden Markov Model and achieve better results. This demonstrates the importance of using CCG as our syntactic formalism, as the Partially Hidden Markov Model must be initialized using the rules of CCG combinatorics (which would be impossible with a traditional part-of-speech tagset). We also demonstrated how the same semantic roles that guided our lexical category choices can help us assemble entire analyses for sentences. In the next chapter, we will describe experiments in using this newly-generated treebank to train an off-the-shelf parser and produce novel analyses for new sentences, then use the SRL models described in this chapter to produce semantic predictions on previously unseen text.

## CHAPTER 6: Generating Syntactic Analyses for Novel Text

In chapter 5, we discussed a method for generating CCG supertags for syntactic training data. The method described relies on having high quality Propbank data and has poor coverage. When a spanning analysis is produced, however, the quality of the analysis is generally very high. The next step is to use these high quality analyses as training data to produce syntactic analyses for previously unseen text that does not have a propbank annotation.

### 6.1 Experiment 1: Training an HMM Supertag Model

The simplest solution is to simply take the HMM that we used in section 5.3.5 – the model whose transition probabilities are based solely on the rules of CCG combinatorics. We will call this MODEL 0. The next simplest option is to re-train an HMM over the predictions of MODEL 0 on the training data. We will call this MODEL 1. The next simple option is to re-train the HMM over the predictions of MODEL 0, but only use the sentences whose tags produce spanning analyses. This will sharply reduce the amount of training data available, but it will be of arguably higher quality. We will call this MODEL 2. The comparison of the performance on these models on the development data is shown in table 6.1.

The performance of the baseline models is consistent with expectations. MODEL 0 performs reasonably well on supertag accuracy, but it is necessary to make many more guesses per word (almost 5 tags per word, on average). The parse coverage is

	Tag Accuracy	Tag/Word	Sentence Coverage	SRL ID			SRL ID+CL		
				P	R	F	P	R	F
MODEL 0	80.3	4.8	<b>47.3</b>	65.6	53.9	59.2	57.6	47.3	52.0
MODEL 1	83.1	2.1	<b>44.6</b>	75.2	62.7	68.4	68.4	57.0	62.2
MODEL 2	82.1	1.5	<b>60.5</b>	75.1	63.0	68.5	68.3	57.3	62.3

Table 6.1: The performance of the three baseline supertag models, using a beta value of 0.1. Even the best baseline model achieves only 60% sentence coverage, which is unacceptably low.

disappointing, coming in at under 50%, and the parse accuracy is abysmal. MODEL 1 gains parse accuracy and improves the tag-per-word average at the cost of coverage. MODEL 2, which is trained on sentences for which a spanning analysis is found, greatly improves coverage, but does not substantially improve parse accuracy. Nonetheless, even the 60% coverage figure is unacceptably low for real-world purposes. It will be necessary improve coverage substantially, or our corpus will not be useful.

## 6.2 Experiment 2: Using the Berkeley Parser to Generate High Quality Analyses

The major problem with the three baseline models in section 6.1 is their unacceptably low sentence coverage. Even if they were to achieve very high SRL accuracy on the sentences for which we have analyses, it would be difficult to compare to other labelers. For this reason, we need a way to improve sentence coverage using only our constructed corpus. One option would be to build a tighter connection between our parser and a supertagger, incorporating an efficient but accurate adaptive supertagging step. Alternately, we could find a high quality parser that is publicly available,

easily usable on a variety of platforms, and easy to train on a new corpus. For these reasons, we chose to use the Berkeley Parser.

### 6.2.1 The Berkeley Parser

The Berkeley parser [Petrov et al., 2006, Petrov and Klein, 2007] is a high quality syntactic parser that generates grammars by splitting and merging nonterminal symbols. The Berkeley parser may be a surprising choice for us, as it is not a CCG parser – it is designed to be used with traditional PTB-style trees – but previous work has demonstrated that the Berkeley parser can be used effectively with CCG [Fowler and Penn, 2010] and CCG-like grammars (Nguyen and Schuler, forthcoming).

The Berkeley parser can be trained over an arbitrary treebank. Instead of simply gathering up statistics on the relative likelihoods of rewrite rules, however, the Berkeley parser divides the nonterminal symbols into different sub-symbols, representing different distributions. For example, NPs that fall in the subject position of a sentence are often very different from the NPs that fall in the object position – it makes sense for the two to be split into two categories for parsing purposes (calling them NP-1 and NP-2, perhaps), and collapsing them together only for evaluation. This method and others like it have been used to great effect to improve parsing performance with unlexicalized grammars [Johnson, 1998, Klein and Manning, 2003]. The process of manually splitting certain categories can also be replaced by an automatic splitting of certain common categories [Prescher, 2005, Matsuzaki et al., 2005]. Petrov et al. [2006] distinguish themselves by incorporating the ability to merge sub-categories back together, enabling the system to split certain symbols (like NNP, JJ, and VBN) into many different sub-categories, while other symbols (like WP, POS, and EX) are

- $NP \rightarrow NP/N + N$
- $PP \rightarrow PP/NP + NP$
- $S \backslash NP \rightarrow (S \backslash NP)/NP + NP$

Figure 6.1: A selection of CFG-style rules learned by the Berkeley parser when trained on the CCG derivations. The Berkeley parser treats the categories as atomic, and therefore does not attempt to make any generalizations about forward application.

split into few sub-categories. The Berkeley parser claims an F-score of 90.2% on labeled bracketings on the Penn Treebank, higher than any other system at the time of publication without resorting to re-ranking.

### 6.2.2 Using the Berkeley Parser as a CCG Parser

Although traditional syntax trees and CCG derivations are different in many ways, it is not difficult to use the Berkeley Parser to generate CCG-style derivations. CCG derivations can be forced into the traditional parenthesis-based tree format (figure 6.2.2), then handed to the Berkeley parser for training. The Berkeley parser does not recognize the complex nature of CCG categories – a very complex category like  $(NP \backslash NP)/(S \backslash NP)$  is treated as a single, atomic category. Similarly, the Berkeley parser does not recognize any of the rules of CCG combinatorics – it simply recognizes the individual instantiations of these rules. That is to say, the Berkeley parser will learn the rules in figure 6.1, but it will not recognize the general principle of forward application. This matters little for our present purposes – we can always derive the syntactic dependencies from the complete parses later.

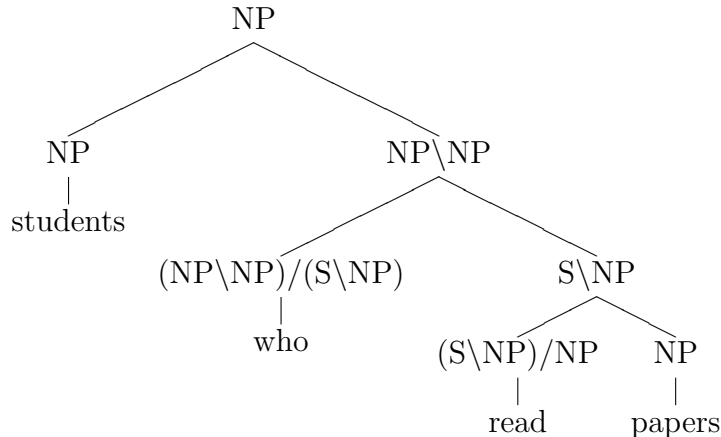


Figure 6.2: CCG derivations must be represented as PTB-style trees before they are given to the Berkeley parser.

	Sentence Coverage	SRL ID			SRL ID+CL		
		P	R	F	P	R	F
Gold	99.8%	87.4%	76.3%	81.5%	79.3%	69.3%	<b>74.0%</b>
Generated	99.2%	77.6%	65.9%	71.3%	71.3%	60.6%	<b>65.5%</b>
Chart	99.2%	80.7%	64.1%	71.4%	73.6%	58.4%	<b>65.1%</b>

Table 6.2: The performance of the Berkeley parser on CCG-style analyses when trained on gold-standard CCG parses, our smaller generated treebank from section 5.4.2, and the chart-based approach from section 5.4.3.

Our first step is to determine how effective the Berkeley parser is at generating CCG-style analyses apart from other factors. In order to measure this, we convert the English CCGbank into PTB-form and train the Berkeley parser over it. This will give us a reasonable upper-bound on the performance of our generated system. The performance of a Berkeley parser model trained on our generated CCGbank is compared to the performance of a Berkeley parser model trained on the gold-standard CCGbank in table 6.2. Performance on the test set is shown in table 6.4.

	Sentence Coverage	SRL ID			SRL ID+CL		
		P	R	F	P	R	F
Gold	99.9%	86.7%	74.8%	80.3%	78.4%	67.6%	<b>72.6%</b>
Generated	99.5%	76.5%	64.2%	69.8%	70.5%	59.2%	<b>64.3%</b>
Chart	99.5%	79.6%	62.3%	69.9%	72.4%	56.6%	<b>63.5%</b>

Table 6.3: The performance of the models from table 6.2 on the test set (WSJ section 23).

It is unsurprising that the performance of the parser trained on the generated treebank is lower than that of the performance of the parser trained on the entire gold standard treebank. There are two reasons for this: data quality and data quantity. The generated model is trained on analyses that are induced from Propbank annotations – they are bound to be noisy. Furthermore, the generated model is trained on fewer analyses than the gold model – only about 30% of the sentences in the generated training set have spanning analyses. Both of these factors cause decreased performance. In order to separate out these two effects, we train a third model (which we will call the SUBGOLD model). This model is trained over gold standard analyses from the CCGbank, but includes *only the sentences that are included in the generated model*. That is, the same sentences as the generated model are used, but we train over the gold standard analyses instead of the generated ones. This will show us how much of an impact the reduced training set size is having, apart from the noise introduced by the Propbank-based induction process. Using a reduced-size training set results in a 3% drop in SRL performance from that of the full gold model. The complete results are shown in table 6.4.

	Sentence Coverage	SRL ID			SRL ID+CL		
		P	R	F	P	R	F
Gold	99.8%	87.4%	76.3%	81.5%	79.3%	69.3%	74.0%
Generated	99.2%	77.6%	65.9%	71.3%	71.3%	60.6%	65.5%
Subgold	98.9%	84.8%	72.8%	78.4%	76.9%	65.9%	<b>71.1%</b>

Table 6.4: The effect of the reduced training set size is demonstrated by the SUBGOLD model. Using an induced model instead of gold-standard data accounts for about 5.5% of the difference in performance.

### 6.3 Error Analysis

It is instructive to examine in detail the errors produced by the semantic model that was trained on the generated treebank. We will focus on recall errors that were made by the generated model but not the SUBGOLD model described in section 6.2.2. This will allow us to see clearly the cases where the generated model is outperformed by the model trained on gold standard data, while ignoring differences caused by a larger quantity of training data. One common problem has to do with the category  $S[\text{adj}]\backslash\text{NP}$ . This category is rarely assigned to Propbank predicates – it and its cousins<sup>15</sup> are usually reserved for predicate adjectives like *John is **angry***. The verbal category prediction process (described in section 5.3.1) never predicts  $S[\text{adj}]$  as the base of a predicate because it very rare and, when it does appear, is very difficult to distinguish from  $S[\text{pss}]$  (passive voice). However, the Berkeley parser sometimes assigns this category to predicate categories. This is not fatal to the parse – both the categories  $S[\text{adj}]\backslash\text{NP}$  and  $S[\text{pss}]\backslash\text{NP}$  are often type-changed to  $\text{NP}\backslash\text{NP}$ . The problem, however, arises when these categories are attached to syntactic adjuncts. Consider

<sup>15</sup> $(S[\text{adj}]\backslash\text{NP})/\text{NP}$ ,  $(S[\text{adj}]\backslash\text{NP})/\text{PP}$ , etc.

the analysis in figure 6.3. Because  $(S[\text{adj}]\backslash\text{NP})/(S[\text{adj}]\backslash\text{NP})$  doesn't appear in the generated training set with a semantic role, the model trained over it fails to identify *still* as ArgM-TMP.

## 6.4 Future Work

The Berkeley Parser is a very high quality parser and has been shown to be effective even when parsing in CCG [Fowler and Penn, 2010] or a CCG-like grammar (Nguyen and Schuler, forthcoming). Nevertheless, it would be instructive to try this same approach using a state-of-the-art CCG parser like that of Clark and Curran [2004b], which was not done here due to the resource-intensive training process [Clark and Curran, 2004b, section 6]. Examining the differences in the errors produced by the Berkeley Parser and a CCG parser would be instructive.

## 6.5 Conclusion

In this chapter, we described how to use an off-the-shelf parser to produce high-quality automatically generated parses on previously unseen text, then used the semantic models from chapter 5 to make semantic predictions on those analyses. In the future, we would like to experiment further with k-best parse results from the Berkeley parser, using joint inference to combine the predictions into a cohesive whole.

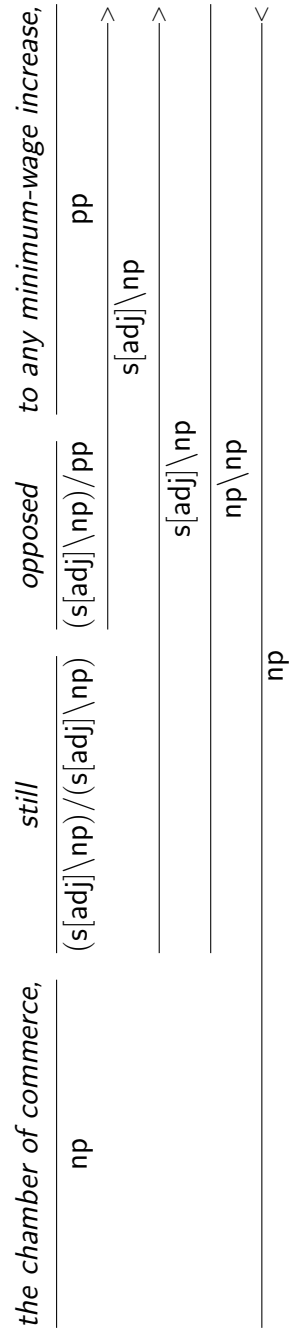


Figure 6.3: The Berkeley parser makes the decision to assign (S[adj]\NP)/NP to *opposed*. Because the verbal category prediction process never assigns S[adj] as the base of a verbal category, the (S[adj]\NP)/(S[adj]\NP) category is unlikely to be chosen for a semantic role.

## CHAPTER 7: Conclusion

This dissertation addressed the problem of developing a complete semantic role labeling system in the absence of syntactic training data. We considered the potential of Propbank semantic role information to predict syntactic structure, both in the form of CCG supertags and the complete syntactic analysis that they form.

### 7.1 Semantic Role Labeling in CCG

We first described a system for predicting semantic role labels for English, closely related to the system described in Boxwell et al. [2009]. This system is deliberately constructed to use only local features, so as to enable it to train over incomplete parses. It first identifies and categorizes predicates, then it identifies and categorizes semantic roles. It can predict roles either on single headwords or complete constituents – for the purposes of this dissertation, we care more about the constituent measure.

### 7.2 Generating CCG Supertags

We next describe a method adapted from Baldrige [2008] and Ravi et al. [2010] to generate lexical categories for a treebank by using Propbank semantic role annotation to guide the selection of verbal and argument categories. This method has the advantage of informing some of the most ambiguous categories, which leads to greatly improved coverage and accuracy.

### 7.3 Generating a Treebank

Finally, we describe a method to construct a treebank using the lexical categories that obeys the constraints laid out by the Propbank annotation. This treebank can be used to train both syntactic and semantic models that make high-quality semantic role predictions, using a very small amount of handcrafting, without the expense of developing a treebank. The handcrafting is limited to a POS-to-CCG tag dictionary, a small word-to-CCG tag dictionary for certain closed class words (Appendix A), and some language specific knowledge about relative clauses and other non-standard word orders for the prediction of verbal categories. Ultimately, this shows that if world circumstances would seem to merit the development of a syntactic treebank for a new critical language, it would be wise to begin with a semantic corpus like Propbank, so that these judgements can be used to automatically produce a syntactic treebank with which to train syntactic and semantic models.

### 7.4 Future Work

There are a variety of directions where the work presented in this dissertation could be taken. One likely direction would be the development of treebanks for languages other than English. Oftentimes it is necessary for governments or international aid organizations to begin activities in a new area on very short notice. Being able to develop natural language resources rapidly for an under-studied language would be of great value, even if the performance of these systems was below that which could be achieved with a large-scale professionally annotated treebank. The treebank generation method described here could even be used to produce a first-pass treebank, which could then be corrected by hand. An analysis of the speed and cost of generating

treebanks in this way would be enlightening, and may motivate the production of Propbank-like annotations of semantic roles for under-studied languages.

## BIBLIOGRAPHY

- M. Auli and A. Lopez. A comparison of loopy belief propagation and dual decomposition for integrated ccg supertagging and parsing. In *Proc. of ACL*, 2011.
- O. Babko-Malaya. Propbank annotation guidelines. 2005.
- C.F. Baker, C.J. Fillmore, and J.B. Lowe. The berkeley framenet project. In *Proceedings of the 17th international conference on Computational linguistics-Volume 1*, pages 86–90. Association for Computational Linguistics, 1998.
- J. Baldridge. Weakly supervised supertagging with grammar-informed initialization. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 57–64. Association for Computational Linguistics, 2008.
- S. Bangalore and A.K. Joshi. Supertagging: An approach to almost parsing. *Computational linguistics*, 25(2):237–265, 1999.
- A.L. Berger, V.J.D. Pietra, and S.A.D. Pietra. A maximum entropy approach to natural language processing. *Computational linguistics*, 22(1):39–71, 1996.
- A. Bies, M. Ferguson, K. Katz, R. MacIntyre, V. Tredinnick, G. Kim, M.A. Marcinkiewicz, and B. Schasberger. Bracketing guidelines for treebank ii style penn treebank project. Technical report, Technical report, University of Pennsylvania, 1995.

- Stephen A Boxwell, Chris Brew, Jason Baldridge, Dennis N Mehay, and Sujith Ravi. Semantic role labeling for ccg without treebanks? In *Proceedings of the 2011 International Joint Conference on Natural Language Processing*, Chiang Mai, Thailand, November 2011.
- Stephen A. Boxwell, Dennis N. Mehay, and Chris Brew. Brutus: A semantic role labeling system incorporating CCG, CFG, and Dependency features. In *Proc. ACL-09*, 2009.
- Stephen A Boxwell, Dennis N Mehay, and Chris Brew. What a parser can learn from a semantic role labeler and vice versa. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 736–744, Cambridge, MA, October 2010. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/D/D10/D10-1072>.
- Stephen A. Boxwell and Michael White. Projecting Propbank Roles onto the CCG-bank. In *Proceedings of the Sixth International Language Resources and Evaluation Conference (LREC-08)*, Marrakech, Morocco, 2008.
- Adriane Boyd, Markus Dickinson, and Detmar Meurers. Increasing the recall of corpus annotation error detection. In *Proceedings of the Sixth Workshop on Treebanks and Linguistic Theories (TLT 2007)*, pages 19–30, Bergen, Norway, 2007. URL <http://jones.ling.indiana.edu/~mdickinson/papers/boyd-et-al-07b.html>.
- E. Charniak. Immediate-head parsing for language models. In *Proc. ACL-01*, volume 39, pages 116–123, 2001.

- J. Chen and Vijay-Shanker. Automated extraction of TAGs from the Penn Treebank. *New developments in parsing technology*, pages 73–89, 2004.
- S. Clark and J.R. Curran. The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of COLING*, volume 4, pages 282–288, 2004a.
- Stephen Clark and James R. Curran. Parsing the WSJ using CCG and Log-Linear Models. In *Proc. ACL-04*, 2004b.
- M. Collins. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4):589–637, 2003.
- Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.
- Markus Dickinson. *Error detection and correction in annotated corpora*. PhD thesis, The Ohio State University, 2005. URL <http://ling.osu.edu/~dickinso/papers/diss/>.
- Markus Dickinson and W. Detmar Meurers. Detecting errors in part-of-speech annotation. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL-03)*, pages 107–114, Budapest, Hungary, 2003. URL <http://jones.ling.indiana.edu/~mdickinson/papers/dickinson-meurers-03.html>.
- Markus Dickinson and W. Detmar Meurers. Detecting errors in discontinuous structural annotation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL-05)*, pages 322–329, Ann Arbor,

- MI, USA, 2005. URL <http://jones.ling.indiana.edu/~mdickinson/papers/dickinson-meurers-05.html>.
- D. Dowty. Thematic proto-roles and argument selection. *Language*, 67(3):547–619, 1991. ISSN 0097-8507.
- Jason Eisner. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 340–345, Copenhagen, August 1996.
- T.A.D. Fowler and G. Penn. Accurate context-free parsing with combinatory categorial grammar. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 335–344. Association for Computational Linguistics, 2010.
- H. Fürstenau and M. Lapata. Graph alignment for semi-supervised semantic role labeling. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 11–20. Association for Computational Linguistics, 2009.
- D. Gildea and D. Jurafsky. Automatic labeling of semantic roles. *Computational Linguistics*, 28(3):245–288, 2002.
- D. Gildea and M. Palmer. The necessity of parsing for predicate argument recognition. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 239–246. Association for Computational Linguistics, 2002.
- Daniel Gildea and Julia Hockenmaier. Identifying semantic roles using Combinatory Categorical Grammar. In *Proc. EMNLP-03*, 2003.

- C.T. Hemphill, J.J. Godfrey, and G.R. Doddington. The atis spoken language systems pilot corpus. In *Proceedings of the DARPA speech and natural language workshop*, pages 96–101, 1990.
- J. Henderson, P. Merlo, G. Musillo, and I. Titov. A latent variable model of synchronous parsing for syntactic and semantic dependencies. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 178–182. Association for Computational Linguistics, 2008.
- J. Hockenmaier and M. Steedman. CCGbank manual. Technical report, MS-CIS-05-09, University of Pennsylvania, 2005.
- Julia Hockenmaier and Mark Steedman. CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396, 2007.
- M. Honnibal and J.R. Curran. Improving the complement/adjunct distinction in CCGbank. In *Proceedings of the 10th Conference of the Pacific Association for Computational Linguistics (PACLING-07)*, pages 210–217. Citeseer, 2007.
- M. Honnibal, J.R. Curran, and J. Bos. Rebanking ccgbank for improved np interpretation. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 207–215. Association for Computational Linguistics, 2010.
- M. Johnson. Pcfg models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632, 1998.

- D. Jurafsky, J.H. Martin, A. Kehler, K. Vander Linden, and N. Ward. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*, volume 163. MIT Press, 2000.
- T. Kasami. An efficient recognition and syntax analysis algorithm for context-free languages. Technical report, DTIC Document, 1965.
- Martin Kay. *Experiments with a Powerful Parser*. RAND Corporation, 1967.
- D. Klein and C.D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics, 2003.
- D. Klein and C.D. Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, pages 478–es. Association for Computational Linguistics, 2004.
- J. Lang and M. Lapata. Unsupervised induction of semantic roles. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 939–947. Association for Computational Linguistics, 2010. ISBN 1932432655.
- J. Lang and M. Lapata. Unsupervised semantic role induction with graph partitioning. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1320–1331, 2011.
- B. Levin. *English verb classes and alternations: A preliminary investigation*, volume 348. University of Chicago press Chicago, IL, 1993.

- X. Lluís and L. Màrquez. A joint model for parsing syntactic and semantic dependencies. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 188–192. Association for Computational Linguistics, 2008.
- C.D. Manning, H. Schütze, and MITCogNet. *Foundations of statistical natural language processing*, volume 59. MIT Press, 1999.
- M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.
- T. Matsuzaki, Y. Miyao, and J. Tsujii. Probabilistic cfg with latent annotations. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 75–82. Association for Computational Linguistics, 2005.
- A. Meyers, R. Reeves, C. Macleod, R. Szekely, V. Zielinska, B. Young, and R. Grishman. The nombank project: An interim report. In A. Meyers, editor, *HLT-NAACL 2004 Workshop: Frontiers in Corpus Annotation*, pages 24–31, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics.
- Martha Palmer, Daniel Gildea, and Paul Kingsbury. The Proposition Bank: An Annotated Corpus of Semantic Roles. *Computational Linguistics*, 31(1):71–106, 2005.
- M.A. Paskin. Grammatical Digrams. In *Advances in neural information processing systems 14: proceedings of the 2001 conference*, page 91. MIT Press, 2002. ISBN 0262042061.

- S. Petrov, L. Barrett, R. Thibaux, and D. Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 433–440. Association for Computational Linguistics, 2006.
- S. Petrov and D. Klein. Improved inference for unlexicalized parsing. In *Proceedings of NAACL HLT 2007*, pages 404–411, 2007.
- Sameer S. Pradhan, Wayne Ward, and James H. Martin. Towards Robust Semantic Role Labeling. *Computational Linguistics*, 34(2):289–310, 2008.
- D. Prescher. Inducing head-driven pcfgs with latent heads: Refining a tree-bank grammar for parsing. *Machine Learning: ECML 2005*, pages 292–304, 2005.
- Vasin Punyakanok, Dan Roth, and Wen tau Yih. The Importance of Syntactic Parsing and Inference in Semantic Role Labeling. *Computational Linguistics*, 34(2):257–287, 2008.
- L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989. ISSN 0018-9219.
- S. Ravi, J. Baldridge, and K. Knight. Minimized models and grammar-informed initialization for supertagging with highly ambiguous lexicons. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 495–503. Association for Computational Linguistics, 2010.
- D. Roth and W Yih. A linear programming formulation for global inference in natural language tasks. In *Proceedings of CoNLL-2004*, pages 1–8, 2004.

- Y. Samuelsson, O. Tackstrom, S. Velupillai, J. Eklund, M. Fisel, and M. Saers. Mixing and blending syntactic and semantic dependencies. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 248–252. Association for Computational Linguistics, 2008.
- K.K. Schuler. Verbnet: A broad-coverage, comprehensive verb lexicon. 2005.
- Mark Steedman. *The Syntactic Process*. MIT Press, 2000.
- W. Sun, H. Li, and Z. Sui. The integration of dependency relation classification and semantic role labeling using bilayer maximum entropy Markov models. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 243–247. Association for Computational Linguistics, 2008.
- M. Surdeanu, R. Johansson, A. Meyers, L. Màrquez, and J. Nivre. The CoNLL-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 159–177. Association for Computational Linguistics, 2008.
- D.H. Younger. Recognition and parsing of context-free languages in time  $n^3$ \*. *Information and control*, 10(2):189–208, 1967.
- L.S. Zettlemoyer and M. Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proc. of UAI*, volume 5. Citeseer, 2005.
- Y. Zhang and S. Clark. Shift-reduce ccg parsing. In *Proceedings of the 49th Meeting of the Association for Computational Linguistics*, pages 683–692, 2011.

## APPENDIX A: Common Closed-Class Words

the, a, an	NP <sub>1</sub> /N <sub>1</sub>
is, are, was, were	(S[dcl]\NP)/NP (S[dcl]\NP <sub>1</sub> )/(S[adj]\NP <sub>1</sub> ) (S[dcl]\NP <sub>1</sub> )/(S[ng]\NP <sub>1</sub> ) (S[dcl]\NP <sub>1</sub> )/(S[pss]\NP <sub>1</sub> )
did	(S[dcl]\NP <sub>1</sub> )/(S[b]\NP <sub>1</sub> )
had, have, has	(S[dcl]\NP)/NP (S[dcl]\NP <sub>1</sub> )/(S[pt]\NP <sub>1</sub> ) (S[dcl]\NP <sub>1</sub> )/(S[to]\NP <sub>1</sub> )
having	(S[ng]\NP)/NP (S[ng]\NP <sub>1</sub> )/(S[pt]\NP <sub>1</sub> ) (S[ng]\NP <sub>1</sub> )/(S[to]\NP <sub>1</sub> ) (S[ng]\NP <sub>1</sub> )/(S[pss]\NP <sub>1</sub> )
to	PP/NP (S[to]\NP <sub>1</sub> )/(S[b]\NP <sub>1</sub> ) ((S\NP <sub>1</sub> ) <sub>2</sub> \(S\NP <sub>1</sub> ) <sub>2</sub> )/NP
be	(S[b]\NP <sub>1</sub> )/(S[pss]\NP <sub>1</sub> ) (S[b]\NP <sub>1</sub> )/(S[adj]\NP <sub>1</sub> )
do, does	(S[dcl]\NP)/NP (S[dcl]\NP <sub>1</sub> )/(S[b]\NP <sub>1</sub> )
been	(S[pt]\NP <sub>1</sub> )/(S[pss]\NP <sub>1</sub> ) (S[pt]\NP <sub>1</sub> )/(S[ng]\NP <sub>1</sub> ) (S[pt]\NP)/PP (S[pt]\NP)/NP (S[pt]\NP <sub>1</sub> )/(S[adj]\NP <sub>1</sub> )
of	(NP <sub>1</sub> \NP <sub>1</sub> )/NP