UNSUPERVISED GRAMMAR INDUCTION WITH COMBINATORY CATEGORIAL GRAMMARS

BY

YONATAN YITZHAK BISK

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Doctoral Committee:

Associate Professor Julia Hockenmaier, Chair
Professor Jason Eisner, Johns Hopkins University
Professor Dan Roth
Professor ChengXiang Zhai

Abstract

Language is a highly structured medium for communication. An idea starts in the speaker's mind (semantics) and is transformed into a well formed, intelligible, sentence via the specific syntactic rules of a language. We aim to discover the fingerprints of this process in the choice and location of words used in the final utterance. What is unclear is how much of this latent process can be discovered from the linguistic signal alone and how much requires shared non-linguistic context, knowledge, or cues.

Unsupervised grammar induction is the task of analyzing strings in a language to discover the latent syntactic structure of the language without access to labeled training data. Successes in unsupervised grammar induction shed light on the amount of syntactic structure that is discoverable from raw or part-of-speech tagged text. In this thesis, we present a state-of-the-art grammar induction system based on Combinatory Categorial Grammars. Our choice of syntactic formalism enables the first *labeled* evaluation of an unsupervised system. This allows us to perform an in-depth analysis of the system's linguistic strengths and weaknesses. In order to completely eliminate reliance on any supervised systems, we also examine how performance is affected when we use induced word clusters instead of gold-standard POS tags. Finally, we perform a semantic evaluation of induced grammars, providing unique insights into future directions for unsupervised grammar induction systems.

Acknowledgements

Julia Hockenmaier has been the most dedicated advisor I could have ever asked for. She was always available when I needed help, and often offered it before I knew that I needed it. I sent her an email about my interest in grammar induction before I arrived at Illinois, and she helped me transform an interest into a passion and eventually this thesis.

When I joined UIUC, our research group was small, but the AI labs created an environment that grew to be my home and family. Dan Goldwasser and Rajhans Samdani sat patiently and supportively with me from day one. Ian Endres and Hannaneh Hajishirzi strong-armed Ali Farhadi into welcoming me to the Vision lab, where I would sit for the next six years.

With every year the face of the lab and Urbana changed. I'm not sure there are enough years in a lifetime to cover all the topics Amin Sadeghi and I want to discuss with each other. The arrival of Christos Christodoulopoulos to Illinois at least doubled the amount of time I spent philosophizing. I am equally happy to have new friends for life in Alice Lai, Daphne Tsatsoulis, Ryan Musa, Robert Deloatch, Kevin Shih, and Jason Rock (not in alphabetical order).

One of the best parts of joining UIUC was the breadth of research and friendliness of faculty. I want to particularly thank Dan Roth for both his academic knowledge and support, above and beyond this thesis, and invitations on the holidays. To Sariel Har-Peled, Hello Hello. To Jeff Erickson, thank you for many long chats at every cafe in town.

I would also like to take a moment to thank those whose influence started long before the Ph.D., including my immediate family (Avital, Bruce, Leah, Reuven, Ruth, and Yosef), the entire wonderful extended family (Bisks, Gordons, Kesslers, Stolars, and Veners), many friends from UT-Austin, and Lorenzo Alvisi. In particular, I would like to acknowledge the team and lineage that is my grandfather and my mother. Together they brought the

dream of being a scientist into my life at an early age. They nurtured it even when none of it made sense to them, and I am immensely lucky to have them. My sister Avital will always be my most important partner in life, and she has supported me through every aspect of it so far.

I would like to acknowledge an ever growing list of hundreds of people in Urbana and at UIUC who made it the very best place to spend six years. I want to thank John Blitzer, who took me under his wing for a phenomenal six months and introduced me to a whole new family in Mountain View. Finally, I would like to thank my committee: Cheng Zhai for reminding me to think beyond NLP, Dan Roth for challenging my most fundamental assumptions about language, and Jason Eisner for a rigor I hope to one day emulate.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

NLP	Natural Language Processing
CL	Computational Linguistics
CFG	Context-Free Grammar
POS	Part-of-Speech
UPOS	Universal Part-of-Speech
CCG	Combinatory Categorial Grammar
PCFG	Probabilistic Context-Free Grammar
CCM	Constituent-Context Model
DMV	Dependency Model with Valence
F1	Harmonic Mean
U/LF1	Unlabeld Undirected and Labeled Directed F1
HMM	Hidden Markov Model
EM	Expectation-Maximization
FOL	First-Order Logic
TSG	Tree-Substitution Grammar
HDP	Hierarchical Dirichlet Process
HMM	Hidden Markov Model
BMMM	Bayesian Multinomial Mixture Model
NED	Neutral Edge Detection
BoW	Bag-of-Words

Chapter 1

Introduction

Human language serves as the means for transferring information from one person to another, but despite being so fundamental must be learned by observing other speakers in the environment. Language is a highly structured form of communication whose rules (syntactic, morphological, phonetic, etc.) are necessary for understanding.

The task of a language learner is to both build a model of the world and learn how words and sentences map to objects and concepts in that space. Within Artificial Intelligence we try to replicate this process by breaking up the learning and reasoning into subtasks, ignoring their co-dependence. Within Natural Language Processing (NLP) and Computational Linguistics (CL) we further break down the problem into many subtasks, including: part-of-speech tagging, named-entity resolution, syntactic parsing, coreference resolution, sentiment classification, question answering, information extraction, and so on. Methodologically, breaking down and understanding different aspects of language is an important first step because it organizes the space of phenomena captured and expressed by language.

To understand these pieces better, the community has built annotated resources and linguistic theories for their structures. Within NLP, machine learning is then applied to abstract sets of features and labels to predict and recover these phenomena.

Unfortunately, the creation of so many tasks and their corresponding data is very labor intensive while also seemingly at odds with the ease with which humans acquire language from exposure to others speaking around them. Unsupervised methods try to lower the annotation burden and address the question of learnability by attempting to replicate existing results on natural language tasks without the use of annotated training data.

In this thesis we focus specifically on the unsupervised acquisition of grammars. The problem domain can take many forms but assumes the presence

of segmented text and the learning goal is to produce syntactic structures (dependencies). We will define these and other assumptions in more detail throughout the thesis.

1.1 Thesis Statement

This dissertation introduces an algorithm for the unsupervised induction of Combinatory Categorial Grammars, and probabilistic models for parsing with the resultant grammars that complete or surpass the state of the art. One major difference to prior work is that our use of Categorial Grammars enables us to perform a detailed, linguistically informed error analysis. This analysis will expose failings in the current approachs to grammar induction and enable head-to-head comparisons between an unsupervised and supervised parser on both a labeled syntactic and semantic evaluation. These are essential first steps towards the goal of inducing unsupervised parsers.

1.2 Contributions of this thesis

Chapter 2: A brief introduction to fundamentals of Combinatory Categorial Grammar (CCG). We explain how the grammar is defined, used for parsing and semantics, and important ways in which it differs from CFGs and Dependency Grammars. CCG will form the basis of our approach in this thesis.

Chapter 3: All existing approaches to creating an unsupervised categorial grammar require a linguist to hand-craft lexical categories. Existing approaches to inject supervision into grammar induction require knowledge of attachment preferences and language-specific syntax. By contrast, we introduce a simple procedure for automatically inducing a grammar with only knowledge of nouns and verbs.

Chapter 4: Given an automatically induced grammar, we train a simple PCFG model that outperforms many existing approaches on English and performs competitively with systems that use more knowledge than our approach.

Chapter 5: We devise a novel non-parametric Bayesian model for CCG which uniquely captures the constrained structure of CCG parses. We explore a uniquely simple modeling of punctuation, extend our model to include words as lexical productions, and explore non-local dependencies.

Chapter 6: We perform competitively or outperform existing approaches on 29 corpora of various languages, formalisms, and train/test splits. Additionally, we produce human interpretable predictions about the lexicon of the languages tested. We then extend our approach to produce the only labeled parses from an unsupervised induction system in the literature. This enables direct comparison to supervised parsers and an in-depth linguistic analysis with insights into the learnability of language. We enumerate constructions which require semantics and therefore require rethinking the grammar induction task.

Chapter 7: We perform grammar induction and labeled evaluation with induced clusters from raw text in lieu of gold part-of-speech tags. Further, our very limited supervision provides large performance gains in many languages and insights into why tagging and syntax should be estimated jointly.

Chapter 8: Finally, we exploit CCG's clean interface to semantics in order to produce grounded semantics from a supervised, semi-supervised, and unsupervised parser. The unsupervised system captures enough semantics to outperform a bag-of-words model on complex sentences. Further, we demonstrate a clear correlation between syntactic and semantic performance, indicating that future work can and should cleanly integrate semantic feedback.

Our goal in this thesis is to introduce a state-of-the-art system in grammar induction and provide a new and unique analysis of the task. We succeed in defining a new state-of-the-art in many languages and begin to break down the abstraction barriers assumed between tagging and parsing and between parsing and semantics. The creation of a true language learner will require that none of these tasks be tackled in isolation or assumed to be independent.

Chapter 2

Background

2.1 Syntactic Parsing

Syntactic parsing is the task of analyzing sentences and annotating them with informative syntactic analyses. Automatic syntactic parsing aims to build computer programs which annotate novel, unseen texts with accuracy comparable to a human annotator. The purpose of creating these syntactic analyses, parses, is to identify a latent structure that links words in a sentence and which we believe is required for extracting the semantics of a sentence. Achieving this raises a number of important questions about the *true* structure of language or what we want an automated system to be capable of producing. There are a number of syntactic theories that have been developed for expressing the range of phenomena observed across the world's languages which can serve as possible representations. Within the empirical field of NLP, in whose purview the task of automatic syntactic parsing falls, the type of information a specific theory captures will be crucially important as input to other language tasks.

Perhaps the most common style of structures are Context-Free Grammars (CFG). A CFG captures knowledge about language by labeling constituents and specifying in which ways they are capable of combining.

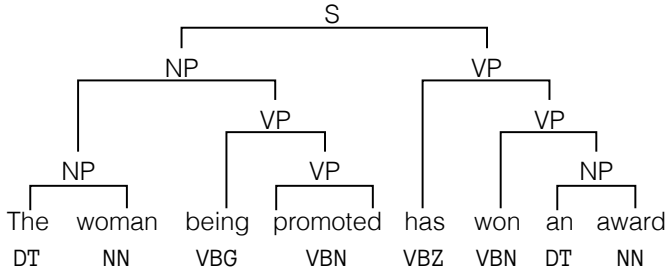


Figure 2.1: A sentence parsed with a Context-Free Grammar.

For our initial discussion we will focus on the analysis in Figure 2.1 as a demonstration. We start with an eight word sentence and its part-of-speech (POS) tags:

The	woman	being	promoted	has	won	an	award
DT	NN	VBG	VBN	VBZ	VBN	DT	NN
DET	NOUN	VERB	VERB	VERB	VERB	DET	NOUN

Most NLP systems assume they have access to these tags. Part-of-speech tags are simply labels which indicate which (morpho)syntactic class a word belongs to. The labels presented here are from the Penn Treebank [1] (top row) (which we will discuss in more detail later). A simplification to the "universal part-of-speech tagset" (UPOS) [2] is included on the bottom row for ease of exposition. Unless stated explicitly, all results in this thesis are working over language specific tagsets, but we often use UPOS to easily convey information and intuition about experimental design.

2.1.1 Part-of-Speech tags

Part-of-speech tags represent basic morphosyntactic information associated with individual words. Tag sets differ in size and specificity, depending on the morphology of the language and on decisions made by their designers. The following are the Penn Treebank tags for our example sentence:

Penn	Description	UPOS equivalent
DT	Determiner	DET
NN	Noun, singular or mass	NOUN
VBG	Verb, gerund or present participle	VERB
VBN	Verb, past participle	VERB
VBZ	Verb, 3rd person singular present	VERB

There are 36 tags [3] in total, not including punctuation, which are mapped to 7 UPOS tags. As can be seen from this example, the Penn Treebank tags capture a much finer level of granularity which can prove very important when making syntactic attachment decisions.

In morphologically rich languages, the word form might capture even more information as it might denote case, gender, and number. In these cases, the

designer of the resource must decide at what granularity to stop differentiating the tags in their tagset. In this thesis we will work with a number of language across language families (e.g. Arabic, Basque, Bulgarian, Chinese, Czech, Danish, Dutch, English, Hebrew, Japanese, Portuguese, Slovene, Swedish, Spanish...) which range from 12 to >200 tags (see Appendix A for more details).

Recovering linguistic structure without the use of these informative labels will be the focus of Chapter 8.

2.1.2 Context-Free Constituency Grammars

Producing a structure like the one in Figure 2.1 requires a number of grammar rules be specified in advance. For this example those include:

S	$\rightarrow$	NP	VP
NP	$\rightarrow$	NP	VP
NP	$\rightarrow$	`DT`	`NN`
VP	$\rightarrow$	`VBG`	VP
VP	$\rightarrow$	`VBZ`	VP
VP	$\rightarrow$	`VBN`	NP
VP	$\rightarrow$	`VBN`	

The grammar denotes which words or groups of words can combine and labels the resulting structure. For example, the combination of a determiner (`DT`) with a noun (`NN`) results in the creation of a noun phrase (NP). We are specifying how parts of a sentence interact and labeling those interactions and incremental derivations. Having the rules of a grammar also defines the valid sentence of a language.

The rules of the grammar can most easily be read right to left: "*An NP combines with a VP to produce a sentence S*". This corresponds to how bottom-up parsing is performed. Whenever the right hand side of a rule is satisfied, we apply the rule to create a new, larger, constituent. This can be done efficiently with dynamic programming (Cocke-Kasami-Younger: CKY [4, 5], we discuss this in depth in the next chapter, section 3.3.1). One common means of deriving a grammar of this form is by reading the rules from a treebank of parses. A treebank is a large collection of parsed sentences which have been manually analyzed. From this resource we can read off the rules of a language's grammar and we can train a model to apply the rules

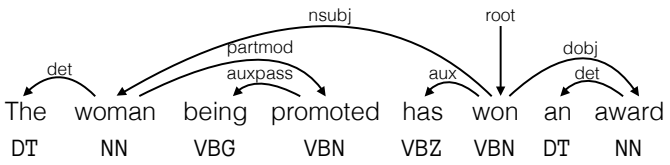


Figure 2.2: A sentence parsed with a Dependency Grammar.

in similar proportions or contexts to those in the treebank. While we will focus most of our discussion of English grammar on the Penn Treebank and treebanks that are based upon it. It is important to note that its creation represents only a single domain and a single linguistic theory. While the Penn Treebank has been foundational, many other treebanks existed at the time (e.g. Brown [6, 7], London-Lund [8], Lancaster-Leeds [9], and Lancaster-Oslo/Bergen Corpus [10][1]) and many others have been created since.

Unlike in our single sentence example with seven rules, the space of rules needed to analyze a large body of text is much larger and the array of substructures is more complicated. This is where the strength of a statistical parser becomes crucial. A supervised parser learns which rules are likely to be applied in which contexts by mimicking the analyses present in the treebank during training. In this way, the model captures information about not only which analyses are possible within the grammar of a language but also which are likely. When treebanks are available, this is a highly efficient and useful technique.

2.1.3 Dependency Grammars

One type of information that is not easily recovered from a CFG parse tree is the notion of syntactic headedness. For example, when analyzing a sentence, it is often useful to identify the subject or object of the verb. This information can be annotated and modeled directly in an alternate formalism: Dependency Grammar [11, 12].

Dependency grammars are based on word-word relations. In particular, dependency grammar substitutes the phrases and structural categories of a constituency tree for directed arcs between words and functional categories as labels [13]. In Figure 2.2 we have a dependency grammar parse of the same sentence from before.

[1] `http://clu.uni.no/icame/manuals/LPC/LPC.PDF`

In the constituency tree we labeled spans of words and the merger of sub-structures. They were given names (noun-phrase, verb-phrase, ...) which indicated how and where they could be used, structurally, to complete a parse tree. In a dependency grammar, arcs are drawn between words so there are no non-terminals, but instead all information is carried by the choice of the arc's direction and label. Arcs are drawn from heads to dependents. There are a number of criteria for headedness [14]. For example, a word may be a dependent because it is an optional word (e.g. modifier) which can be dropped without affecting the meaning of the sentence. A word may also be a dependent because it is the argument of another word (e.g. nouns are arguments of a verb). Unfortunately, the choice of a syntactic head is difficult, and any annotation standard must make many potentially arbitrary headedness decisions (auxiliaries vs main verbs, the role of conjunctions, etc). As dependency grammars will be the main type of treebank against which we evaluate models in this thesis, several of these annotation decisions will be discussed at length (Sections 3.1.4 and 7.2.3).

Finally, functional labels are attached to each arc. For example, the labels mark the subject, *woman*, as `nsubj` and the direct object, *award*, as `dobj`. In a dependency treebank there are several dozen such arc labels marking important distinctions between constructions. Again, a full grammar of how words or part-of-speech tags can be linked and labeled can be read off of the set of derivations provided in a treebank. These outline the space of (word/tag, label) tuples allowed by a language and the frequency of those labels. We will convert many of the predicted structures of our models (Section 3.2.4) into this format for evaluation (Sections 5.5 and 7).

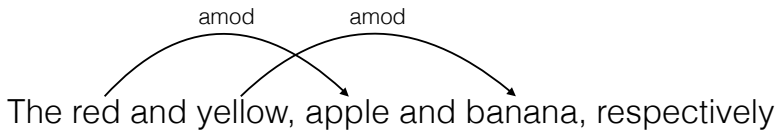
2.1.4 Computational Complexity and Expressivity

Both constituency and dependency grammars are widely used within CL/NLP and treebanks for training parsers have been constructed in dozens of languages. As with the choice of part-of-speech tagsets, whose size is an important design choice for distinguishing or conflating the linguistic phenomena of a language, the rules of a grammar encode many more linguistic biases. What types of constructions to annotate and express are both theoretical linguistic questions about the properties of a language and practical questions of implementation and modeling. In the computational hierarchy of lan-

guages and automata, as one moves up the hierarchy from regular grammars to recursively enumerable, the computational power required to accept the language moves from finite state automata up to a Turing Machine [15, 16].

Therefore, by choosing to analyze language with a context-free grammar we are making parsing the grammar easier by bounding our computation to only requiring pushdown automata (efficiently parseable in $O(n^3)$). Simultaneously, we are making linguistic assumptions about the complexity and expressive power of human language.

The way this additional complexity would be represented in the formalisms we have seen thus far would be to allow the brackets of the constituency parse to cross one another, or for the edges of the dependency parse to cross. Within dependency parsing this is eponymously referred to as non-projective parsing. We see use cases for this across languages, but most modern parsers do not try to capture it. A simple example of crossing dependencies happens in English when analyzing constructions with the word *respectively*. How much computational power and expressivity should be captured by a grammar formalism is still an open question [17, 18]



In the next chapter we will introduce another, less common, grammar formalism: Combinatory Categorial Grammar (CCG). CCG will take the form of a constituency grammar but will capture non-projective dependencies. It will be efficiently parseable ($O(n^6)$) but lies slightly outside of context-free, in a computational class called Mildly Context-Sensitive [19, 20, 21].

2.1.5 Supervised Parsing

In the presence of treebanks we can train models to produce an automatic syntactic parser. First, the rules of the grammar being read from the training data define the space of possible parses for a sentence. Second, a model must be trained to score constituents or arcs in each sentence. In a generative grammar, like those used in this thesis, we assume that the sentence is *generated* by the parse tree, which is itself the outcome of a process of rewrite rules originating with some initial start symbol (like S in our grammar).

More concretely, let us return to our constituency parse from before. Given that we have a sentence (S), we can define a distribution over the rules which have S as the left hand side, and randomly draw a right hand side transformation. In this case, we draw the right-hand side NP VP. We can say this happens with probability $p($ NP VP $\mid$ S $)$. We can now recurse down both children with probabilities $p($ `VBZ` VP $\mid$ VP $)$ on the right and $p($ NP VP $\mid$ NP $)$ on the left until we reach terminal nodes. Preterminal nodes are those which emit a word and then no longer recurse. For example, we might have the grammar produce part-of-speech tags and then have each tag emit a word, $p($ promoted $\mid$ `VBN` $)$. In this way we have assigned a probability to each step of the parse and the observed words. By taking the product of all of the probabilities of left-hand sides (X) producing right hand sides, non-terminals or leafs (α) we can compute a joint probability for the parse tree (T) and the sentence ($\vec{w}$):

$$p(\vec{w}, T) = \prod_{X \to \alpha \in T} p(X \to \alpha)$$

This very simple probability model is referred to as a probabilistic context-free grammar (PCFG) [22] and forms the basis for many more sophisticated parsers [23]. Analogous models and processes can be used for dependency trees, but where a CFG defines rules for how to combine non-terminals (NP, VP, ...), dependency grammars specify specify which words can be attached to which other words.

2.1.6 Evaluating Syntactic Predictions

We have just briefly defined constituent and dependency grammars, and how a basic probability model can be defined over parses. The goal of creating syntactic parsers is to predict structures which must then be evaluated against some human annotated ground truth.

Constituency Within constituency grammars the basic unit is the labeled constituent. To evaluate parses one first computes the percent of correct predictions, yields, (C). The yield of a non-terminal is the region of the sentence it spans. There are two common ways to compute this evaluation metric: labeled and unlabeled. In both cases the model must predict the correct yield to increment C, but in labeled evaluation the yield must also

have the correct non-terminal label predicted to be counted. This number is then divided by the number of total predicted yields (S) to compute a precision $p = \frac{C}{S}$, and by the total number of ground truth constituents (G) to compute a recall $r = \frac{C}{G}$. Papers then report the precision, p, recall, r, and harmonic mean (f-score) [24] of the two: $f = \frac{2pr}{p+r}$. This metric is also known as Parseval [25].

Dependency For dependency grammars we perform an analogous computation but instead of computing the number of correct yields, we compute the number of correct arcs (labeled, unlabeled, and undirected) to define C. In much of this thesis, we will be comparing to dependency trees. A dependency tree assumes that every word can only be the dependent of one other word in the sentence. This means the number of arcs predicted and in the gold truth match (being equal to the number of words in the sentence), making the precision and recall computations redundant. For this reason, most of the evaluation will report only a single accuracy number: $\frac{C}{G}$. In dependency graphs (where a word may be the dependent of several heads), we will return to reporting precision, recall and f-score. Finally, in much of the discussion of this thesis, the evaluation will be over unlabeled arcs and as such we will only present directed accuracies (DA) as opposed to the labeled arcs used for evaluating supervised systems. Directed arcs capture both the pair of words being linked and which of the two is the head.

2.2 Grammar Induction

We have outlined two common representations of syntactic grammar and how a treebank is used to train syntactic parsers. The goal of unsupervised methods is to take tasks, like syntactic parsing, and attempt to predict the same linguistic structures without access to labeled training data, in this case the treebank. The desire to learn structure without labeled supervision is not new, as being able to do so would be a huge boon for our understanding of language acquisition and allow us to quickly scale NLP to the ∼3,500 living written languages of the world instead of the few dozen for which data has been annotated.

Interest in creating a system which performs unsupervised language acquisition dates back several decades and has been worked on continuously

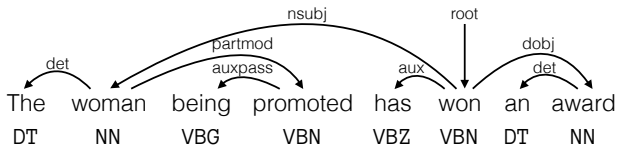
Table 2.1: Selected Historical Works in Grammar Induction/Estimation

1992	**Carroll & Charniak**: Two Experiments on Learning Probabilistic Dependency Grammars from Corpora [26]
1996	**de Marcken**: Unsupervised Language Acquisition [27]
1998	**Yuret**: Discovery of Linguistic Relations Using Lexical Attraction [28]
2001	**Clark**: Unsupervised Language Acquisition: Theory and Practice [29]
2001	**Paskin**: Grammatical Bigrams [30]
2005	**Klein**: The Unsupervised Learning of Natural Language Structure [31]
2006	**Smith**: Novel estimation methods for unsupervised discovery of latent structure in natural language text [32]
2009	**Cohen & Smith**: Shared Logistic Normal Distributions for Soft Parameter Tying in Unsupervised Grammar Induction [33]
2009	**Headden III, Johnson, & McClosky**: Improving Unsupervised Dependency Parsing with Richer Contexts and Smoothing [34]
2010	**Berg-Kirkpatrick & Klein**: Phylogenetic Grammar Induction [35]
2010	**Blunsom, Cohn & Goldwater**: Inducing Tree-Substitution Grammars [36]
2010	**Naseem & Barzilay**: Using universal linguistic knowledge to guide grammar induction [37]
2011	**Boonkwan & Steedman**: Grammar Induction from Text Using Small Syntactic Prototypes [38]
2013	**Spitkovsky**: Grammar Induction and Parsing with Dependency-And-Boundary Models [39]
2013	**Christodoulopoulos**: An Iterated Learning Framework for Unsupervised Part-of-Speech Induction [40]

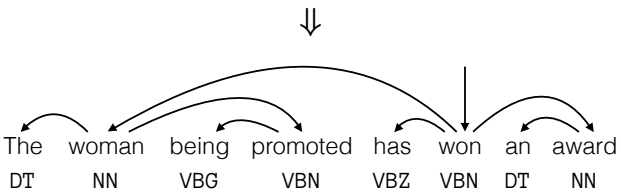
since then (Table 2.1). The goal is to explore the limits of learnability of structure from text. Of particular interest to us is work that follows from and expands on Klein 2005 [31]. The task as defined by Klein and Manning [41] (extending the paradigms of [28, 30, 26]) entails predicting unlabeled dependency arcs between words.

The goal of grammar induction should be to create a cog which produces linguistic structures of equal utility to those produced by supervised syntactic parsers. Doing so has largely been assumed impossible because reproducing the structures requires knowledge of the linguistic theory used to generate those structures and the space of the grammar used in the annotation. If linguistic structures are truly that idiosyncratic it implies there is little if anything truly universal about language in our representations. The literature's response to this issue has been to produce unlabeled representations which lack much of the detail present in labeled structures produced by supervised parsers.

Klein and Manning's Constituent-Context Model (CCM) attempts to learn constituency bracketings (unlabeled and without crossing), and their Dependency Model with Valence (DMV) learns to produce an unlabeled dependency tree. In both cases the phrase labels for constituents or the arc labels for dependencies have been discarded. These labels are arbitrary symbols created as part of an annotation standard, and are therefore not recoverable in an unsupervised manner. This drastically simplifies the learning and parameter space of the models.



Correct Output of a Supervised Parser



Correct Output of an Unsupervised Parser

Within this task definition, the literature generally assumes that every word depends on a single head (the source of the arrow pointing to it), and

that with the addition of a root node this structure forms a tree. Performance can therefore be computed as a simple prediction accuracy over word pairs (Section 2.1.6).

In the first half of this thesis, we will design models for predicting these unlabeled structures to follow the literature's evaluation standard. Unfortunately, it should be obvious from the figure above that a tremendous amount of useful linguistic information is lost when labels are removed. Further, since most NLP systems that use a parser assume the presence of these labels as features which help classify important distinctions in a task, we also limit the utility of grammar induction by changing the definition of parsing. In particular, if unsupervised parsers are treated as a cog to be substituted into NLP systems when treebanks are unavailable, the representation and features produced will be greatly impoverished.

The second half of this thesis addresses this concern by introducing a method for automatically producing labeled dependencies, evaluating induced parses on the same metric as supervised parsers, and showing how unsupervised labels can be used in downstream tasks.

2.2.1 Methods

Since the work of Klein and Manning, a number of approaches have extended their models or introduced completely novel dependency grammar induction systems. These approaches have provided new insight into smoothing [34], more sophisticated priors, constraints, curricula, and initialization [33, 42, 43, 44, 45, 46], and looked at the effects of using additional data from the web [47]. Given the weak performance of many approaches, methods for injecting additional supervision [37, 38, 48, 35] have also been introduced and have proven very effective.

We provide here a brief overview of some of the history of approaches and insights in the literature. Early work, like that of Klein and Manning, focused on inducing grammars in English, Chinese and German for short sentences (at most 10 words without punctuation). Much of the work grew more narrow by honing in on English for evaluation before broadening to other languages. The last decade witnessed a transition to two dozen languages with varying amounts of data and from short sentences of length 10, to 20, 40, and finally to full set of sentences in the test set. The exact way to evaluate is still not

settled (Section 2.2.3).

Spitkovsky et al. have produced a number of papers in this space. Their early work demonstrated improved DMV performance when trained with a curriculum [49] of short sentences before longer ones. They then attempted to control for ambiguity in longer sentences by using Viterbi-EM [50], which only updates the model with counts from the best model prediction. As the average sentence in the Penn Treebank is on the order of 40 words, there is a tremendous amount of data (and ambiguity) in these sentences. To further inform their model, they came up with mechanisms for constraining the parse based on its punctuation [51]. Finally, plagued by local optima, they introduced techniques for random restarts and model recombination [45] to circumvent the non-convex nature of the learning problem.

Headden et al. [34] introduced the Lexicalized Extended Valence Grammar which lexicalizes the DMV models and includes a valence term which captures subcategorization information and models the proximity of a word to the head. Cohn et al. [36] learn a non-parametric Bayesian model of tree-substitution grammar that is biased towards a sparse grammar with shallow productions. Underlying the model is a base distribution computed over CFG trees derived from the DMV model.

Another approach introduced by Mareček and Žabokrtský is to compute the primacy of different word classes through a score they call "reducibility" [47], which computes how often a tag can be dropped from a sequence without the sequence becoming ungrammatical. This provides a measure of the importance of a given word with the insight that, for example, verbs are more essential to a sentence than nouns, and nouns are more necessary than adjectives. They followed up with work to better model the valence of words [46] and function words [52]. They were also the first to demonstrate the utility of working over larger corpora from the web. This is perhaps in part because their model is very data-hungry.

2.2.2 Adding Supervision to Grammar Induction

Several approaches have also explored incorporating additional supervision. Naseem et al. [37] demonstrate the effectiveness of universal linguistic knowledge. Their model has access to 13 soft universal dependency constraints:

Root → Auxiliary	Noun → Adjective	Verb → Noun
Root → Verb	Noun → Article	Verb → Pronoun
Preposition → Noun	Noun → Noun	Verb → Adverb
Adjective → Adverb	Noun → Numeral	Verb → Verb
Auxiliary → Verb		

For example, the rule "Noun → Adjective" will bias the model towards treating adjectives as the dependant of nouns. They only impose these rules in expectation. In this way, specific analyses can violate the rules to complete a parse, but the analysis of the complete corpus will, on average, exhibit these attachment preferences. Their work experiments with how performance is affected as a function of how strongly these biases are imposed. Naseem et al. also evaluate a variant of their system that uses a number of highly effective English-specific heuristics at test time. This is a good demonstration of using limited annotation for strong performance gains. In particular, when evaluating directed attachments on short English sentences they show how a poorly performing system with no rules can go from a directed accuracy of 24.9 to 71.9 with a handful of universal rules and 73.9 with English specific ones.

Boonkwan et al. [38] demonstrate a simple way to get these rules for a language and encode them in a categorial grammar (see Chapter 3). They do so via a 30 question survey that covers basic facts about the language. The results of this survey determine the inventory of lexical types for the language, which are then mapped to specific part-of-speech tags by the experimenter to create a custom language specific lexicon. In this way, they model a realistic annotation environment where a linguist sits down with grammatically savvy native speaker for a short questionnaire. It is possible that their approach could be further extended to ease the burden on the participant. They show nice results for how their performance changes as a function of the questions answered (40.2 → 74.8)

Kuzman et al. [48] use a less direct source of supervision by exploiting bitext projections as constraints. Relatedly, Berg-Kirkpatrick et al. [35] use biases from the phylogenetic history of languages to inform sharing of model parameters across languages.

2.2.3 Evaluation and Idiosyncrasies

Coordination, verb chains, and relative clauses are just a few of the many common but difficult constructions for a parser to analyze. Specifically, all of them require knowledge of the specific annotation standard of the treebank being used for evaluation, as their analyses differ greatly from one annotation standard to the next.

When training a supervised parser, the model has access to the correct analyses in the training data. These same annotation standards, and largely the same grammar, were used for annotating the test data as well. For this reason, it is generally safe to assume that modeling the training data well will correlate with good and consistent predictions on the heldout test data. In grammar induction, the model is presented with text in a language and must predict structures without having seen any examples or annotation guidelines. This means that we have little guarantee that the structures it finds will match the annotation of a test set. This is particularly problematic for constructions whose annotation is inherently ambiguous. One such example is coordination. In our experiments, we will look at over a dozen languages and find five different annotations standards for conjunction.

In a constituency treebank, two conjoined nouns might be simply represented by a ternary rule: NP $\rightarrow$ NP conj NP. In fact, our system will produce structures of this form, but we will need to convert this ternary relation into dependencies for evaluation. We have identified five main styles of conjunction in our data (Figure 2.3), although several corpora distinguish multiple types of coordinating conjunctions which use different styles (not all shown here). These all differ from how CCG handles coordination (Section 3.1.4). This is one particularly easy annotation to spot which is completely arbitrary, and our system will have to be instructed as to how to produce dependency arcs based on the treebank being analyzed.

Alternate Metrics

A direct comparison between different dependency treebanks, dependencies produced by CCG [53, 54], or the output of induction systems is difficult and inconsistent, since dependency grammars allow considerable freedom in how to analyze specific constructions such as verb clusters (which verb is the head?), prepositional phrases and particles (is the head the noun or the

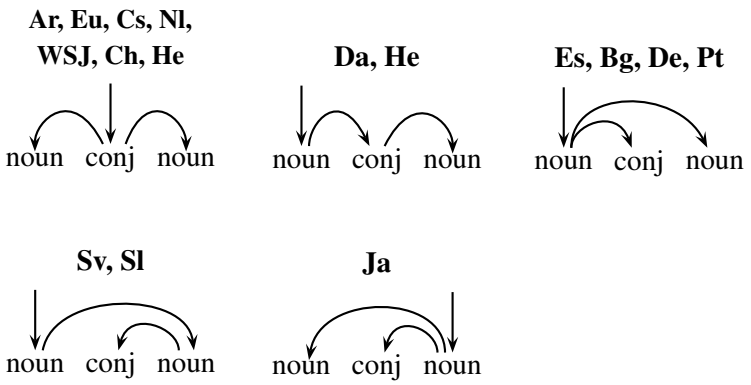


Figure 2.3: In the treebanks used for evaluation, different standards exist for annotating coordination. While not exhaustive, this table demonstrates five of the most common schemes used in the literature. Syntactically these are identical and traditionally CCG draws arcs only to the arguments without attaching the conjunction. For the purposes of comparison with the literature we have implemented these five translation schemes.

preposition/particle?), subordinating conjunctions (is the conjunction a dependent of the head of the main clause, and the head of the embedded clause a dependent of the conjunction, or vice versa?), and this is reflected in the fact that the treebanks we consider often apply different conventions for these cases. Although remedying this issue is beyond the scope of this work, these discrepancies very much hint at the need for a better mechanism to evaluate linguistically equivalent structures or treebank standardization. One such approach was that proposed by Schwartz et al. [55] who introduced Neutral Edge Detection (NED) as a metric that tried to smooth out these decisions. Unfortunately, it also eliminates useful and linguistically important distinctions making it unsuitable for our purposes. Another approach is to simply evaluate undirected edges. In this way, head decisions are ignored but the structure of the tree maintained. Unfortunately, again this simple approach does not remedy issues like coordination but also discards important and useful information from the tree.

2.2.4 Data Splits and Head-Finding rules

An additional source of ambiguity when interpreting results in the literature is differences in which data the system was trained or tested on, and the style of head-finding rules that were used for evaluation, even when the same

corpus was used. Head-finding rules are a means of converting constituency trees into dependencies. A number of them exist in the literature [56, 57, 58, 59] for the Penn Treebank and they make different decisions about how arcs should be drawn for a given constituent. We have collected a few of the experimental setups in Table 2.2. The Penn Treebank also has two forms, the original annotated corpus and a newer fix to the treebank to incorporate internal NP structure [60]. Whether this fix is included by various approaches is unclear. When compiling this table, we first looked for details in data sections of each paper. When details were missing we trusted they used the setup of whomever they were comparing against. Unfortunately, even if the data section specifies what was done many comparisons are inconsistent.

Finally, while Collins [57][2] is often cited as the source of certain head finding rules, these rules originated with Magerman (1995) [56], and there are a number of software implementations of the conversions which differ in some of their details. It is therefore unclear when a paper claims to be using Collins' head-finding rules if they are implementing their own, using Nivre's [61]'s Penn2Malt[3] conversion which is actually from Yamada and Matsumoto [58] or another system[4]. This complicates evaluation further.

A final dimension to this problem is what sentence lengths were used during training and evaluation. The community appears to be converging to evaluating on all sentence lengths. The best demonstration of how much of an effect the choice of head-finding rules and amount of data might have on a system is Spitkovsky et al. 's work [45] shown in Table 2.3 we see the performance of their system when trained on two different subsets of the same corpus and evaluated on different annotations of the same corpus. The training and testing sections are denoted in the second and third column. The style of head-finding rules is in the first column and the model's performance is the last two. DA stands for Directed Attachment. Here, they report what percent of the arcs in the training data they correctly predict. DA10 is an accuracy computed on only short sentences with at most 10 words (ignoring punctuation), and DA∞ is computed on the full test-set. When looking at the full test set we see how the same model has a six point performance difference between the two setups. The performance might slip further, if

[2] `http://www.cs.columbia.edu/~mcollins/papers/heads`
[3] `http://stp.lingfil.uu.se/~nivre/research/Penn2Malt.html`
[4] e.g. `http://nlp.cs.lth.se/software/treebank_converter/`

Year	Paper	Train	Dev	Test	Head rules
2004	Klein & Manning[41]	00-24		00-24	Collins
2007	CoNLL 2007[62]	02-11		$\subset$23	J&N
2009	Cohen & Smith[33]	02-21	22	23	Collins
2009	Headden III et al. [34]	02-21	22	23	Collins
2010	Berg-Kirkpatrick et al. [35]	02-21		02-21	Collins
2010	Blunsom & Cohn[42]	02-21	22	23	Collins?
2010	Naseem & Barzilay[37]	02-21		02-21	Collins
2011	Boonkwan & Steedman[38]	02-22		23	Collins
2012	PASCAL Shared Task[63]	00-24		23	J&N[3]
2013	Spitkovsky & Jurafsky[45]	00-24		23	Collins
		02-11		$\subset$23	J&N

Table 2.2: We have selected several key pieces of work in the literature to demonstrate the lack of consensus on how to evaluate grammar induction within English. *Collins* refers to the Collins head-finding rules [64] and *J&N* to Johansson & Nugues [59]. The final two lines both refer to the same paper which contains two evaluations. Here, we use a subset to denote that an only part of section 23 was used for evaluation.

	Train	Test	DA10	DA∞
Collins	00-24	23	72.0	64.4
J&N 07	02-11	$\subset$23	75.0	58.2

Table 2.3: Spitkovsky et al. 's [45] reported results on WSJ section 23.

evaluated on the Johansson & Nugues head-finding rules with NP internal structure. Other older results on that difficult setup include Blunsom & Cohn [42], who perform at a directed attachment of 56.0, which may be an equivalently good model but has not been trained and tested on the same data. This wild variation leads to some difficulty in determining the true state-of-the-art on the task of grammar induction. We will try and provide as fair comparisons as possible throughout the thesis. In particular, in Chapter 7 we will perform three different English performance evaluations to attempt to compare to most of the available literature.

[3]Includes the internal NP structure of Vadas and Curran (2007)

Chapter 3

Combinatory Categorial Grammars

Combinatory Categorial Grammar [65, 66, 67, 68, 53] is a linguistically expressive, lexicalized grammar formalism which associates rich syntactic types with words and constituents. These rich representations and their functional nature, discussed below, allow for a transparent mapping from a word's syntactic role to one or more semantic interpretations.

3.1 Formalism

3.1.1 Categories

The basic vocabulary of the grammar assumes two atomic types: S (sentences) and N (nouns). Complex types are of the form X/Y or X\Y and represent functions which combine with an immediately adjacent argument of type Y to yield a constituent of type X as the result. The slash indicates whether the Y precedes (\) or follows (/) the functor. The lexicon pairs words with categories and is of crucial importance since it captures the only language-specific information in the grammar. An English lexicon may contain entries such as:

N: $\{he, girl, lunch, ...\}$	N/N: $\{good, the, eating, ...\}$
S\N: $\{sleeps, ate, eating, ...\}$	(S\N)/N: $\{sees, ate, ...\}$
S\S: $\{quickly, today...\}$	(S\N)/(S\N): $\{good, the, ...\}$

Work in this chapter was first published in Y. Bisk and J. Hockenmaier, "Probing the linguistic strengths and limitations of unsupervised grammar induction," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Beijing,China, July 2015. [133] and is reprinted here with permission by the copyright holder.

While the set of categories is theoretically unbounded, the inventory of lexical category types is in practice assumed to be finite and of a bounded maximal arity (typically 3 or 4). The arity of a category is simply the number of arguments it takes. Above, the intransitives, *sleeps*: $\mathsf{S}\backslash\mathsf{N}$, takes one argument while the transitives, *sees*: $(\mathsf{S}\backslash\mathsf{N})/\mathsf{N}$, take two.

3.1.2 Combinatory rules

Categorial grammar rules are defined as schemas over categories (where X, Y, Z etc. are category variables and $| \in \{\backslash, /\}$ is a slash variable), and are usually given in a bottom-up manner. All variants of categorial grammar [65, 66] use the basic rule of forward ($>$) and backward ($<$) application, which specifies that a functor $\mathsf{X}|\mathsf{Y}$ can combine with an adjacent argument $\mathbf{Y}$ to form a new X:

$\mathsf{X}/\mathbf{Y}$	$\mathbf{Y}$	$\Rightarrow \mathsf{X}$	$(>)$	Forward Application
$\mathbf{Y}$	$\mathsf{X}\backslash\mathbf{Y}$	$\Rightarrow \mathsf{X}$	$(<)$	Backward Application

(C)CG parses are typically written as logical derivations. A simple example of forward application in English is the attachment of a determiner *The* to a noun *man*. Backward application is used by the verb *ate* to take its subject *man*:

$$\dfrac{\dfrac{\dfrac{\textit{The}}{\mathsf{N}/\mathsf{N}} \quad \dfrac{\textit{man}}{\mathsf{N}}}{\mathsf{N}}> \quad \dfrac{\textit{ate}}{\mathsf{S}\backslash\mathsf{N}}}{\mathsf{S}}<$$

CCG includes additional rules: in function composition (the B combinator of Curry and Feys [69]), the arity of the secondary functor can vary from 1 to a fixed upper limit n. To unify our notation, we will denote application as a functor of arity 0 (B^0) below when comparing a grammar's expressivity.

$\mathsf{X}/\mathbf{Y}$	$\mathbf{Y}$	$\Rightarrow$	X	$>\mathsf{B}^0$	Forward Application
$\mathsf{X}/\mathbf{Y}$	$\mathbf{Y}\|\mathsf{Z}$	$\Rightarrow$	$\mathsf{X}\|\mathsf{Z}$	$>\mathsf{B}^1$	Forward Composition
$\mathbf{Y}$	$\mathsf{X}\backslash\mathbf{Y}$	$\Rightarrow$	X	$<\mathsf{B}^0$	Backward Application
$\mathbf{Y}\|\mathsf{Z}$	$\mathsf{X}\backslash\mathbf{Y}$	$\Rightarrow$	$\mathsf{X}\|\mathsf{Z}$	$<\mathsf{B}^1$	Backward Composition

If the functor is composing into a category with n arguments, we refer to this as **generalized composition**:

$$\begin{array}{llcll} \mathsf{X/Y} & \mathsf{Y}|\mathsf{Z_1}|...|\mathsf{Z_n} & \Rightarrow & \mathsf{X}|\mathsf{Z_1}|...|\mathsf{Z_n} & >\mathsf{B^n} \quad \text{Forward} \\ \mathsf{Y}|\mathsf{Z_1}|...|\mathsf{Z_n} & \mathsf{X\backslash Y} & \Rightarrow & \mathsf{X}|\mathsf{Z_1}|...|\mathsf{Z_n} & <\mathsf{B^n} \quad \text{Backward} \end{array}$$

In practice, n rarely takes a value greater than three in the literature, which will also be the limit of what we explore in this thesis. When discussing specific rules, we will instantiate n appropriately. Two cases of generalized composition are presented below for where an adverb is modifying a transitive and ditransitive verb:

$$\begin{array}{llcll} \mathsf{S/S} & (\mathsf{S\backslash N})/\mathsf{N} & \Rightarrow & (\mathsf{S\backslash N})/\mathsf{N} & >\mathsf{B^2} \\ \text{quickly}(\mathsf{S}) & \text{ate}(\mathsf{N_1}, \mathsf{N_2}) & & \text{quickly}(\text{ate}(\mathsf{N_1}, \mathsf{N_2})) & \end{array}$$

$$\begin{array}{llcll} \mathsf{S/S} & ((\mathsf{S\backslash N})/\mathsf{N})/\mathsf{N} & \Rightarrow & ((\mathsf{S\backslash N})/\mathsf{N})/\mathsf{N} & >\mathsf{B^3} \\ \text{quickly}(\mathsf{S}) & \text{took}(\mathsf{N_1}, \mathsf{N_2}, \mathsf{N_3}) & & \text{quickly}(\text{took}(\mathsf{N_1}, \mathsf{N_2}, \mathsf{N_3})) & \end{array}$$

Finally, when the directionality of the slashes does not match categories may still compose and this is denoted as (forward/backward) crossing composition.

$$\begin{array}{llllll} \mathsf{X/Y} & \mathsf{Y\backslash Z} & \Rightarrow \mathsf{X\backslash Z} & (>\mathsf{B^1_X}) & \text{Forward Crossing} \\ \mathsf{Y/Z} & \mathsf{X\backslash Y} & \Rightarrow \mathsf{X/Z} & (<\mathsf{B^1_X}) & \text{Backward Crossing} \end{array}$$

Just as before, the functors are the left and right categories for forward and backward crossing composition respectively. This is easy to see because the functor must have an argument which matches the second category's return type (bolded here).

3.1.3 Derivations

The combinators directly control the expressivity of the grammar (the space of allowed constructions) and its ambiguity. For example, in the following sentence with a ditransitive, the verb takes three arguments, and the adverb would like to combine with it to form *quickly took*.

$$\begin{array}{cccccc} I & \textit{quickly} & \textit{took} & \textit{her} & \textit{home} \\ \hline \mathsf{N} & \mathsf{S/S} & (\mathsf{(S\backslash N)/N})/\mathsf{N} & \mathsf{N} & \mathsf{N} \end{array}$$

To do so with forward composition $>\mathsf{B}^1$ requires the verb first to consume both of the arguments to its right, so there is only an intransitive verb $\mathsf{S}\backslash\mathsf{N}$ remaining. $>\mathsf{B}^1$ is then sufficient to attach the adverb to the verb, as the adverb's argument, the (forward) argument S of S/S, can take the return type of the intransitive, the S of $\mathsf{S}\backslash\mathsf{N}$, by only consuming the innermost atomic S:

$$\dfrac{\dfrac{\dfrac{I}{\mathsf{N}} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\textit{quickly}}{\mathsf{S}/\mathsf{S}} \quad \dfrac{\textit{took}}{((\mathsf{S}\backslash\mathsf{N})/\mathsf{N})/\mathsf{N}}}{(\mathsf{S}\backslash\mathsf{N})/\mathsf{N}}>}{\mathsf{S}\backslash\mathsf{N}}> \quad \dfrac{\textit{her}}{\mathsf{N}}}{\mathsf{S}\backslash\mathsf{N}}>\mathsf{B}^1 \quad \dfrac{\textit{home}}{\mathsf{N}}}{\mathsf{S}\backslash\mathsf{N}}$$

In contrast, with arity three generalized composition ($>\mathsf{B}^3$) the modifier can recurse through three arguments in search of its argument to attach immediately:

$$\dfrac{\dfrac{I}{\mathsf{N}} \quad \dfrac{\dfrac{\textit{quickly}}{\mathsf{S}/\mathsf{S}} \quad \dfrac{\textit{took}}{((\mathsf{S}\backslash\mathsf{N})/\mathsf{N})/\mathsf{N}}}{((\mathsf{S}\backslash\mathsf{N})/\mathsf{N})/\mathsf{N}}>\mathsf{B}^3 \quad \dfrac{\textit{her}}{\mathsf{N}} \quad \dfrac{\textit{home}}{\mathsf{N}}}{(\mathsf{S}\backslash\mathsf{N})/\mathsf{N}}$$

This additional power is both a blessing and a curse. In these examples, the additional expressivity creates ambiguity in the grammar without increasing expressivity. Specifically, the ability to combine *quickly* with *took* in these two different derivations does not increase the number of semantic interpretations of the sentence. This power will become necessary when discussing non-standard word order.

$$\dfrac{\dfrac{\textit{he}}{\mathsf{N}} \quad \dfrac{\dfrac{\textit{ate}}{(\mathsf{S}\backslash\mathsf{N})/\mathsf{N}} \quad \dfrac{\textit{quickly}}{\mathsf{S}\backslash\mathsf{S}}}{(\mathsf{S}\backslash\mathsf{N})/\mathsf{N}}<\mathsf{B}^2_\times \quad \dfrac{\textit{the lunch he bought}}{\mathsf{N}}}{\mathsf{S}\backslash\mathsf{N}}$$

For example, in this sentence *quickly* must combine with *ate* before any of the verb's arguments (subject and object) can be taken. Because the verb is transitive, having two arguments, we require arity two composition. Specifically, we require backwards crossing composition ($<\mathsf{B}^2_\mathsf{X}$) in this example.

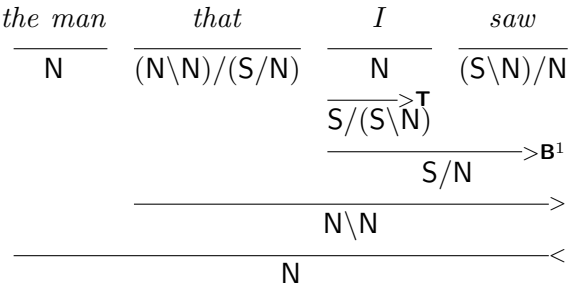
Additionally, in these examples we see two different types of dependency relations between words or constituents [11] which CCG distinguishes explicitly: in a head-argument relation, the head $\mathsf{X}|\mathsf{Y}$ (e.g. $\mathsf{S}\backslash\mathsf{N}$) takes its

dependent Y (N) as an argument, whereas in a head-modifier relation, the modifier X|X (N/N) takes the head X (N) as an argument. One of the roles of composition in CCG is that it allows modifiers such as adverbs to have generic categories such as S\S, regardless of the verb they modify.

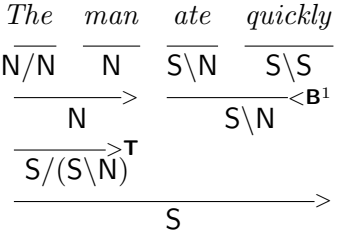
CCG also includes a unary type-raising rule, which reverses the relationship between functors and arguments, and allows Y (which may be the argument of X\Y) to turn into a functor that takes X\Y as an argument and returns X:

Y	⇒	X/(X\Y)	(>T)	Forward Type-Raising
Y	⇒	X\(X/Y)	(<T)	Backward Type-Raising

The category X\Y or X/Y is generally restricted to be of a type that also occurs in the lexicon of the language [53]. Although type-raising Y followed by application of the type-raised argument to the original functor X\Y is equivalent to applying the functor itself (and we, therefore, disallow type-raised categories to apply to other categories to reduce the number of spurious ambiguities), type-raising and composition act together to capture non-local dependencies which arise through extraction or coordination, e.g.:



While ambiguity is an inherent component of syntactic parsing, we want the model only to have to decide between unique semantic interpretations of the sentence. In contrast, the spurious ambiguities introduced by type-raising add derivations to the parse forest without adding new semantic analyses. One such derivation is presented below, which uses type-raising unnecessarily.



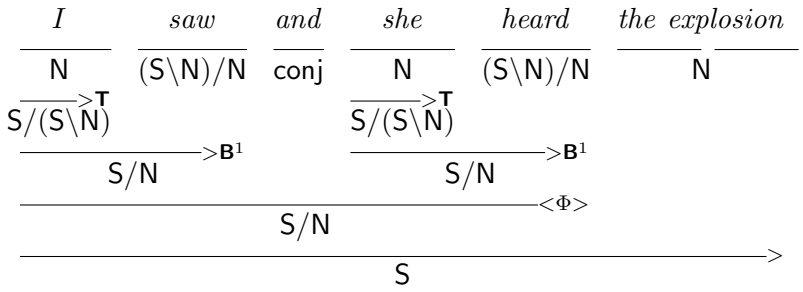
If spurious ambiguities can be restricted or removed, it significantly reduces the size of the parse forest and in turn makes learning a parsing model easier. We constrain the use of type-raising with a CCG normal form (Section 3.3.2), and we analyze the benefits to training in Section 7.2.1.

3.1.4 Coordination

Finally, for coordination we assume a special ternary rule (following CCGbank [70]) that is binarized as follows:

X	X[conj]	$\Rightarrow_{\&1}$	X	(&1)
conj	X	$\Rightarrow_{\&2}$	X[conj]	(&2)

In coordination, CCG will allow us to extract the argument role of every conjunct. Below we demonstrate how the syntactic parse enables us to capture this argument-filling information about predicates. Currently, there is no work in the unsupervised grammar inductin literature that can recover the same style of information from dependency or constituency trees.



This sentence contains two predicates: *saw* and *heard*. These share an object but have different subjects. The ternary subject rule allows us to extract the semantics of both predicates and conjoin them:

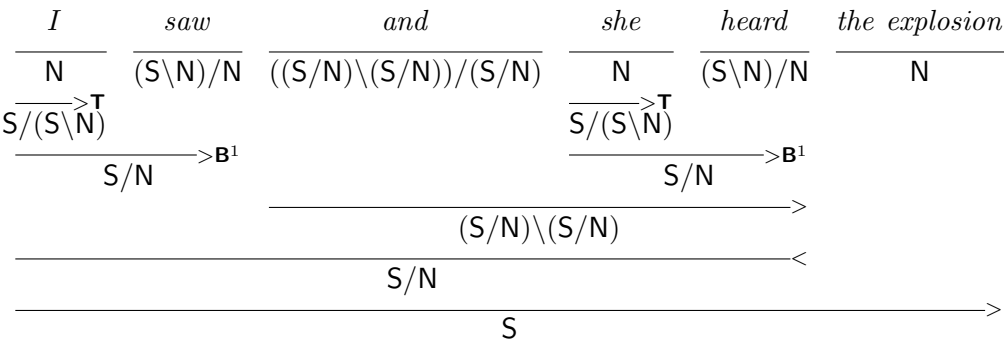
$$saw'(I, explosion) \ \wedge \ heard'(she, explosion)$$

We will discuss semantics further in the next section.

Another option for representing coordination is a category of the form (X\X)/X for conjunctions. This seemingly simpler (non-ternary) treatment, requires introducing complex categories (e.g. ((S/N)\(S/N))/(S/N)) to merge

the two transitive verbs. Another category would be required for ditransitive verbs, another for nouns, another for adjectives, and so forth. This leads to a proliferation of otherwise unnecessary categories. Additionally, its use causes us to choose one verb (the left) to serve as the "main" verb, dropping the argument link from *heard* to *explosion* and breaking the clean predicate-argument structure of the CCG derivation.

The corresponding derivation is as follows:



We will discuss how these choices manifest in dependency structures in Section 3.2.2.

3.1.5 Semantics

A fundamental difference between CCG and other formalisms is its transparent syntax-semantics interface. CCG derivations arise from categories combining via application and composition. This is only possible because the categories are themselves functions with simpler categories as arguments and results.

This functional representation of the sentence allows for capturing the predicate-argument structure of language, which can be exploited for semantic tasks. In particular, each CCG category can be converted to a logical representation [71], syntactic parses have been used to construct ungrounded semantic parses [72] (Chapter 9), and semantic role-labeling labels correspond cleanly to the arc labels of CCG [73].

Lambda Calculus [74, 75, 76] is a common semantic representation for defining logical predicates, variables and meaningful ways in which they can combine. The full representation has equivalent computational power to Turing Machines [77], but we use it here simply as glue for constructing

logical representations. We will briefly work through how an example CCG derivation aligns with lambda calculus β-reductions.

In the following sentence,

John saw and Mary heard the explosion

there are two words to which we attach semantic predicates, and treat treat nouns as semantic constants:

word	CCG Category	Semantics
saw	$(\mathsf{S}\backslash\mathsf{N})/\mathsf{N}$	$\lambda y.\lambda x.saw'(x,y)$
heard	$(\mathsf{S}\backslash\mathsf{N})/\mathsf{N}$	$\lambda y.\lambda x.heard'(x,y)$

Lambda calculus is defined in terms of variables, abstraction and application. Here we are assuming the variables x and y which have yet to take on any meaning. Second, we also have abstractions (anything of the form $\lambda x.f(x)$). Finally, application is the process of applying a function to an input.

In function application (also called β-reduction), a lambda-abstraction (function) $\lambda x.f$ is applied to an argument a. β-reduction returns a copy of f in which all (free) occurrences of the variable x are replaced by a:

Application:
$(\lambda y.\lambda x.saw'(x,y))(the\ explosion) \quad \rightarrow \quad \lambda x.saw'(x, the\ explosion)$

Throughout a CCG derivation, operations like application and composition correspond to predicates taking argument. In this way, the syntactic operations correspond to lambda calculus operations (β-reduction). It is important to note that every operation we perform is only binary in the parse, but the underlying predicates are n-ary functions. The ability to break up the process by decomposing a function such that each argument is satisfied individually (n binary operations to fill every one of the n arguments in a function) is made possible by currying [69].

This sentence encodes two events, a seeing event and a hearing event which are built during the syntactic derivation (Figure 3.1):

$$saw'(John, the\ explosion) \wedge heard'(Mary, the\ explosion)$$

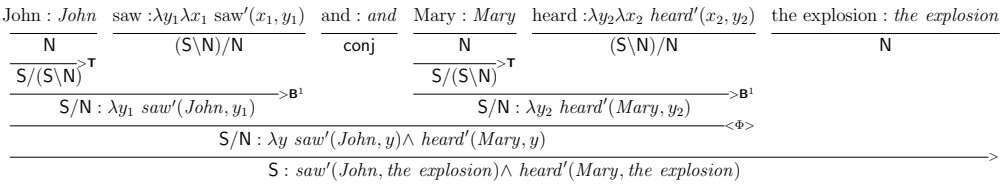


Figure 3.1: A CCG derivation builds a logical representation of the text.

If the corresponding first-order logic (FOL) predicates are provided to words and categories, then a logical representation, useful for a downstream task, can be constructed using the procedure above [78, 79, 80, 81, 82, 83]. Alternatively, by following the procedure above, with dummy predicates, the semantics interface of CCG can build a generic logical representation, that does not correspond to a particular task/environment, but does isolate the sentence's semantics. In Chapter 9, we will demonstrate how the resulting representation, which we will call "ungrounded semantics", can be grounded to database semantics for tasks like information extraction.

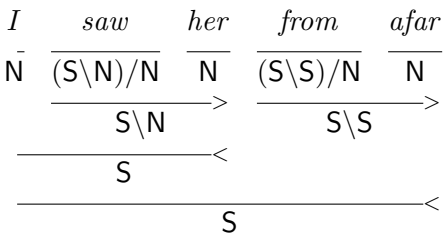
3.2 Dependencies and CCG

Dependency Grammars make the simplifying assumption that every word can only be the dependent of exactly one other word. The choice of how to define headedness is treebank dependent. By defining dependencies to hold between functors and their arguments, CCG avoids making potentially arbitrary headedness decisions and does not require words be only the dependent of one other word. CCG predicate-argument dependencies were introduced by Clark et al. (2002) [54] and have become the basis for evaluating CCG based parsers [84, 85].

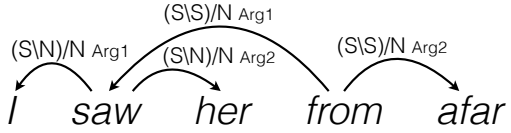
Relying on predicate-argument structure alleviates the need to make potentially arbitrary decisions about the head of a given constituent. It also often leads to producing DAG structures in lieu of trees (e.g. Section 3.2.2).

3.2.1 Basic Predicate-Argument Dependencies

In particular, for every argument-taking lexical category we trace through the derivation to find which word filled its argument slot. This argument-taking lexical category is then used as the label for the arc, supplemented by the slot being filled. We can see this in the following parse:



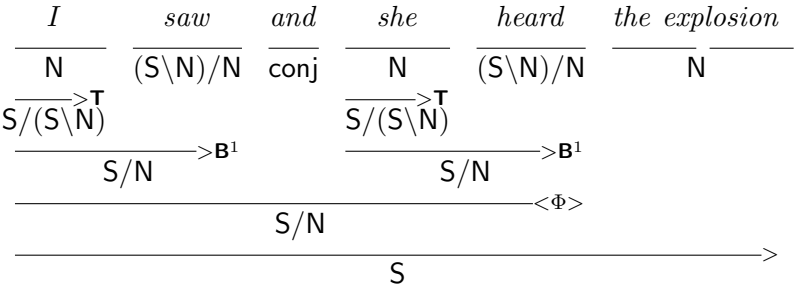
By tracing through the derivation, we keep track of each argument and when it is filled. For example, let us look at just the two words *saw her*. Here *saw* takes *her* as an argument. In particular, *saw* has the category $(\mathsf{S}\backslash\mathsf{N})/\mathsf{N}$ which has two argument slots. The inner-most N is the first argument and the outermost is the second. Since the second slot is being filled we draw an arc from the predicate (*saw*) to its argument (*her*) and label it with the predicate's category, $(\mathsf{S}\backslash\mathsf{N})/\mathsf{N}$, and which slot, 2, is being filled. Once we repeat for every word in the sentence we can produce the following graph:



Or, if we format these arcs in a table, we can read them off as follows:

Dependent	**Head**	**Label**	**Slot**
I	saw	$(\mathsf{S}\backslash\mathsf{N})/\mathsf{N}$	1
her	saw	$(\mathsf{S}\backslash\mathsf{N})/\mathsf{N}$	2
her	from	$(\mathsf{S}\backslash\mathsf{S})/\mathsf{N}$	1
afar	from	$(\mathsf{S}\backslash\mathsf{S})/\mathsf{N}$	2

One thing to note is that lexical categories are used to label the arcs, so no other category introduced during the derivation will affect the labels. In particular, we can see this with type-raising. With coordination we discussed the use of type-raising:



With the exception of scrambling [86], the effect of type-raising is to change the order in which the arguments are filled but not which word fills which argument slot. By labeling the dependencies with the lexical category we will get the same predicate argument structures with type raising as we would if we were analyzing two simple sentences: *I saw the explosion* and *she heard the explosion*. The full set of labeled dependencies are as follows:

Dependent	**Head**	**Label**	**Slot**
I	saw	$(S\backslash N)/N$	1
she	heard	$(S\backslash N)/N$	1
explosion	the	N/N	1
explosion	saw	$(S\backslash N)/N$	2
explosion	heard	$(S\backslash N)/N$	2

Evaluation metrics for supervised CCG parsers [54] measure labeled f-score (LF1) of these dependencies (requiring the functor, argument, lexical category of the functor, and slot of the argument to all match). A second, looser, dependency evaluation which measures unlabeled, undirected dependency scores (UF1) is often also performed. The third standard CCG evaluation metric is supertagging accuracy [87], which simply computes how often a model chooses the correct lexical category for a given word. This is useful as the correct category is a prerequisite for recovering the correct labeled dependency.

The reason both undirected-unlabeled dependencies as well as directed-labeled dependencies are traditionally evaluated, because of the argument-adjunct distinction of prepositions. In CCGbank, prepositions can be given the PP category to denote they should act as arguments to the verb instead of modifiers. We will not discuss this at length, but the distinction is illustrated by the following two analyses:

She	*walked*	*with*	*him*
N	$S\backslash N$	$(S\backslash S)/N$	N

She	*walked*	*to*	*the store*
N	$(S\backslash N)/PP$	PP/N	N

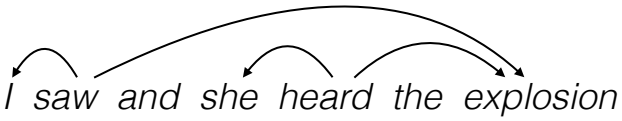
We see that the argument analysis means the verb takes an additional PP (to) argument, rather than being modified by with: $(S\backslash S)/N$. Confusing these analyses produces not only the wrong label but also the wrong head direction. Undirected-unlabeled dependencies will not penalize this distinction as both analyses draw an arc between the preposition and the verb.

For those familiar with CCGbank, we should note that our discussion of prepositions attaching to the verb phrase has used the category $(S\backslash S)/N$, while CCGbank uses $((S\backslash NP)\backslash(S\backslash NP))/NP$. This category allows for the preposition to apply to the verb before it takes a subject argument. The category's first argument though is not filled and we will discuss simplifying this complex category to the simple one in our examples in section 3.4.2.

3.2.2 Coordination Dependencies

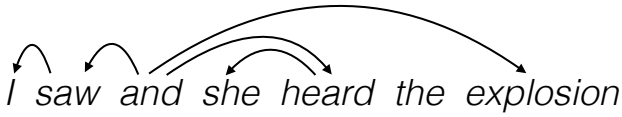
In Section 3.1.4, we introduced a ternary rule for coordination. We also discussed the possibility of using a set of categories of the form $(X\backslash X)/X$ in lieu of adding a special conj category. Given our knowledge of CCG dependencies, we can now illustrate how these approaches differ.

Because CCG draws dependency arcs between predicates and their arguments, we produce a directed acyclic graph which excludes the conjunction, linking its arguments to their predicates, instead of producing a tree which attempts to attach semantic meaning to the conjunction.

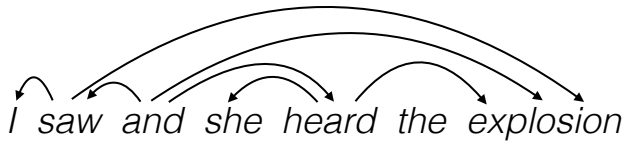


This treatment will be useful for semantic applications, but proves tricky to use when evaluating with dependency treebanks, which require every word be attached to the tree. One way to address this is to use categories of the form $(X\backslash X)/X$ for conjunction, instead of our ternary rule. Doing so yield two possible dependency graphs.

If the N arguments are not co-indexed, *and* serves as a function word taking both verb phrases and the object as arguments:



In contrast, if the arguments are co-indexed we end up with a union of both of the previous dependency structures:



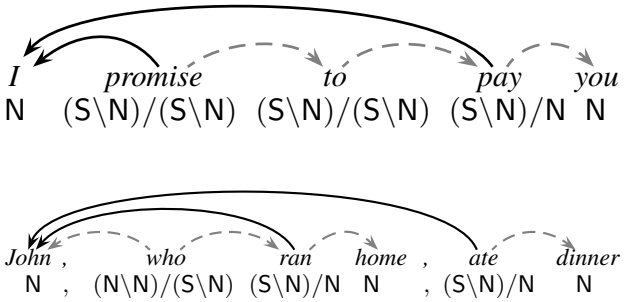


Figure 3.2: Unlabeled predicate-argument dependency graphs for two sentences with co-indexed subjects.

Rather than take either of these options and introduce the extra coordination categories, we will instead implement coordination conversion rules to produce the various styles present in dependency treebanks (Section 2.2.3), or use the ternary rule for accurate comparisons to CCGbank.

3.2.3 Non-local Dependencies and Complex Arguments

One advantage of CCG is its ability to recover the non-local dependencies involved in coordination, control, raising, or *wh*-extraction [54, 84, 70]. Since these constructions introduce additional dependencies, CCG parsers return dependency graphs (DAGs), not trees (like those demonstrated for coordination in Section 3.2). To obtain these additional dependencies, relative pronouns and control verbs require lexical categories that take complex arguments of the form $\mathsf{S\backslash NP}$ or $\mathsf{S/NP}$, and a mechanism for co-indexation of the NP inside this argument with another NP argument (e.g. $(\mathsf{NP\backslash NP_i})/(\mathsf{S|NP_i})$ for relative pronouns). These co-indexed subjects can be seen in Figure 3.2 where two verbs share the same subject (the solid black arcs).

Knowing the indexation is another source of supervision. In many cases there is no ambiguity, but there can be with control verbs [68, 70]:

$$\frac{I}{\mathsf{N}} \quad \frac{\textit{promised}}{((\mathsf{S\backslash N_i})/(\mathsf{S\backslash N_i}))/\mathsf{N}} \quad \frac{\textit{her}}{\mathsf{N}} \quad \frac{\textit{to pay}}{\mathsf{S\backslash N}}$$

$$\frac{I}{\mathsf{N}} \quad \frac{\textit{persuaded}}{((\mathsf{S\backslash N})/(\mathsf{S\backslash N_i}))/\mathsf{N_i}} \quad \frac{\textit{her}}{\mathsf{N}} \quad \frac{\textit{to pay}}{\mathsf{S\backslash N}}$$

The two sentences have the same syntactic analysis, but a different co-indexation. In the first sentence, *I* did both the promising and the paying.

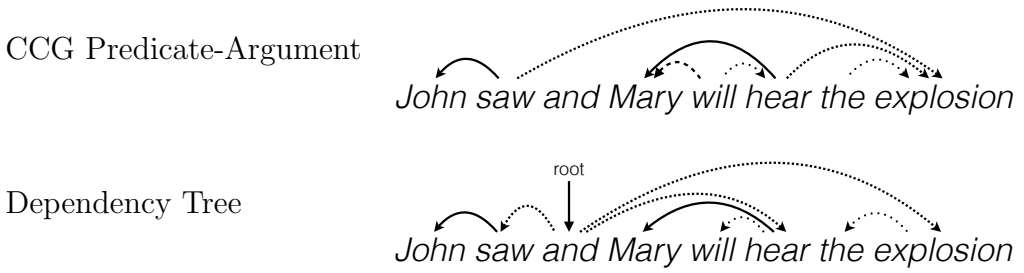


Figure 3.3: For meaningful comparison of CCG dependency structures to the rest of the unsupervised grammar induction literature, we convert the directed edges of a CCG predicate-argument structure (top) to dependency trees (bottom). The arrow types show the corresponding structures in the two analyses.

In the second, *I* did the persuading but the paying was done by *her*. In supervised parsers we assume we have access to the correct co-indexation. In chapter 9, when performing a semantic evaluation of our unsupervised system, we will extend it to produce non-local dependencies, but rather than provide the co-indexation, we will enumerate all of the possibilities and trust that the semantic grounding machinery can handle the ambiguity.

3.2.4 Converting Predicate-Argument to Dependency Trees

In section 7.1 we will compare the output of our system against dependency grammar treebanks. Unfortunately, CCG dependencies, while convenient and transparent to semantics (Section 3.1.5), do not always align with how heads are specified or arcs are drawn in dependency formalisms (Section 2.1.3). This will pose a problem during evaluation (Chapters 5 and 7) when our predicted structures cannot be compared directory to the literature (Section 2.2.4). To try and alleviate this discrepancy we perform several deterministic transformations of CCG dependencies:

1. Treat modifiers as dependents of their heads In CCG modifiers (X/X) take arguments (X) and so the arc is drawn to mirror this process. For comparison, we will invert the direction of these arcs in the output. This flip is shown with the dotted arrows in Figure 3.3 for the words *the* and *will*.

2. Every sentence has a single head word which is the dependent of a "root" node In CCG, the many predicates in a sentence are all treated equally in the output representation. In the CCG analysis in Figure 3.3, we do not assume that one of the action is "primary". Dependency treebanks require that we choose a primary predicate to head the sentence and treat all others as dependents. With the exception of coordinaton, we arbitrarily treat the first verb (or noun if there is no verb) in the sentence as the root.

3. Conjunctions must be or have a dependent. In CCG coordination, the conjunction is there only to assist in coordination by indicating that multiple predicates take the same argument (e.g. the object *explosion* in Figure 3.3). The conjunction itself does not have any dependents or serve as a dependent. Dependency treebanks, on the other hand, require every word be attached to the tree. We will, therefore, implement several conversion schemes based on the specifics of the corpus (Section 2.2.3) which link the conjunction to the tree. The most common analysis is shown in Figure 3.3. Here the conjunction becomes the root of the sentence and takes both verbs and their object as dependents (shown in small-dash arrows).

4. Non-local dependencies are ignored. One of the strengths of CCG is its ability to capture non-local dependencies (Section 7.2.3). These extra dependencies lead to a DAG structure instead of a tree. During the conversion, we drop any non-local dependency. This is equivalent to assuming that CCG categories have no co-indexation. In our example (Figure 3.3), the dashed arc between *Mary* and *will* is dropped for the conversion.

A longer and more complete set of transformations for every treebank was beyond the scope of our work here. There are still many inconsistencies between our conversions and even the English treebank (Section 7.2.1).

3.3 Parsing

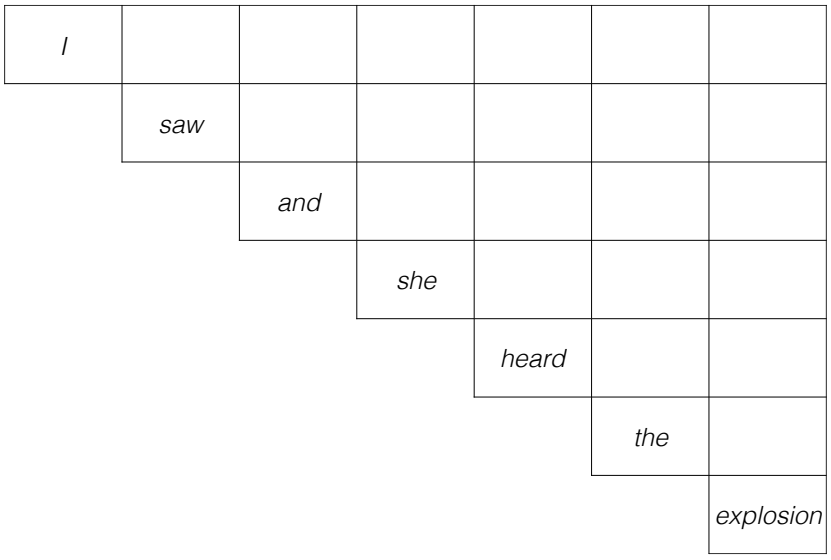
We have just outlined how a CCG derivation is constructed from lexical categories and combinators. It is important to note that this process can be performed efficiently (both in terms of time and space), and all possible

derivations can be found using the CKY algorithm. This is the same algorithm used for parsing context-free constituency grammars (Section 2.1.2), but generalized composition (Section 3.1.2) may allow for more than one left-hand side for a given pair of categories, increasing the computational complexity of parsing to $O(n^6)$ in the worst case.

The reason the same algorithm can be used for parsing both CFGs and CCG is because all of the stages of a CCG derivation can be expressed through unary and binary rules. This includes coordination, whose ternary rules can be binarized.

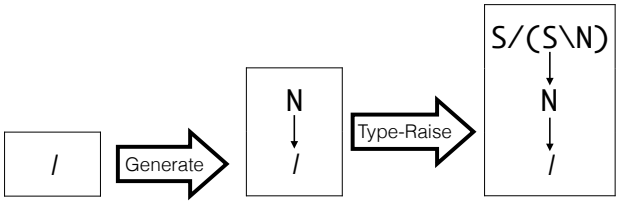
3.3.1 CKY

We begin by creating a square 2D array of size n^2 for a sentence of length n. Every cell in the upper triangular of this array corresponds to a span in the original sentence. The following procedure will analyze every span in the sentence to determine if it corresponds to a syntactic constituent. This array is referred to as a chart, and every value placed inside it is called a chart item. The chart items will correspond to words and CCG categories. In the case of non-lexical items, the chart item also stores pointers to the pair of constituents combined to create the given span. We will now step through this process in detail. To start, we place the words of the sentence on the diagonal of this array:

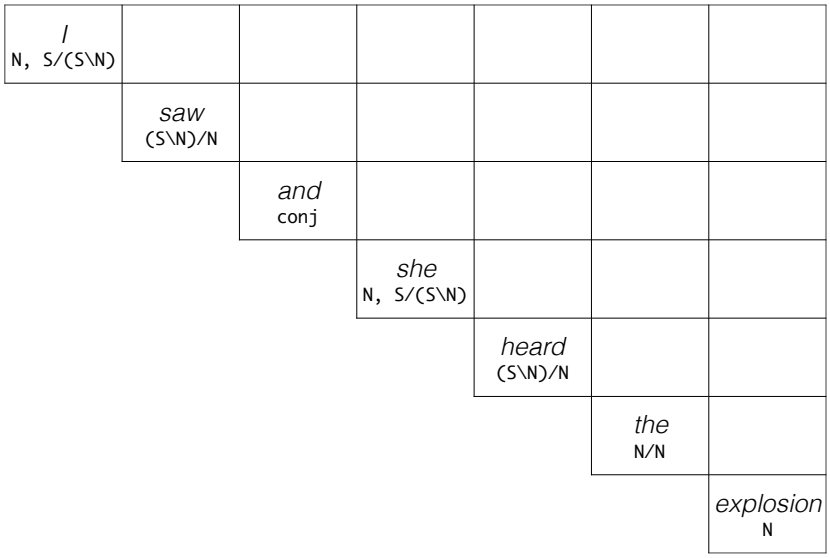


Every word in the language must be licensed by some set of rules in the grammar. We place every lexical category that can produce a given word

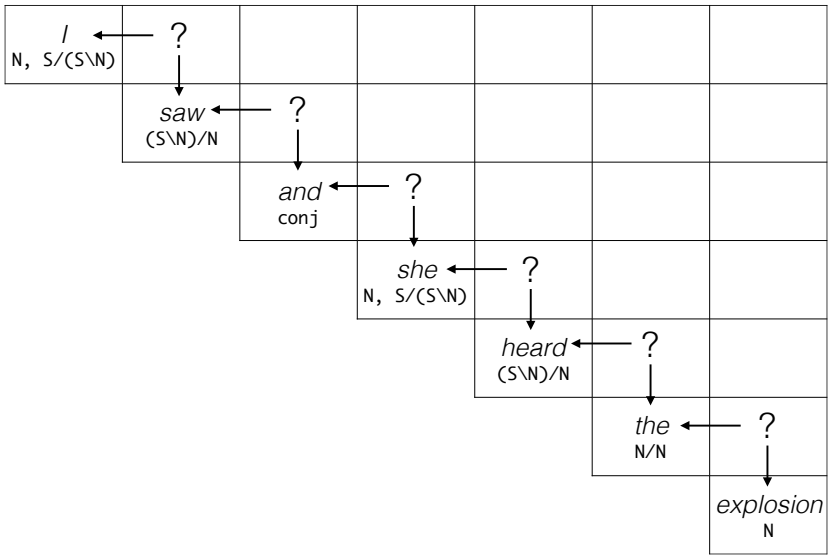
into the same cell as the word it produces. This category is a chart item and it contains a pointer to the word being produced. Rather than allowing every category licensed by the grammar, a model may choose only to use a subset of likely categories for a given word. For simplicity, we will only place a single category in every cell and the necessary type-raised categories. The type-raising and unary rules for generating lexical items all reside within the same cell:



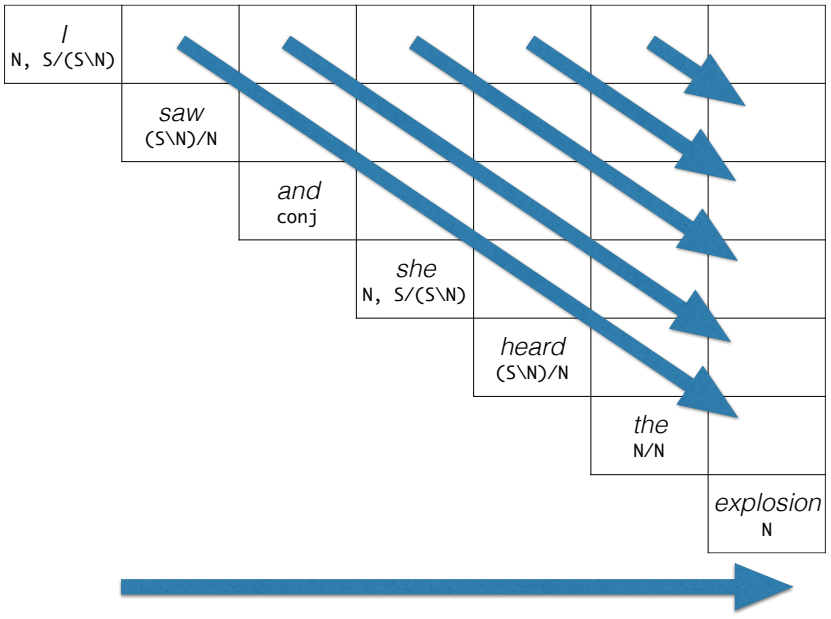
For reasons of space we will simply list the items in each cell next without the use of these generating arrows:



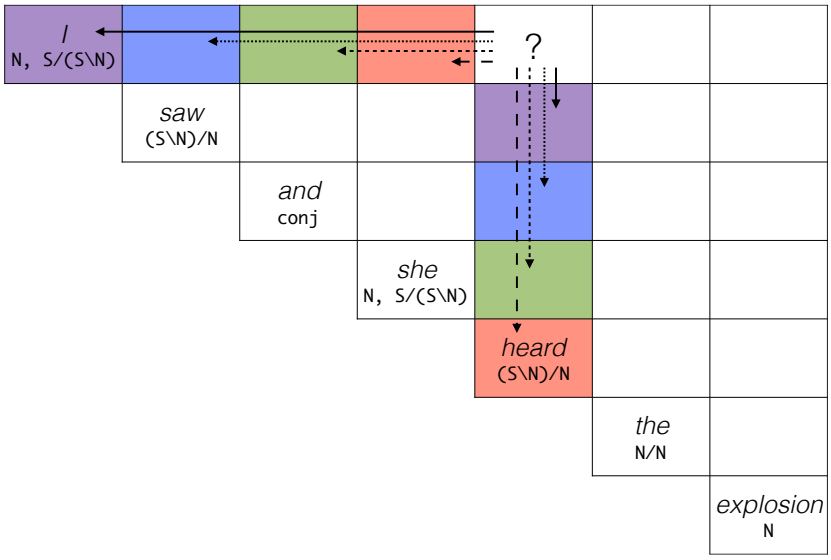
Given the entires/items/edges in these cells, the algorithm progresses by attempting to combine adjacent cell values in the next diagonal



which is then repeated for every cell in the upper triangular of the array. Specifically, spans are analyzed by size, starting with the smallest (the first diagonal adjacent to the words) and working up to the full sentence (the cell in the top right of the array).



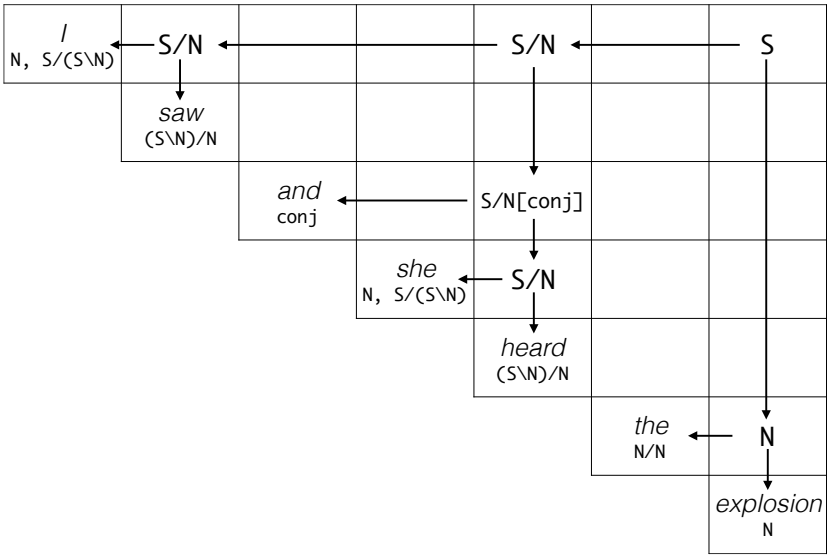
For cells beyond the first row there are several pairs of cells which can be combined, which correspond to string-adjacent spans. We have color coded them here and drawn matching arrows:



Here we are combining constituents rather than words. In particular, we have the following pairs being tested:

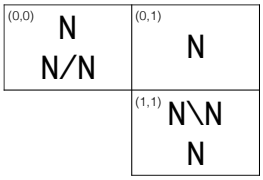
I	saw and she heard	`Purple` / Solid arrows
I saw	and she heard	`Blue` / Dotted arrows
I saw and	she heard	`Green` / Small dashes arrows
I saw and she	heard	`Red` / Dashed arrows

Any of these pairs have the potential to combine to fill the cell in question, which corresponds to the formation of a constituent: *I saw and she heard*. The completed derivation shows that the blue cell combination was correct:



The number of possible ways to combine two entries in a cell depends on the size of the grammar G and the length of the sentence n. There are n possible constituents to combine and n^2 cells to fill. This yields an algorithm with time complexity $O(G \times n^3)$ in the length of the sentence (n) and size of the grammar (G). As mentioned earlier, CCG parsing can be $O(n^6)$ in the worst case. When the set of combinatory rules is spelled out in the form $lhs \leftarrow rhs$, G grows to encompass the additional n^2. Additionally, the chart has space requirements of $O(n^2)$. To recap the process we just executed, remember that every entry in a cell is referred to as a chart item. A lexical chart item is simply a word in the sentence. The rest of the chart items tell us which component chart items combined (binary rules) or were transformed (unary rules) to create the category at that point in the chart. Unary rules, like type-raising, simply apply to an element of the cell to introduce a new chart item in the same cell.

In this discussion, we focused on finding a single parse tree using a single lexical category per cell. In general, many derivations will share common substructure, or chart items, during the parse. For example, we can look at the case of a simple two-word sentence:



In this simple chart, there are two parses, but the top right cell only contains a single chart item. In the CKY algorithm, a parent chart item may store multiple "backpointers" to which pairs of chart items combined to form the parent. In this case, there the parent (N) stores two backpointers. One specifies that N/N of (0,0) combined with N of (1,1) and the second specifies that N\N of (1,1) combined with N of (0,0). This compact representation of parses in the 2D array is referred to as a packed parse forest. By having a single chart item N, which stores knowledge about two parses, if N is used in p parses, we have a compact representation of $2 \times p$ parses.

This data-structure is particularly useful if our grammar or model want to capture fine-grained details about the parse. In this discussion, the chart items are simply words and categories, but if multiple derivations lead to the same chart item a probability model may want to differentiate them by

their lexical heads or other features of the parse. This enables the model to score them differently. We can accomplish this by augmenting the chart items to maintain information about the head word of a constituent or to store additional derivational information about the parse that led there. The former will be useful for models that wish to condition parses on the words within them, and the latter will be necessary for restricting parses based on a normal form.

3.3.2 Normal Form

In CCG, the lexicon contains a tremendous amount of information about the grammar. In particular, because the categories are functions, they combine via application and composition, so no additional rules are necessary for specifying the grammar. That being said, we can constrain the ways and contexts in which categories combine to limit ambiguity. Ambiguity arises when type-raising or higher arity composition are used unnecessarily.

We have discussed the utility and ambiguity of type-raising and the arity of composition, (B^1, B^2, and B^3) in Section 3.1.3. Because the goal of this thesis is to investigate the learnability of grammar, the initial experiments presented in the forthcoming chapters will not entertain the full expressive power and ambiguity of CCG. Experiments will initially be restricted to using a context-free fragment of CCG, which only allows application (B^0), arity 1 composition (B^1) and prohibits type-raising or complex arguments. In this way, we keep the parse forests small and easier to learn from. Later in this thesis (section 6.4.1), we will relax these constraints to B^3, complex arguments and two type-raised categories:

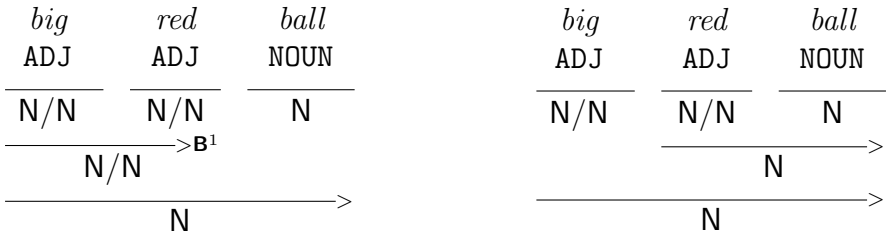
$$\mathsf{N} \rightarrow \mathsf{S}\backslash(\mathsf{S}/\mathsf{N}) \qquad \mathsf{N} \rightarrow \mathsf{S}/(\mathsf{S}\backslash\mathsf{N})$$

Entertaining the broadest grammar possible is necessary for ensuring the greatest coverage of a language's syntactic phenomena as possible. Unfortunately, many spurious ambiguities (Section 3.1.3) are also introduced, which increase the space of derivations, but not of the resultant semantics. Spurious ambiguities refers to CCG's ability to generate a large number of parses that produce the same predicate-argument structure. In this way, the number of parses in the syntactic parse forest is larger than the number of unique semantic analyses. This makes learning more difficult as it splits the probability

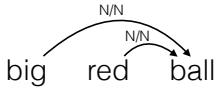
mass of a single interpretation across a number of equivalent derivations. This cannot be completely avoided, but normal-form parsing greatly diminishes this ambiguity, easing the learning and increasing the performance of our models (Chapter 7.2.1). Therefore, unless otherwise specified, all results will be presented will use normal-form parsing.

As noted, CCG allows for many redundant derivations of the same semantic. To address this, there are two CCG normal-forms parsing algorithms (Eisner [88] and Hockenmaier & Bisk [89]) which eliminate differing amounts of these spurious ambiguities.

The normal form works by prohibiting a sequence of parser actions when other, simpler, derivations are possible. For example, the Eisner normal form was introduced to eliminate the unnecessary use of composition when a derivation can be completed with application:



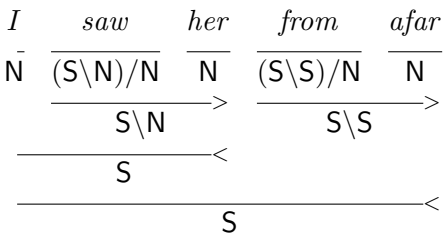
Both derivations yield the same dependencies:



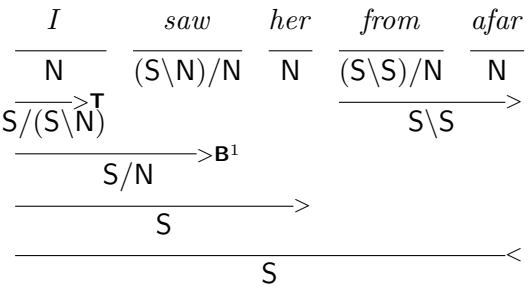
so there is no need to use composition and introduce a second derivation. To capture this restriction, we must record within each constituent how it was derived. Specifically for this case, if X was created via composition it cannot act as the primary functor in application.

Another, problematic case is the unnecessary use of type-raising. When type-raising reorders how arguments are taken to allow the subject slot to be filled before the argument we produce two derivations with the same dependencies:

Correct analysis:



Incorrect analysis which contains a spurious ambiguity:



To eliminate the second derivation, the normal form prohibits the use of type-raising when regular application suffices for completing a derivation.

For many downstream tasks (e.g. semantic parsing in Chapter 9), where the dependencies or semantic representation built by the parse are used, these spurious ambiguities add noise to the system. For grammar induction, eliminating these ambiguities will reduce the space of derivations and parameters by several orders of magnitude. Further, because we evaluate our system with a single best prediction for the parse (Viterbi decoding), we want as much of the mass of the model concentrated on a single prediction as possible. The full effects of the normal form is evaluated in Section 7.2.1. The Hockenmaier and Bisk normal form is more complete when using generalized composition, and so we will use it when experiments are run with type-raising or complex arguments and Eisner otherwise.

As previously noted (Section 3.3.1), implementation of either normal form follows cleanly from CKY. When performing chart parsing one stores a constituent label at every span of the sentence: cell (i, j) for the span $w_i \dots w_j$. To implement the normal forms, each chart item (currently consisting of a constituent label) is augmented with the combinator used in its derivation. This means the cell will now contain many repeated categories each with a different history. Then using this history, we can easily check whether the next combinator is a valid choice for continuing the derivation.

In the case of supervised parsing (see Chapter 5 of [84]) the model is never presented with training data outside of the normal form, and so these

derivations will hopefully never be entertained by the parser. Unfortunately, an unsupervised model does not have this training bias.

3.3.3 Supervised CCG Parsing and Treebanks

We discussed (Section 2.1.5) supervised parsing for context-free parsing of the Penn Treebank with constituency grammars. The same chart parsing and modeling based on a treebank can be done for CCG. In particular, a treebank of CCG derivations provides a lexicon for tags and words in a language and a set of derivations whose unary and binary rules behave in much the same way as a constituency grammar. The primary difficulty in creating a supervised CCG parser is the need for a treebank. Several approaches were proposed for learning a (Combinatory) Categorial Grammar from existing resources [90, 91] before a full conversion of the Penn Treebank to CCG was introduced by Hockenmaier and Steedman [70].

Since then other languages have had their treebanks converted to CCG as well. Most notable are perhaps German [92] and Chinese [93] but many other conversions, partial conversions, and annotations have been introduced [94, 95, 96, 97]. These new resources allow for training supervised CCG parsers.

Since the inception of CCGbank, a number of models have been introduced for highly accurate supervised syntactic parsing [98, 99, 85, 100, 101, 102]. Our goal within this thesis is to build an accurate CCG parser without access to the treebank. Despite being deprived access to the grammar of the language, our output should be comparable to that produced by these sophisticated approaches. In particular, we will use the generative model HWDep of Hockenmaier and Steedman [98] as a comparison system in Chapter 9. This model augments the simple CFG model discussed previously to capture head direction and dependencies between words.

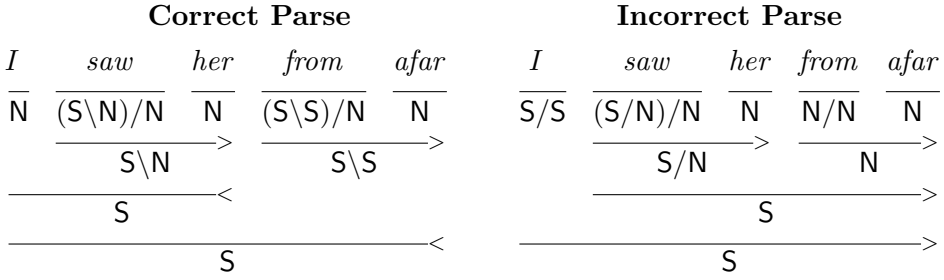
3.4 Evaluation

As was briefly discussed earlier (Section 3.2), CCG has three standard evaluation metrics: Supertag accuracy, Labeled Dependency F1, and Undirected Unlabeled Dependency F1. The supertag (lexical category) of a word indicates if we have correctly determined the syntactic type/role of a word. If a

word with the correct supertag is attached incorrectly, as is commonly the case with prepositional phrases, the labeled dependency will be wrong. Until this work (Section 7.2), the grammar induction literature has not been able to perform labeled dependency evaluation. This is unfortunate as labeled dependency evaluation makes the failings of a system more informative.

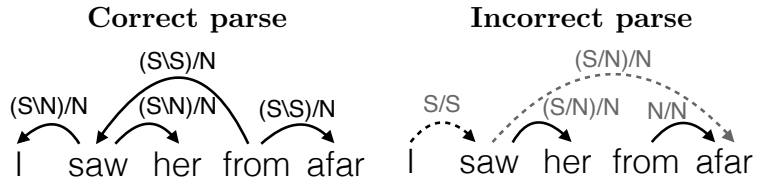
3.4.1 The Need for Labeled Evaluation

The standard definition of grammar induction [41, 63] focuses on the recovery of directed or undirected dependency arcs between words. As previously noted, CCG dependencies indicate predicate-argument relations and are labeled by the predicate and its argument. Predicting labels incorrectly may have no bearing on grammar induction performance (under Directed Attachments: Sections 2.1.6 and 2.2.3), but indicate that the system did not learn the functional role of words in the sentence. Where previous approaches have been unable to produce labels, one contribution of this thesis is performing labeled evaluation of our unsupervised grammar induction system. The types of information lost when a system only produces unlabeled directed attachments becomes clear when analyzing the incorrect parse below. In this analysis, the subject is treated as an adverb, and the prepositional phrase as a noun-phrase object of the verb:



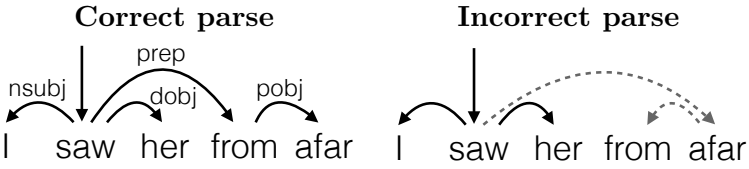
Because none of the argument-taking categories (in the incorrect parse) are correct, none of the labeled directed CCG dependencies are correct. But under the more lenient unlabeled directed evaluation [103], and the even more lenient unlabeled undirected metric [54], two (the solid black arcs) or three (the black arcs) of the four dependencies would be deemed correct:[1]

[1]For ease when reading we will often omit the argument/slot indices from labels where they are easily recovered from the categories.

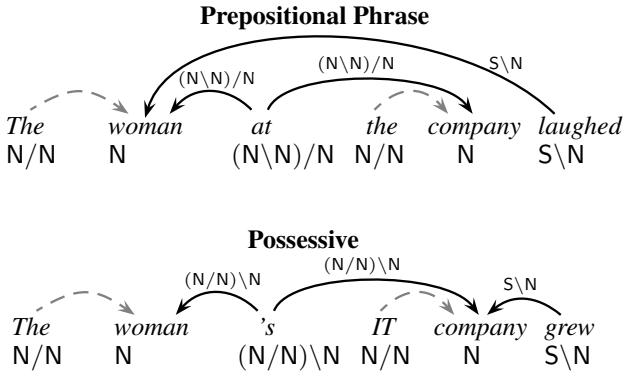


This is because the lexical attachments are often correct, even though they are labeled incorrectly. Treating a subject as an adverb does attach it to the verb, but the semantics are that of modifying the verb rather than filling an argument slot.

This underspecification of the meaning of an unlabeled arc is equally apparent when parses are translated into standard dependency grammar trees. As discussed previously (Section 3.2.4), we translate the CCG analysis to an unlabeled dependency tree by flipping the direction of modifiers, adding a root edge, and removing the labels. Now three out of five attachments are deemed correct:[2]



Again, the dashed gray edges are incorrect. One particularly interesting semantic error that is easily exposed by labeled evaluation is the possessive. The categories of noun-modifying prepositions (*at*) and possessive markers (*'s*) differ only in the direction of their slashes:



The unlabeled dependencies inside the noun phrases are identical, but the heads differ. The first sentence turns the prepositional phrase (*at the*

[2]We do not produce labels, but write them here to help understanding the structure.

company) into a modifier of *woman*. In contrast, in the possessive case, *woman 's* modifies *company*. According to an unlabeled (directed) score, confusing these analyses would be 80% correct for the sentence, whereas LF1 would only be 20%. Without a semantic bias for *companies growing* and *women laughing*, it appears there is no purely syntactic signal for the learner to properly differentiate these parses. One possible way to address this is through grounding entities and predicates to a semantic representation/world (Section 9.3).

3.4.2 CCGbank Simplification

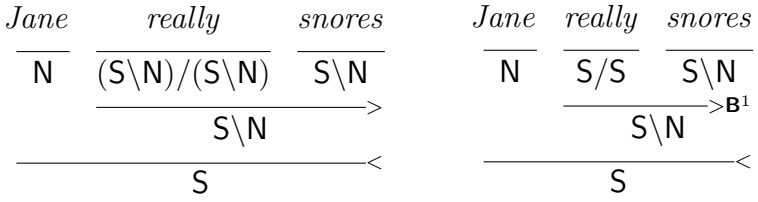
The basis for our labeled evaluation is CCGbank. In our discussion until now we have been using a simple and more basic version of CCG than exists in the treebank. For example, one major difference is that CCGbank augments categories that are not modifiers with morphosyntactic features extracted from the Penn Treebank to mark different types of constructions. These are not part of the basic atomic set of CCG categories, and they affect the manner in which the combinators can be applied.

In order to enable a fair and informative comparison of unsupervised CCG parsers against the lexical categories and labeled dependencies in CCGbank, we define a simplification of CCGbank's lexical categories that does not alter the number or direction of dependencies, but makes the categories and dependency labels directly comparable to those produced by an unsupervised parser. We also do not alter the CCGbank derivations themselves, although these may contain type-changing rules (which allow e.g. participial verb phrases $\mathsf{S[ng]\backslash NP}$ to be used as NP modifiers $\mathsf{NP\backslash NP}$) that are beyond the scope of our induction algorithm.

Although the CCG derivations and dependencies that CCG-based parsers return should in principle be amenable to a quantitative labeled evaluation when a gold-standard CCG corpus is available, there may be minor systematic differences between the sets of categories assumed by the induced parser and those in the treebank. In particular, the lexical categories in the English CCGbank are augmented with morphosyntactic features, written in English, that indicate e.g. whether sentences are declarative ($\mathsf{S[dcl]}$), or verb phrases are infinitival ($\mathsf{S[to]\backslash NP}$). Prior work on supervised parsing with CCG found that the information contained in the features can be recovered by modeling

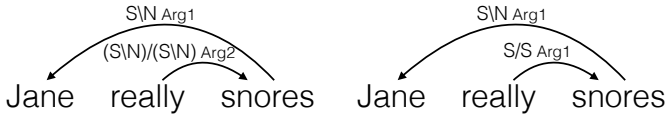
state-splitting with latent variables in the derivation [104]. Since we wish to evaluate a system that does not aim to induce morphosyntactic features, we remove the features from the evaluation. We also remove the distinction between noun phrases (NP) and nouns (N). CCGbank very often uses a unary type-changing rule $\mathsf{NP} \rightarrow \mathsf{N}$ to transform a bare N into a noun-phrase, so there are no syntactic ramifications to simplifying NP to N.

Finally, CCGbank distinguishes between sentential modifiers (which have categories of the form S|S, without features) and verb phrase modifiers (which take the form (S\NP)|(S\NP), again without features). But since the NP argument slot of a VP modifier is never filled, we can maintain the same number of gold standard dependencies by removing this distinction and changing all VP modifiers to be of the form S|S, with slash direction preserved. Maintaining the same number of dependencies and parses is made possible by higher-order composition:



Uses Application Requires Composition

These two parses result in the same set of directed edges. The only difference between the two graphs is the label on the arc attaching *really* and *snores*:



With these three simplifications we eliminate much of the detailed knowledge required to construct the precise CCGbank-style categories, and dramatically reduce the set of categories without losing expressive power. Table 3.1 shows the number of unique CCGbank categories with and without our simplifications. The complete set of categories decreases by more than a factor of three.

This simplification is consistent with the most basic components of CCG and can therefore be easily used for the evaluation and analysis of any weakly or fully supervised CCG system, not just that of our work. An example

	CCGbank	w/out Feats	Simplified
All	1640	458	444
Lexical	1286	393	384

Table 3.1: Category types in CCGbank 02-21

simplification is present in Figure 3.2. Similar simplifications should also be possible for CCGbanks in other languages.

There are two other simplifications we could have made which would have inflated the performance of our models, but we chose not to because doing so would remove necessary semantic categories (used in Chapter 9) or would be conflating the argument/adjunct distinction in prepositional phrases.

Modals and Auxiliaries Categories of the form $(S[\cdot]\backslash NP_i)/(S[\cdot]\backslash NP_i)$, which are used e.g. for modals and auxiliaries, are changed to $(S\backslash N_i)/(S\backslash N_i)$, not $S|S$ in order to maintain their non-local dependency on the subject (Section 7.2.3).

PP Arguments CCGbank differentiates prepositional phrases being used as arguments from those which are adjuncts by giving arguments the category PP. This requires prepositions to have the category PP/NP, while adjuncts take the categories (NP\NP)/NP or ((S\NP)\(S\NP))/NP) as discussed in Section 3.2.

	Our simplification of CCGbank's lexical categories					
	Congress	*has*	*n't*	*lifted*	*the*	*ceiling*
Original	NP	(S[dcl]\NP)/(S[pt]\NP)	(S\NP)\(S\NP)	(S[pt]\NP)/NP	NP[nb]/N	N
Simplified	N	(S\N)/(S\N)	S\S	(S\N)/N	N/N	N

Table 3.2: We remove morphosyntactic features, simplify verb phrase modifiers, and change NP to N.

Chapter 4

Inducing a Categorial Grammar

Before defining probability models over grammars, we must define the space of grammars that can be weighted or learned. In supervised parsing, the grammar is provided via a treebank, and licenses the space of parses for any given string. The rules of the grammar either take the form of a Context-Free Grammar or a Dependency Grammar.

A Context-Free Grammar (section 2.1.2) requires a set of rules that specify how non-terminals combine: S $\rightarrow$ NP VP. Depending on the word order and syntax of a language the types of rules and the ordering of the non-terminals will differ. Correspondingly, a Dependency Grammar specifies a set of labels on edges connecting words or part-of-speech tags: noun $\xleftarrow{nsubj}$ verb. In supervised parsing, the inventory of dependency labels and the words or tags that can be linked by any individual dependency label is given by the treebank. As our approach is unsupervised, we will not have access to grammars of either form, so we need a way to enumerate a set of possible parses for a sentence given only the part-of-speech tags (we will weaken this assumption later in Chapter 8).

Dependency-based approaches to grammar induction assume a trivially over-general space, the fully connected graph between words. Since they do not aim to predict labeled dependencies, the task for their model simply reduces to predicting which edges belong in the tree.

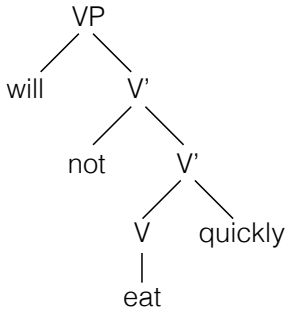
Because other approaches are not interested in recovering labeled structures, a fully connected graph is a simple and sufficient representation. If we were to attempt to perform grammar induction with a Context-Free Grammar, we would need a set of possible non-terminals (e.g. NP, VP, S, PP, etc.) from which we could generate an overly complete set of rules for parsing the

Work in this chapter was first published in Y. Bisk and J. Hockenmaier, "Simple Robust Grammar Induction with Combinatory Categorial Grammars," in *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence (AAAI-12)*, Toronto, Canada, July 2012, pp. 1643-1649 [112] and is reprinted here with permission by the copyright holder.

language.

$$
\begin{aligned}
\text{S} &\rightarrow \text{NP VP} \\
\text{S} &\rightarrow \text{NP PP} \\
\text{S} &\rightarrow \text{NP S} \\
\text{S} &\rightarrow \text{NP NP} \\
&\ldots
\end{aligned}
$$

This approach would enable us to search over labeled constituents. Constituents alone do not specify information about dependencies. For a CFG to encode headedness, every binary rule must specify whether to propagate the head from the left or the right constituent. This follows from the basic definition of X-bar grammar [105] which assumes that any phrase can contain intermediate constituents that propagate the head. This becomes clear with a simple example:



A CFG must therefore indicate whether each derivation of a constituent propagates a head from the left or the right child. In this case, the head of the final VP is still the word *eat* so we must propagate from the left once and then twice from the right. We outline this approach here to indicate that inducing head-annotated context-free grammars, may, in fact, be possible, but different constraints and seed knowledge would be necessary.

CCG provides a representation that has the strengths of both constituency and dependency parsing while capturing fine-grained linguistic information in the categories. What is missing is a mechanism for automatically producing the space of these linguistically informative categories.

In this chapter, we introduce just such a process which will use only initial knowledge of the atomic categories S, N, and conj to automatically construct an expressive and overly-general space of lexical categories for parsing text in any language. We will introduce two procedures for defining this space and then learn models to refine the space in future chapters.

4.1 Seed Knowledge

We assume that a categorial grammar can be defined in terms of just two atomic categories, N (nouns or noun phrases) and S (sentences), a special conjunction category conj, and a special start symbol TOP. We assume that all strings can be parsed to create either a noun phrase or a sentence:

$$\text{TOP} \rightarrow \text{N} \qquad\qquad \text{TOP} \rightarrow \text{S}$$

We do not allow any input strings to be parsed as both a noun phrase and a sentence. We enforce this by only allowing the rule TOP $\rightarrow$ N to fire if there is no verb in the sentence. A slightly looser restriction will be imposed in Chapter 8 where the definition of a verb is unclear. In that context, we modify this constraint to only allow TOP $\rightarrow$ N if no parse can result in an S. This provides an important bias for the grammar to prefer analyzing verbs with categories resulting in S.

Secondly, we assume that POS tags can be grouped into four classes: nominal, verbal, coordination, and other. This allows us to create an initial lexicon that only contains entries for atomic categories, e.g. for the English Penn Treebank tag set [1] or Universal POS tags [2]:

N	: {NN, NNS, NNP, PRP, DT[1]}	N	: {NOUN, PRON, NUM}
S	: {MD, VB, VBZ, VBG, VBN, VBD}	S	: {VERB}
conj	: {CC}	conj	: {CONJ}

We construct these mappings using either the Universal POS tags for a language or the annotation guidelines when the mapping is ambiguous. In this way, given a mapping from the tags of a language to UPOS we can propagate the information from our basic lexicon above to any new tagset. Where possible we deviate slightly from this initial lexicon to only allow coordinating conjunctions to take the category conj and leaving subordinating conjunctions to be learned by the system (language specific seed knowledge is provided in the Appendices).

[1] In early experiments we found the model performed best when DT was allowed to act as a noun. In all language beyond English we used the UPOS mapping (right) eliminating this anomaly.

4.2 Category Induction Algorithm

As discussed previously, CCG categories are either atomic categories (e.g. S, N, ...) like those in our seed lexicon or recursively defined functions (e.g. N/N, S\N, ...). In our induction algorithms, we will create categories of this form automatically from the seed knowledge subject to a handful of constraints (Section 4.2.2). These complex categories are necessary, since the initial lexicon would only allow us to parse single word utterances (or conjunctions thereof). The lexicon for atomic categories remains fixed, but all POS-tags will be able to acquire complex categories during induction.

In our discussion we use the following terminology when referring to categories: atomic, modifier, argument taking. Atomic categories are the most basic units of the grammar. They have no slashes or recursive structure: S, N, ...

Next we have modifiers. These are categories that have the same argument and return type. For example, an adjective has the category N/N because it both consumes a noun and returns one. To denote any possible category and any slash direction we write this as X|X or (X|X)|(X|X). (X|X)|(X|X) is a modifier of a modifier. This is used in cases like the adverb "very small." Additionally, in the case of modifiers of modifier the directions of the first and third slash must match for the category to have the same argument and return type.

Finally, argument taking categories are those whose argument and result are different: X|Y where $Y \neq X$. We will modify these definitions slightly in Chapter 7.2.2 when discussing auxiliary verbs. They may take the form we have just described for modifiers ((S\N)/(S\N) for English), but they are argument taking categories. This will be discussed later in the thesis.

4.2.1 Basic Induction Procedure

We will start off with the simplest induction algorithm. Induction is an iterative process. In each iteration the set of categories introduced and their complexity grows. In each round, the arity of categories increases by one and correspondingly, our ability to parse the data improves. In practice, in the English and Chinese CCGbanks it is rare for categories to have an arity greater than four (Table 4.1). We will now step through up to three rounds of

	English				Chinese			
Arity	Types	C%	Tokens	C%	Types	C%	Tokens	C%
0	23	1.8	348687	37.5	16	1.6	249618	40.4
1	128	11.8	306111	70.4	72	9.0	147837	64.4
2	443	46.3	224485	94.7	227	32.3	184051	94.2
3	487	**84.2**	48270	**99.9**	379	**71.1**	34083	**99.7**
4	175	97.8	1894	100.0	206	92.1	1761	100.0
5	27	99.9	75	100.0	60	98.3	130	100.0
≥ 6	1	100.0	1	100.0	17	100.0	22	100.0

Table 4.1: Distribution over the CCG category arities from the training sections of the English and Chinese CCGbanks. The raw counts are provided as well as the cumulative distribution. While the category token columns show the long tail of complex categories, the counts by token indicate the rarity of these complex categories in the corpus.

the induction process on a few simple sentences to show how new categories can be introduced automatically given our seed knowledge.

To parse a sentence $S = w_0...w_n$, all words $w_i \in S$ need to have lexical categories that allow a complete parse (resulting in a constituent TOP that spans the entire sentence). Initially, only some words will have lexical categories:

The	*man*	*ate*	*quickly*
DET	NOUN	VERB	ADV
-	N	S	-

This leaves us with no lexical categories for the determiner and adverb. To remedy this, we assume that any word may modify adjacent constituents (by taking the form X|X). For example, because *The* is adjacent to *man* and *man* has the category N, *The* will be allowed to modify *man* by taking the category N/N. This is a modifier because its argument and return type match and the forward slash indicates that it is modifying a word to its right.

When this is applied to the entire sentence, we introduce many modifying categories:

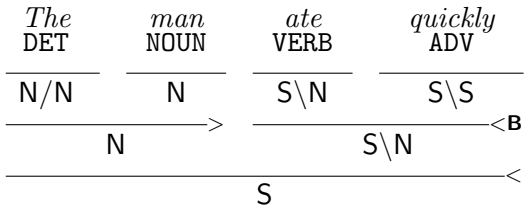
The	*man*	*ate*	*quickly*
DET	NOUN	VERB	ADV
N/N	N, S/S	S, N\N	S\S

Next, we assume that any category other than N (which we postulate does not take any arguments) can take any adjacent non-modifier category as an

argument. Here, the S assigned to the verb is adjacent to an N which it can take as an argument, allowing us to introduce the new category S\N for the verb:

The	*man*	*ate*	*quickly*
DET	NOUN	VERB	ADV
N/N	N, S/S	S, N\N, S\N	S\S

In the case of this particular sentence, this is sufficient for obtaining the one and only correct parse:

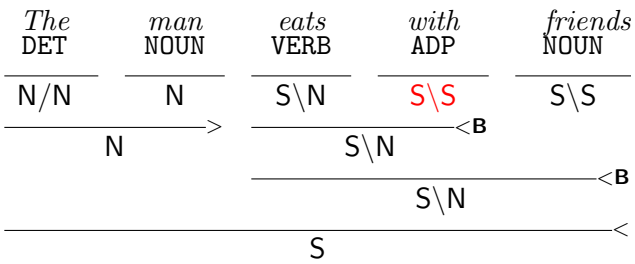


In general, we will require additional rounds of induction to increase the lexicon before we will be able to complete a parse or introduce the correct categories, leading to a very large and ambiguous parse forest. To do this we take the categories introduced in round one and update the lexicon with all new tag-category pairs:

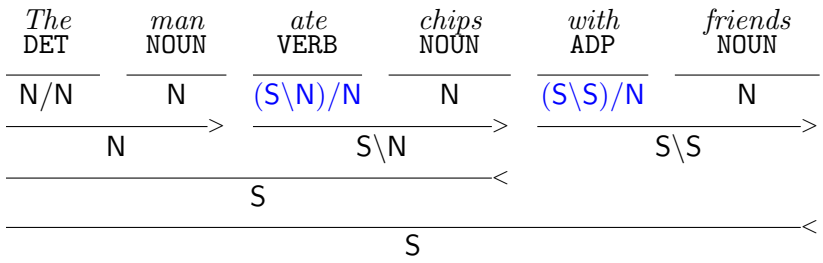
	DET	NOUN	VERB	ADV
Round 0:		N	S	
Round 1:	N/N	S/S	S\N	S\S
			N\N	

The first stage of induction can only introduce functors of arity 1, but many words, such as prepositions or transitive verbs, require more complex categories. Without them, for many sentences we will only be able to produce incorrect parses such as:

Incorrect Preposition Category



If we use the lexicon learned in the first round of induction as the input to a second round, we can discover additional simple categories, as well as more complex categories. For example, we can now introduce transitive verb categories and the correct preposition:[2]



The simple procedure we have just described will be the primary workhorse for the thesis and results therein, despite its incredible simplicity.

Pseudocode

The implementation of our approach is very simple. We provide pseudocode here for running induction across a corpus of part-of-speech tagged text (Algorithm 1). The approach is very basic in that it is a brute force search for new categories over the set of two word contexts in the corpus. Given a pair of POS-tags, left and right, the algorithm tries to find a way for them to modify each other or take one another as an argument. For readability, we separate the category creation into Algorithm 2. Finally, we refer to the function *valid*() to test if the proposed category violates any of our induction constraints (Section 4.2.2).

[2]Our constraints do not allow us to introduce the redundant category $(S/N)\backslash N$. This is discussed in Section 4.2.2.

Algorithm 1: Basic CCG Category Induction Algorithm

Data: Array of part-of-speech tagged sentences
Data: Seed Knowledge to POS-tag map
Result: CCG Lexicon per POS tag
`// Initialize Lexicon with seed knowledge`
Lexicon $\leftarrow$ {};
foreach *tag in Mapping* **do**
    Lexicon[*tag*] $\leftarrow$ Mapping[*tag*];
end
newLexicon $\leftarrow$ {};
`// Perform` n `rounds of induction`
for $r = 1$ **to** n **do**
    foreach *s in sentences* **do**
        for *i=1* **to** *len(s)* **do**
            `// First word in the sentence`
            if $i == 1$ **then**
                InduceRight(Lexicon[$s[i]$], Lexicon[$s[i+1]$]);
            `// Last word in the sentence`
            else if $i ==$ *len(s)* **then**
                InduceLeft(Lexicon[$s[i]$], Lexicon[$s[i-1]$]);
            `// Otherwise`
            else if $Mapping[s[i]] \neq$ `conj` **then**
                InduceLeft(Lexicon[$s[i]$], Lexicon[$s[i-1]$]);
                InduceRight(Lexicon[$s[i]$], Lexicon[$s[i+1]$]);
            end
        end
    end
    `// Update the lexicon with the new categories`
    Lexicon $\leftarrow$ Lexicon $\cup$ newLexicon;
end

Algorithm 2: InduceRight algorithm. The algorithm attempts to take the category on the right as an argument or to create a modifier for it. InduceLeft follows analogously.

```
Data: Left POS-tag/constituent's categories
Data: Right POS-tag/constituent's categories
Result: New categories for left POS-tag.
foreach R in Right do
    foreach L in Left do
        // Can the left cat take the right cat as an argument?
        if valid(L/R) then
            newLexicon[Left].append(L/R)
        end
    end
    // Can the right category be modified?
    if valid(R/R) then
        newLexicon[Left].append(R/R)
    end
end
```

4.2.2 Induction Constraints

One way we can limit the size of the induced lexicon is via constraints on the types and shapes of categories that can be introduced. The goal is to restrict the introduction of categories that are redundant or nonsensical, without cutting into those needed by the 15 languages we learn in this thesis. These constraints are imposed in all three of our induction algorithms: Section 4.2.1, 4.2.4, and 4.2.5.

1. Nouns (N) do not take any arguments This does not prohibit modifier categories (N/N) but it does prohibit nouns taking sentences or prepositional phrases (not present in our grammar) as arguments: N/S. These are very rare constructions.

2. The heads of sentences (S|...) and modifiers (X|X, (X|X)|(X|X)) may take N or S as arguments. In contrast to rule 1, S and modifier categories can take arguments. The case of S taking arguments is perhaps obvious as this is what produces verb categories: S\N, (S\N)/N, and so forth.

This is particularly important for preposition categories. Because our approach does not include a PP category, all prepositions are analyzed as adjuncts.

	president	*of*	*France*
Allowed:	N	(N\N)/N	N
Disallowed:	N/PP	PP/N	N

To get this analysis, we assume that modifiers (categories of the form X|X) may take arguments. For the English preposition, this will result in the categories (S\S)/N and (N\N)/N. Here, X can range over atomic categories and modifiers: X|X ⇒ S\S, S/S, (S\S)/(S\S), etc. In our discussion, we write modifier of modifiers with vertical slashes (X|X)|(X|X), but the result (X|X) and the argument (X|X) have to be (instantiated to) the same categories. The first and third (the result and argument) must match. Later in the thesis we will discuss categories of the form (S|N)|(S|N), which will not be treated as modifiers.

3. Sentences (S) may only take nouns (N) as arguments. (We assume S\S and S/S are modifiers). To limit ambiguity in the grammar we assume that every lexical category S|S is a modifier. Because modifiers have the opposite head direction of other argument taking categories, without this restriction S/S S could combine to be either head left or head right. This ambiguity only increases in the length of the verb-chain.

4. The maximal arity of any lexical category is 3. Under any of the schemes we have presented for category induction, categories can grow to be arbitrarily complex. In particular, for n rounds of induction, we may introduce categories that take n arguments. Recall that the number of arguments a category takes is known as its arity. We will restrict all categories to have at most arity 3. Additionally, categories which contain a modifier will be restricted to arity 2. This prohibits categories of the form ((X|X)|(X|X))|((X|X)|(X|X)) (arity 3), but does allow nearly every arity three verb: (((S|N)|N)|N), subject to constraint 5.

5. Since (S\N)/N is completely equivalent to (S/N)\N, we only allow the former category. There are four categories for transitive verbs (i.e. categories that take two arguments of type N and yield the result S):

$$(S\backslash N)\backslash N \quad (S\backslash N)/N \quad (S/N)\backslash N \quad (S/N)/N$$

Though there are four forms, there are only three verb placements captured here: verb final ($(S\backslash N)\backslash N$), verb initial ($(S/N)/N$) and where the verb sits between its arguments ($(S\backslash N)/N$ and $(S/N)\backslash N$). To reduce spurious ambiguity in the lexicon, we eliminate $(S/N)\backslash N$ in favor of the more traditional $(S\backslash N)/N$. We have not experimented with removing this constraint to evaluate whether the ambiguity is detrimental to the learner. Finally, we disallow categories that add arguments to $(S/N)\backslash N$. For example, this means that three N argument ditransitives can not take the form $((S/N)\backslash N)|N$.

6. Coordinating Conjunctions are restricted to conj if not sentence initial or final. Additionally, conj can neither take arguments nor be taken as one. This is because coordination uses a special ternary rule, and is therefore outside the regular CCG calculus. If a sentence starts or ends with a conjunction (e.g. *"And then..."*), the other half of the coordination is in another sentence. For this reason, we allow conjunctions to induce and use modifier categories in these special cases.

7. Disallow $(X/X)\backslash X$ to reduce ambiguity. In our initial experiments, we will keep the grammar small by only allowing $(X\backslash X)/X$. We will remove this restriction in our final models (Section 6.4.1 & 7.2.1).

4.2.3 Failings of Category Induction

It is important to note that any induction procedure (even with constraints like described above) would likely introduce a large number of unnecessary categories, such as complex modifiers of the form $(X/X)|(X/X)$ or $(X\backslash X)|(X\backslash X)$, e.g.:

The	*man*	*ate*	*very*	*quickly*
DET	NOUN	VERB	ADV	ADV
N/N,	N, S/S	S, N\N,	S\S,	S\S,
(S/S)/(S/S)	(N\N)/(N\N)	S\N	(S\S)/(S\S)	(S\S)\(S\S)
	(N/N)\(N/N)	(S/S)\(S/S)	(N\N)\(N\N)	
		(S\S)/(S\S)		

It will be the task of the probability model to identify which of the categories proposed by the induction algorithm should actually belong to a languages lexicon.

Our basic induction procedure fails in several ways:

1. It does not consider constituent adjacency (Section 4.2.4 & 4.2.5)
2. It vastly overgenerates (Section 4.2.2)
3. It adds arguments in the wrong order for the CCG calculus (Section 4.2.5)

Intuitively it seems to follow that addressing any of these concerns would allow for the induction algorithm to constrain the learning to benefit the grammar induction process. Unfortunately, our attempts to do so were unsuccessful. We will describe two such approaches now, although they are not necessary for understanding the main results of this thesis. The anxious reader can comfortably skip to Section 4.2.2.

4.2.4 Constituent-Based Induction

The first insight not addressed by our basic induction algorithm is that grammars capture dependencies between constituents that are not necessarily string-adjacent. Therefore, it seems natural to extend our existing procedure to taking adjacent constituents as arguments. For example, by completing a partial parse below, we find that *a friend* can combine to N. This allows us to induce the correct preposition category, (S\S)/N, for *with*.

The	*man*	*eats*	*with*	*a*	*friend*
DET	NOUN	VERB	ADP	DET	NOUN
N/N S/S	N, S/S	S, N\N, S\N	S\S (S\S)/N	N/N, S\S, S/S	N, S\S
	S			N	

While at first it appeared this step was necessary to recover the correct lexical categories (Ch. 5), this turned out to be an artifact of only using short sentences for training. When induced categories are shared across the corpus between rounds of induction on longer sentences, the coverage gains introduced by induction over constituents are eliminated. Specifically, it is

very likely that there is another sentence in which ADP is next to NOUN and can, therefore, introduce the necessary (S\S)/N category to be used in this sentence.

If we had a language in which the verb was never adjacent to a noun but was always separated by a determiner, the constituent induction would be necessary to induce the correct category. The success of our simple procedure might be an artifact of performing induction over part-of-speech tags. Were induction carried out on a per-word basis, rather than a per-tag basis, it is possible that determiners would need to be dealt with more intelligently. In such a case, the previous induction algorithm (with and without constituents) would be incorrect as it incorrectly introduces arity 3 or higher categories (a correct induction procedure is presented in section 4.2.5).

The only change necessary to the existing pseudocode to accommodate constituent-based induction is parsing the text between rounds and iterating over all pairs of adjacent spans where one entry in the pair is lexical (Algorithm 3). One downside of this approach is that it requires the extra computation time of parsing the sentence, which dramatically slows the algorithm.

Algorithm 3: Constituency Based CCG Category Induction Algorithm. This algorithm includes a parsing step where all constituents allowed by the current lexicon are used for induction.

```
Data: Array of part-of-speech tagged sentences
Data: Seed Knowledge to POS-tag map
Result: CCG Lexicon per POS tag
// Initialize Lexicon with seed knowledge
Lexicon ← {};
foreach tag in Mapping do
    Lexicon[tag] ← Mapping[tag];
end
newLexicon ← {};
// Perform n rounds of induction
for r = 1 to n do
    foreach s in sentences do
        // Assume chart is a 2D array which when indexed returns
           all categories for the given constituent.
        chart = CKY(s, Lexicon);
        n = len(s);
        for i=1 to n do
            for j=1 to n-i+1 do
                for k=1 to i-1 do
                    // Lexical Constituent in on the left
                    if k == j then
                        InduceRight(Lexicon[s[k]], chart[i − k][j + k]);
                    end
                    // Lexical Constituent in on the right
                    if i-k == j+k then
                        InduceLeft(chart[k][j], Lexicon[s[i − k]]);
                    end
                end
            end
        end
    end
    // Update the lexicon with the new categories
    Lexicon ← Lexicon ∪ newLexicon;
end
```

4.2.5 A Corrected Induction Algorithm

Thus far we have assumed that new categories are introduced by appending a new argument to an existing category. Our approaches thus far do not handle the case where a word needs to take two arguments A, B that have different categories, but appear on the same side of the word. To handle this case, and to correct argument ordering problems with our current induction scheme, we can create an induction algorithm that propagates arguments taken by a constituent down to the lexical category. To see this, let us trace the induction of a simple ditransitive verb. First we can compare the correct lexical category for the verb with what our induction algorithm introduces:

Correct Lexical categories:

	I	*told*	*her*	*that ...*
R0	N	((S\N)/S)/N	N	S

Our Current Induction Procedure:

	I	*told*	*her*	*that ...*
R0	N	S	N	S
R1		S\N		
R2		(S\N)/N		
R3		((S\N)/N)/N		

In this context, we have introduced the wrong category and one which will not allow us to find the correct derivation. The reason for this is because the two arguments taken to the right of *told* have different categories: N and S. For this reason, the naive algorithm's use of only the adjacent categories will not be able to introduce a category that takes an S argument in this sentence. To address this, we must build partial parses of the sentence and then infer the correct lexical categories.

Corrected Induction Procedure: First the verb takes its indirect object (right) which allows it to form a new constituent: S : *told her*.

	I	*told*	*her*	*that...*			*I*	*told*	*her*	*that...*
R0	N	S	N	S	⇒	R1		S/N	N	
R1		S/N						S		

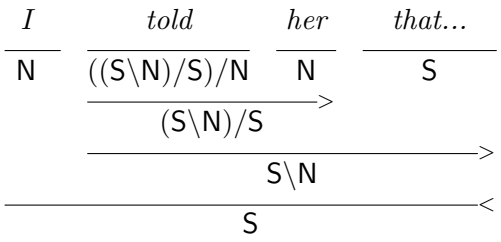
Next the verb phrase takes a direct object (*that...*):

	I	*told her*	*that...*			*I*	*told her*	*that...*
R1	N	S	S	⇒	R2		S/S	S
R2		S/S					S	

Finally, the category must take its subject:

	I	*told her that...*			*I*	*told her that...*
R2	N	S	⇒	R3	N	S\N
R3		S\N				S

Now let us trace through the derivation. In round 1, we produced an S spanning *told her*. This category then took an argument S in round 2. Since it was only possible for it to take S after consuming N we can propagate the argument into the category from round 1, S/N, to create a new category for *told*: (S/S)/N. We can then repeat this for the constituent *told her that...* which took a category N to produce the correct lexical category for *told*: ((S\N)/S)/N.



This new category preserves the derivation order and takes S as its second argument, unlike our current induction algorithm that produced the category ((S\N)/N)/N. In practice, we share categories between sentences between each round, and perform induction over part-of-speech tags, and, therefore, induce all of the same categories that otherwise require this correct induction scheme and so we will not report results using it. Despite this, we do feel that future work that performs induction on individual words or within the sentence will benefit from this corrected approach. The only change to the induction code is within the InduceRight and InduceLeft code, to perform the argument propagation.

4.3 Existing Techniques for Injecting Knowledge

What we have illustrated is a series of ways in which to generate a large set of Combinatory Categorial Grammar categories. Again, in the case of a CFG, we would need to define the set of non-terminals and take their cross product to define the space. In the case of a dependency grammar, the unlabeled edges of a fully connected graph define the search space. In both cases, the space is highly ambiguous.

Practically speaking, it is tempting to constrain the space of constructions with a small amount of knowledge about language broadly (in the form of universal constraints) or the specific language we are trying to parse. At the same time, there is an interesting question about how the grammar of a language can be learned and what knowledge a child or machine should have access to. The more annotation/supervision provided, the more expensive the system is to produce and the less we learn about the intrinsic information content of the string. Supervision can range from none, in the form of raw text, to providing the exact grammar of a language, via an annotated treebank.

We will briefly discuss a few points on this spectrum from most to least supervised. We will start with two CCG-based approaches.

Knowledge from a Treebank Garrette et al. [103] assume they have access to most of the lexical categories of a language and must only learn their attachments. While this is cheaper than using a treebank, it does require that a CCG trained linguist is available to annotate a large body of text. The intuition for this approach relies on exploiting the uniquely informative nature of CCG categories and their relative ease of annotation.

Knowledge from a Linguist Boonkwan and Steedman [38] also assume knowledge of a CCG lexicon, but avoid dependence on a treebank by using a simple questionnaire. By laying out a series of questions about the types of constructions allowed by a language and fundamental properties like the language's word order, they only require a few hours with a linguist to quickly create a CCG lexicon. In this way, the linguist does not need to be trained in CCG, but a CCG lexicon can still be recovered from their answers.

Universal Knowledge and Prototypes In contrast, Naseem et al. [37] try to move away from having language specific information by relying on "universal" knowledge. Their work is with dependency grammars which allows them to easily bias certain attachments (nouns as children of verbs, adjectives modifying nouns, etc.). In this way, the language specific information is simply having access to a correctly tagged sentence. We should briefly mention that there are many other approaches [106, 107] that similarly attempt to provide prototypical information to their systems with varying success.

Our Seed Knowledge Among these approaches, we believe our work falls closest to being fully unsupervised by only providing information about nouns, verbs, and coordination conjunctions. Rather than encode attachment preferences of a dozen or more different linguistic categories that may or may not exist in any given language, we choose what we believe to be a minimal universal set that exist in all languages: Nouns (N) and Verbs (S). Their universality appears to be corroborated by the psychology literature [108, 109, 110, 111] of child language learning.

Finally, we should mention that all of these approaches assume some knowledge of the part-of-speech tags. For example, we use the tags to initiate our induction algorithm with seed knowledge. Being able to attach knowledge to these clean syntactic classes is a form of supervision present in all of these approaches. Later in this thesis we will remove gold part-of-speech tags from our system and replace them with induced clusters and a small set of labeled words. This lessens our reliance on supervision, but acquiring this seed knowledge automatically would require semantics from the world. For example, were we to investigate language learning as a robot, we would expect the class of nouns to be grounded in physical observations and verbs in actions. Finding a clean and naturalistic source of this supervision is of immediate interest for future work.

4.4 Ambiguity in the Induced Lexicons

Every induction algorithm will produce an overly general grammar. In particular, our simplest approach will match nearly every possible category to every part-of-speech tag. Further, because parsing and induction are per-

formed with part-of-speech tags, only looking at the subset of the lexicon successfully used to parse the corpus does not restrict the lexicon (unless parsing is only performed on short sentences).

As the goal of our induction algorithm is to define the search space for our probabilistic models, there are several knobs both in the induction algorithm and parsing algorithm that can be tuned to limit or increase coverage and ambiguity.

4.4.1 From Lexicons to Parse Forests

Given our induced CCG lexicon, we can use the CCG combinators (Section 3.1.2), and the CKY algorithm (Section 3.3.1) to exhaustively parse the corpus. To control the number of parses per sentence produced with a given lexicon, we can constrain the parser to use only a subset of the CCG combinators and only optionally allow type-raising. The result of running the CKY algorithm is a packed parse forest that will be used for training the models in subsequent chapters.

If only function application and composition of arity 1 categories is allowed, CCG expressivity is limited to capturing context-free languages. This bounds the parsing time at $O(G \times n^3)$ and drastically reduces the size of the parse forests. In contrast, generalized composition both slows worst case parsing to $O(n^6)$ and permits new ambiguity into the forest. We do not know what the "right" settings for the parser are, but we experiment with up to arity 3 composition and type-raising in the most general case.

We will place several restrictions on the way combinators are allowed to be used. First, we assume all parsing is completed under normal-form constraints (section 3.3.2). Second, within our experiments the arity of composition only applies to modifiers and type-raised categories. Specifically, we limit the arity of composition for type-raised categories to two, even in the case of B^3, and we do not allow for composition into modifiers.

Because CCG is a lexicalized grammar formalism the primary mechanism for ensuring a syntactic analysis is in the parse forest and model's search spaces is the induced lexicon. This fact also implies that the main source of ambiguity is also the overly general induced lexicons.

4.4.2 Visualizing Lexical Ambiguity

To appreciate how broad the lexicon is we introduce, we present the full set of induced categories used during parsing for our simplest, most constrained, setting (arity two categories with atomic arguments) for English in Table 4.2. The model's task is to choose the correct categories from this space to use when parsing.

The first column shows the pairing of categories (left) with tags (right). It should be clear from this column that the grammar is highly ambiguous and provides very little in the way of constraints on the learner. To visualize how the model constrains the effective lexicon, we train a model (B^1 in Section 7.2.1) and compute a new lexicon from the Viterbi parses (single best parse per sentence) on section 22 of the WSJ corpus. These are the only categories the model chooses to use, presented in column 2. They are a much smaller and more English-like set of categories than was induced.

Finally, rather than print the model's distributions, we visualize where the mass of the distributions is congregated, by showing a thresholded version of the Viterbi lexicon in column 3. These are the categories that comprise 95% of the lexical tokens, and the tags are those that make up 95% of the tokens per category. The goal of this column is simply a visual representation of pruning the tail phenomena which demonstrates how small the frequent lexicon is when compared to the original search space provided.

While a fair number of these initial (category, tag) pairs are used in English at least once in CCGbank, many are never used. We can use the treebank to quantify this ambiguity precisely as we increase the grammar's complexity.

4.4.3 Increasing Grammatical Complexity

To explore this further, we will analyze four settings of our original induction algorithm. We will run the algorithm for two or three iterations, and we will first try constraining the set of arguments to being atomic (S and N) and then broaden it to allow two complex categories ($\mathsf{S\backslash N}$ and $\mathsf{S/N}$). There are no changes required to the induction algorithms to induce categories with complex categories, beyond allowing them as possible arguments. These results are presented in Table 4.3. All of the induction is performed on sentences of up to length 20 of sections 2-21 of the WSJ, and we will evaluate on the development section 22.

Category	Induced Lexicon	Viterbi Lexicon	Most Common Categories Lexicon	
N/N	ADJ ADP ADV CONJ DET NOUN NUM PRON PRT VERB X	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ DET NOUN NUM PRON VERB	Tags that comprise 95% of the category's tokens.
N	DET NOUN NUM PRON X	DET NOUN NUM PRON X	NOUN PRON	
(N\N)/N	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADP NOUN PRT	
(S\S)/N	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ ADP ADV DET NOUN NUM PRON PRT VERB	ADP ADV DET PRT VERB	
(S\S)\(S\S)	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ ADP ADV DET NOUN NUM PRON PRT VERB	ADJ ADP ADV DET NUM VERB	
S\S	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ ADP ADV DET NOUN NUM PRT VERB	ADJ ADV PRT VERB	
S\N	VERB	VERB	VERB	
(S\S)/S	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ ADP ADV DET NOUN NUM PRON PRT VERB	ADP ADV DET NOUN PRT VERB	
(S\N)/S	VERB	VERB	VERB	
S/S	ADJ ADP ADV CONJ DET NOUN NUM PRON PRT VERB X	ADJ ADP ADV CONJ DET NOUN NUM PRON PRT VERB X	ADP ADV CONJ DET PRON VERB	
conj	CONJ	CONJ	CONJ	
(S\N)/N	VERB	VERB	VERB	
N\N	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ ADP ADV DET NOUN NUM PRON PRT VERB	ADJ ADV DET NOUN PRON	
S/N	VERB	VERB		Categories that occur (in aggregate) ¡5% of the time.
S	VERB	VERB		
(N\N)/(N\N)	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ ADP ADV DET NOUN NUM PRON PRT VERB		
(N\N)/S	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ ADP ADV DET NOUN NUM PRT		
(N\N)\N	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADP ADV NOUN NUM VERB		
(S\N)\N	VERB	VERB		
(N/N)/S	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ ADV NOUN VERB		
(N/N)/N	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ NOUN VERB		
(N/N)/(N/N)	ADJ ADP ADV CONJ DET NOUN NUM PRON PRT VERB X			
(N/N)\(N/N)	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X			
(N/N)\S	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X			
(N\N)\(N\N)	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X			
(N\N)\S	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X			
(S/N)/N	VERB			
(S/N)/S	VERB			
(S/S)/(S/S)	ADJ ADP ADV CONJ DET NOUN NUM PRON PRT VERB X			
(S/S)/N	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X			
(S/S)/S	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X			
(S/S)\(S/S)	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X			
(S/S)\N	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X			
(S\N)\S	VERB			
(S\S)/(S\S)	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X			
(S\S)\N	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X			
(S\S)\S	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X			

Induced categories which do not get used in a single Viterbi parse of Section 22.

Table 4.2: To provide a visceral sense of the ambiguity in induced lexicons, we show here the full set of induced categories, as paired with part of speech tags, produced by two rounds of induction with atomic categories (left column). The model's job is to prune this space. The second column the (category, tag) pairs used in Viterbi parses by our simplest model from Section 7.2.1. Finally, to visualize where the mass of the lexical distributions is focused, the categories used for 95% of the lexical tokens, and the tags that make up 95% of tokens per category, are shown in the third column.

As the (vast) majority of categories introduced for a tag will be incorrect, we want a sense of how large the set of candidate categories is per tag. We compute the total number of lexical categories introduced and how many on average that allows per part-of-speech tag. This value ranges from 26.4 to 56.4.

Given the significant increase in ambiguity, we assess the utility of introducing these new categories by computing the type and token based coverage of these categories in CCGbank. We find that as arity and complexity increase, the number of treebank tokens we can cover goes from 84.3% to 90.2%, but that this only accounts for 20.3% or 32.4% of the category types in the corpus. Additionally, even the most general lexicon only has a full sentence coverage of 66%. This means the correct analysis of a third of sentences it not within our search space. Many of these missing categories are required for complicated constructions. We should note that despite low coverage of the correct analyses, all of our configurations provide parses for 99.9% of the sentences in the development set.

Finally, and perhaps most interestingly for learning, we compute Type-based Precision, i.e. the percentage of the categories introduced that appear in the English CCGbank at least once. We see that as the grammar grows, this number drops precipitously from 81.1% to 36.1%. This means that in the most restrictive setting most categories introduced are valid for at least one English construction, while in the most general lexicon (necessary for coverage), the majority of categories are wrong. In contrast, we see a small but monotonic increase in Type-based coverage, i.e. the percent of English categories being entertained by the induction algorithm.

This analysis paints a rather bleak picture. It indicates both that our induced lexicons are insufficient to correctly parse the WSJ, and that our attempts to increase coverage so dramatically increase the size of the lexicon as to make it very difficult for the model to learn to use its categories correctly. One aspect missing from this analysis is the distribution of the categories in the corpus. In particular, all of the approaches appear to have high token-based coverage, which may indicate that we are recovering much of the necessary categories for English and are only missing tail phenomena.

To investigate this, we repeat our Type based Coverage and Precision analyses with varying category frequency thresholds for the most restricted (2A: Arity 2 with Atomic Arguments) and most general (3C: Arity 3 with

Size, ambiguity, coverage and precision of the induced lexicons				
Arguments:	**Atomic**		**Complex**	
# Lexical Arity:	**2**	**3**	**2**	**3**
# Lexical Categories	37	53	61	133
Avg. #Cats / Tag	26.4	29.5	42.3	56.3
Token-based Coverage	84.3	84.4	89.8	90.2
Type-based Coverage	20.3	21.6	27.0	32.4
Full Sentence Coverage	57.8	59.5	65.5	66.0
Type-based Precision	81.1	60.4	65.6	36.1

Table 4.3: We ran our original induction algorithm four times with different settings (two versus three rounds and with or without complex arguments). We report here a comparison of the size, ambiguity, coverage and precision (evaluated on Section 22) of the different induced lexicons. The full sentence coverage indicates the percent of sentences for which we have introduced all of the correct categories, but does not account for the use of type-changing rules which may be necessary to complete the parse.

	2A				3C			
%	90	95	99	100	90	95	99	100
Type Coverage	69.2	73.7	45.7	20.3	84.6	89.5	65.2	32.4
Type Precision	24.3	37.8	56.8	81.1	8.1	12.6	22.2	36.1

Table 4.4: We complement the analysis in Table 4.3 by investigating how type based precision and coverage change when tail phenomena are ignored by the analysis. Here we show results when the corpus categories are thresholded to the top 90, 95, 99 and 100%.

Complex Arguments) settings in Table 4.4. We sorted the categories by frequency and thresholded at 90%, 95%, 99% and 100% token coverage to see how the values change as we entertain more and more of the tail.

We see that when tail phenomena are ignored (left side of the table), the common phenomena are largely covered by induction, but the vast majority of induced categories are invalid. The situation then flips as we broaden to more complex constructions likely outside the ability of our models.

To give a better sense of these tail phenomena in the corpus and how peaked the distribution of categories is in the treebank, we present the number of categories (under our simplification) in CCGbank that meet various frequency thresholds and how many categories are required to meet different

	% Token Coverage				Freq Threshold			
	90	95	99	100	5	10	50	100
# Categories in 22	14	21	51	157	81	59	32	25
# Categories in 02-21	14	21	52	382	203	168	107	88
% Sentence coverage in 22	65	73	90	100				

Table 4.5: To analyze how the number of categories and sentence coverage drop off as a function of lexical category coverage, we present the number of categories that make various thresholds, both percentage (left) and counts (right). It becomes clear that sentence coverage on the development set falls quickly as tail phenomena are removed from the lexicon.

amounts of corpus token coverage (Table 4.5).

The last line of the table shows the percent of sentences in section 22 that can be correctly parsed using the categories that comprise 90, 95, or 99% of the token coverage. We see up to a 30% gap between token and sentence coverage.

In addition, to give a more visceral sense of scale as to the number of induced categories and how they are reduced by the models in later chapters, Table 4.6 shows the same three columns (Induced, Viterbi, 95%) as we saw earlier in Table 4.2, but for our most ambiguous setting (Arity 3 categories with complex arguments with B^3 from Table 7.5). To save space, the categories that were not used in any Viterbi parses are simply listed at the bottom of the table below their Universal Part-of-Speech (UPOS) tags.

4.5 Conclusions

We have laid out minimally supervised mechanisms for creating a CCG lexicon from seed knowledge. Because CCG is a lexicalized formalism, a tremendous amount of information about a language is encoded in its lexical categories. As such, it is important to ensure the procedures introduce the necessary categories for a language (e.g. Table 4.3). Our analysis of induced lexicons indicated that our procedure introduced the majority of English's common categories and all of the necessary categories to correctly parse a majority of the corpus. Unfortunately, we also found many English categories were missing from our search space and that most categories introduced were not valid English.

Category	Induced Lexicon	Viterbi Lexicon	Most Common Categories Lexicon
N/N	ADJ ADP ADV CONJ DET NOUN NUM PRON PRT VERB X	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ ADP DET NOUN NUM VERB
N	DET NOUN NUM PRON X	DET NOUN NUM PRON X	NOUN PRON
S\S	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ ADP ADV DET NOUN NUM PRON PRT VERB	ADJ ADP ADV NOUN NUM PRT VERB
(N\N)/N	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ ADP ADV DET NOUN NUM PRON PRT VERB	ADP NOUN PRT VERB
conj	CONJ	CONJ	CONJ
S/S	ADJ ADP ADV CONJ DET NOUN NUM PRON PRT VERB X	ADJ ADP ADV CONJ DET NOUN NUM PRON PRT VERB X	ADJ ADP ADV CONJ DET PRON PRT VERB
N\N	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ ADP ADV DET NOUN NUM PRON PRT VERB	ADJ ADV DET NOUN PRON VERB
(S\N)/N	VERB	VERB	VERB
S/N	VERB	VERB	VERB
((S\N)/N)/(S/N)	VERB	VERB	VERB
(S/N)/N	VERB	VERB	VERB
(S\N)/S	VERB	VERB	VERB
(N\N)/(S/N)	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADP ADV DET NOUN NUM PRON PRT VERB	ADP DET NOUN PRON PRT VERB
((S\N)/N)/N	VERB	VERB	
(N\N)/(N\N)	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ ADP ADV NOUN NUM PRT	
(S\N)/(S/N)	VERB	VERB	
(N\N)/S	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ ADP ADV DET NOUN PRON PRT VERB	
S\N	VERB	VERB	
(S\S)/S	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ ADP ADV DET NOUN NUM VERB	
((S/N)\(S/N))\N	VERB	VERB	
(S\S)/N	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ ADP ADV DET NOUN NUM PRT VERB	
(N/N)/S	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ NOUN NUM VERB	
(S\N)\N	VERB	VERB	
(N/N)/N	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADP NOUN VERB	
(N\N)\N	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	NOUN NUM	
(N\N)\S	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	NOUN	
(S/S)/N	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ DET NOUN VERB	
S	VERB	VERB	
(S\S)/(S/N)	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADV NOUN	
(N/N)/(S/N)	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADP ADV VERB	
(N\N)\(N\N)	ADJ ADP ADV DET NOUN NUM PRON PRT VERB X	ADJ	
((S\N)\(S/N))/N	VERB	VERB	

Unused ADJ ADP ADV CONJ DET NOUN NUM PRON PRT VERB X categories:
(S/S)/(S/S),(N/N)/(N/N)

Unused ADJ ADP ADV DET NOUN NUM PRON PRT VERB X categories:
(S\S)\S, (S\S)\N, (S\S)\(S\S), (S\S)\(S\N), (S\S)\(S/N), (S\S)/(S\S), (S\S)/(S\N), (S/S)\S, (S/S)\N, (S/S)\(S/S), (S/S)\(S\N), (S/S)\(S/N), (S/S)/S, (S/S)/(S\N), (S/S)/(S/N), (N\N)\(S\N), (N\N)\(S/N), (N\N)/(S\N), (N/N)\S, (N/N)\N, (N/N)\(N/N), (N/N)\(S\N), (N/N)\(S/N), (N/N)/(S\N)

Unused VERB categories:
(S\N)\S, (S\N)\(S\N), (S\N)\(S/N), (S\N)/(S\N), (S/N)\(S\N), (S/N)\(S/N), (S/N)/S, (S/N)/(S\N), (S/N)/(S/N), ((S\N)\S)\S, ((S\N)\S)\N, ((S\N)\S)\(S\N), ((S\N)\S)\(S/N), ((S\N)\S)/S, ((S\N)\S)/N, ((S\N)\S)/(S\N), ((S\N)\S)/(S/N), ((S\N)\N)\S, ((S\N)\N)\N, ((S\N)\N)\(S\N), ((S\N)\N)\(S/N), ((S\N)\N)/S, ((S\N)\N)/N, ((S\N)\N)/(S\N), ((S\N)\N)/(S/N), ((S\N)\(S\N))\S, ((S\N)\(S\N))\N, ((S\N)\(S\N))/S, ((S\N)\(S\N))/N, ((S\N)\(S\N))/(S\N), ((S\N)\(S\N))/(S/N), ((S\N)/S)\(S\N), ((S\N)/S)\(S/N), ((S\N)/S)/S, ((S\N)/S)/N, ((S\N)/S)/(S\N), ((S\N)/S)/(S/N), ((S\N)/N)\(S\N), ((S\N)/N)\(S/N), ((S\N)/N)/S, ((S\N)/N)/(S\N), ((S\N)/N)/(S/N), ((S\N)/(S\N))\S, ((S\N)/(S\N))\N, ((S\N)/(S\N))/S, ((S\N)/(S\N))/N, ((S\N)/(S/N))\S, ((S\N)/(S/N))\N, ((S\N)/(S/N))/S, ((S\N)/(S/N))/N, ((S/N)\(S\N))\S, ((S/N)\(S\N))\N, ((S/N)\(S\N))/S, ((S/N)\(S\N))/N, ((S/N)\(S/N))\S, ((S/N)\(S/N))\N, ((S/N)\(S/N))/S, ((S/N)\(S/N))/N, ((S/N)/S)\(S\N), ((S/N)/S)\(S/N), ((S/N)/S)/S, ((S/N)/S)/N, ((S/N)/S)/(S\N), ((S/N)/S)/(S/N), ((S/N)/N)\(S\N), ((S/N)/N)\(S/N), ((S/N)/N)/S, ((S/N)/N)/N, ((S/N)/N)/(S\N), ((S/N)/N)/(S/N), ((S/N)/(S\N))\S, ((S/N)/(S\N))\N, ((S/N)/(S\N))/S, ((S/N)/(S\N))/N, ((S/N)/(S/N))\S, ((S/N)/(S/N))\N, ((S/N)/(S/N))/S, ((S/N)/(S/N))/N

Table 4.6: We replicate the analysis from Table 4.2 using our most ambiguous lexicon and expressive grammar from Section 7.2.1, B^3 with type-raising for arity 3 categories with complex arguments. Column one shows the full set of induced categories, as paired with part of speech tags. The second column shows pairs used in Viterbi parses. Finally, the categories used for 95% of the lexical tokens, and the tags that make up 95% of tokens per category, are shown in the third column.

Having defined the space of lexical categories and, correspondingly, the space of syntactic parses, we switch to creating models for scoring these parse forests and choosing the correct analysis. We already have a qualitative sense from Tables 4.2 and 4.6 that the models will produce an effective lexicon that is drastically smaller than was induced. Next, we lay out the details of these models and perform quantitative multilingual analyses.

Chapter 5

A Baseline PCFG Model

Having introduced a mechanism for creating a grammar from seed knowledge with varying amounts of expressivity and ambiguity, we can parse a large body of text to produce many possible analyses for every sentence. Our task now is to learn statistical models capable of choosing the correct analysis for a sentence from among the full set licensed by the grammar. The thesis will compare two models for scoring the parse forests: A PCFG (this chapter) and the HDP-CCG model (next chapter).

A Probabilistic Context-Free Grammar (PCFG) [22] is defined by providing a probability to every rule in a context-free grammar such that the children are conditioned on the parent: $p(B\ C|A)$ in such a manner that:

$$\sum_{B'C'} A \rightarrow B'\ C' = 1$$

We will factorize this representation slightly in the next section. The PCFG is an important baseline because the model has no knowledge of CCG's combinators or of the internal structure of CCG categories.

For both models, after constructing the lexicon, we parse the training corpus and use the Inside-Outside algorithm [113], a variant of the Expectation-Maximization algorithm [114] for probabilistic context-free grammars, to estimate model parameters. In supervised parsing, the parameters of every distribution can be estimated by counting the frequency of constructions in the treebank. This means that very complex models can be estimated accurately to capture complex phenomena and reproduce them on test data. In

Work in this chapter was first published in Y. Bisk and J. Hockenmaier, "Simple Robust Grammar Induction with Combinatory Categorial Grammars," in *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence (AAAI-12)*, Toronto, Canada, July 2012, pp. 1643-1649 [112] and Y. Bisk and J. Hockenmaier, "Induction of Linguistic Structure with Combinatory Categorial Grammars," in *NAACL HLT Workshop on Induction of Linguistic Structure*, Montreal, Canada, June 2012, pp. 90-95 [118] and is reprinted here with permission by the copyright holder.

contrast, when the parameters of a model are estimated in an unsupervised fashion, the estimation procedure has only access to the space of all possible structures for each training sentence, but is not given any information about which structures are correct. As a result, the optimization problem becomes highly non-convex, making it susceptible to local optima and quirks of initialization. Even in the unsupervised case, training is plagued by issues of data sparsity, leading to insufficient support for reliable parameter estimates. For this reason, we must focus on simple models that minimize the number of parameters while still capturing important properties of the grammar.

While issues like data sparsity also plague supervised and semi-supervised approaches, it is important to spend a minute to understand what the model is being tasked with when learning in our setup. In the supervised setting, parses are provided for every sentence, and the treebank's creators define the grammar. In this setting, a model with a large number of parameters may have very little support for any particular attachment decision or when modeling bi-lexical dependencies, but the the little data it has is correct. In contrast, in our domain the vast majority of parses are incorrect, models must therefore be constructed to search for very general and stable patterns across the parse forests in the hope that these general properties like word order or branching directions are consistent enough to be deciphered from the noise.

Similarly, a semi-supervised setting has analogous benefits to aid learning. A semi-supervised approach to CCG parsing (like [38, 103] or Chapter 9.5) specifies the lexical categories of the language and therefore highly constrains the set of analyses. This supervision makes learning much simpler as the parse forest has been constructed to largely have the correct structure and type of relations.

Before exploring more sophisticated modeling solutions that involve Hierarchical Dirichlet Processes and a novel factorization for CCG based models (Chapter 6), we will describe a set of experiments based on PCFGs.

5.1 A CFG Factorization

We use the baseline model of Hockenmaier and Steedman [98], which is a simple generative model that is equivalent to an unlexicalized PCFG. In a

CFG, the sets of terminals and non-terminals are disjoint, but in CCG most categories will be both lexical and associated with complex constituents.

Since this model is also the basis of a lexicalized model that captures word-word dependencies (i.e. which words tend to be arguments or modifiers of which other words), it distinguishes between lexical expansions (which produce words), unary expansions (which are the result of type-raising or the TOP rules), binary expansions where the head is the left child, and binary expansions where the head is the right child. Distinguishing the head direction will differentiate otherwise identical parses, capturing some information about the derivation that lead to a specific attachment decision. For example, we might have an S/S category, which we assume to be a modifier when it occurs as a lexical category (Today: S/S), but is not a modifier when it was derived via composition:

$$\mathsf{S/(S\backslash N)} \quad \mathsf{(S\backslash N)/S} \quad \rightarrow \quad \mathsf{S/S} \quad >\mathsf{B}^1$$

This means that when the non-modifier variant of S/S combines with S, its head will be the left child, whereas when the modifier S/S combines with a verb phrase or sentence, the head will be the right child (i.e. the verb phrase of sentence). Capturing head direction helps to model this distinction.

Each tree is generated top-down from the start category TOP. For each (parent) node, first its expansion type $exp \in \{\text{Lex}, \text{Unary}, \text{Left}, \text{Right}\}$ is generated. Based on the expansion type, the model then produces either the word w or the category of the head child (H), and possibly the category of the non-head sister category (S):

$$\begin{array}{ll}
\text{Lexical} & p_e(\text{exp}=\text{Lex} \mid \text{P}) \times p_w(w \mid \text{P}, \text{exp}=\text{Lex}) \\
\text{Unary} & p_e(\text{exp}=\text{Unary} \mid \text{P}) \times p_{\text{H}}(\text{H} \mid \text{P}, \text{exp}=\text{Unary}) \\
\text{Left} & p_e(\text{exp}=\text{Left} \mid \text{P}) \times p_{\text{H}}(\text{H} \mid \text{P}, \text{exp}=\text{Left}) \\
 & \quad \times p_S(\text{S} \mid \text{P}, \text{H}, \text{exp}=\text{Left}) \\
\text{Right} & p_e(\text{exp}=\text{Right} \mid \text{P}) \times p_{\text{H}}(\text{H} \mid \text{P}, \text{exp}=\text{Right}) \\
 & \quad \times p_S(\text{S} \mid \text{P}, \text{H}, \text{exp}=\text{Right})
\end{array}$$

The space of these distributions is constrained by the space of the grammar and the parses we see during training. Specifically, P, H and S are all CCG

categories that occur together in the training data. In practice this means we will only have certain pairs (H, S) which can combine to create P, not the full set of actions licensed by CCG. Moreover, since the combinatory rules of CCG are highly constrained, the sister S can be uniquely predicted from the parent P, head H and the combinator. In contrast to a CFG, where any nonterminals A, B, C can be combined via a rule A $\rightarrow$ B C, these constraints reduce the number of possible rules significantly. We will design a model to explicitly exploit them in the next chapter.

Finally, to evaluate this model, the predicted Viterbi parses are converted to word-word dependencies (Section 3.2.4) which can be compared against those extracted from the Penn Treebank. Specifically, we use Johansson and Nugues's [59] code[1] to obtain these dependencies, and the CoNLL 2008 shared task script [115] to evaluate unlabeled directed attachment. In order to extract comparable structures, we performed our CCG to Dependency conversion (Section 3.2.4).

This simple setup will provide us a basic evaluation of the learnability of an unsupervised CCG grammar. In the next chapter we will exploit the shared structure of categories. For now in this baseline model we only experiment with modifications to the grammar and the smoothing of our EM algorithm.

5.2 Grammatical Expressivity

The first important question we research is the effect of grammatical expressivity on performance. There are two main parameters to explore: the arity of lexical categories introduced by induction (with and without constituents), and the arity of composition allowed during parsing. Ultimately, we would like to examine how they affect performance on a number of languages, but in order to keep this cross product manageable, we treat English as our case study and only train on short sentences ($\leq$10 tokens) before trying to parse other languages. The questions we explore are:

- What arity of composition should be allowed?
- Should type-raising be allowed?
- How much lexical arity should be induced?

[1] `http://nlp.cs.lth.se/software/treebank-converter`

- Should constituent induction be included?

Parsing Power and Induction We will address the first two questions together. We will look at five parser settings. The parser can be restricted to only application (B^0), or allowed to use composition. Composition can be with arity 1 or 2 (B^1 & B^2) and optionally we can include Type-Raising in the parsing. The settings correspond to only allowing composition (B^0), allowing arity 1 composition with and without type-raising (B^1, B^1 +TR), and finally arity 2 composition with and without type-raising (B^2, B^2 +TR).

Lexical Arity and Constituent Induction Our second experiment will take the best combinator setting from our parsing power experiment and evaluate the importance of including constituents in the induction and if we should increase lexical arity. For these experimental settings we will use a number for the arity (1,2 or 3) and +d to denote the inclusion of derived constituents in the induction algorithm.

5.3 Training regimes: Full EM, Viterbi EM, K-best EM

The second question for training is the type of Expectation Maximization (EM) to use. For each of the aforementioned experimental setups, we will run three variations which correspond to different variants of EM. As noted, the parse forests of our training data specify the possible outcomes of every distribution. Once these have been computed, the model is initialized with uniform distributions. We choose this initializer because it is simple and reproducible, although it is not linguistically well motivated. This is particularly noticeable in the case of lexical emissions, $p_w(w \mid P,\ \exp = \text{Lex})$.

For example, $p_w(w \mid (\mathsf{N}\backslash\mathsf{N})/\mathsf{N},\ \exp = \text{Lex})$ can emit all part-of-speech tags aside from `CONJ` which means its initial distribution will weight the probability of the correct tag `IN` and any other equally at a probability of $\sim$0.03. Despite this, we will see the model performs surprisingly well.

Expectation Maximization When estimating a generative probability model, our goal is to maximize the probability of the observed data. When

labeled data is available, the frequency of events in the corpus can be summed and normalized to produce a Maximum Likelihood Estimate (MLE) solution for the values for every distribution in the model. Expectation Maximization provides a means for computing these distributions in the absence of labeled data. The approach has two components: the collection of expected counts (E-step), and updating the model (M-step). The expected counts are the sum of conditional probabilities for the data under the current model. When these are summed and normalized, we have defined a new model. This process is repeated until the model stops changing (convergence).

The process starts by defining an initial model. Instead of assuming random distributions for the initial model (as is often done), our initial models will use uniform distributions. This means that our results are deterministic, rather than subject to random variations, and easily reproducible. The initial model allows us to score parses in our data. Specifically, every chart item with category X for the cell spanning $w_i...w_j$ can be assigned some likelihood under the model as a function of the probability that a nonterminal with category X produces the yield $w_i...w_j$ (the inside probability) and the probability of a nonterminal X appearing in the context of $w_0...w_{i-1}$ X $w_{j+1}...w_n$ (the outside probability).

EM progresses by first computing the probability of a specific chart item existing in a sentence by multiplying the outside probability of X spanning $w_i...w_j$ by the rule probability $p(\text{Y Z} \mid \text{X})$ and the inside probabilities of Y (spanning $w_i...w_k$) and Z (spanning $w_{k+1}...w_j$). These probabilities can all be computed from the chart. When this product is divided by the marginal probability of the sentence (i.e. the total probability mass of all its parses) under the model, we have produced an expected count for the rule $\text{X} \rightarrow \text{Y Z}$ in this sentence, with X spanning $w_i...w_j$, Y spanning $w_i...w_k$ and Z spanning $w_{k+1}...w_j$. We obtain an expected count for the rule $\text{X} \rightarrow \text{Y Z}$ in this sentence by summing over all possible spans $w_i...w_j$ $(i < j)$ and all possible split points k $(i \leq k < j)$. Repeating this for every process for every rule in the grammar completes the E-step. EM, as applied to PCFGs is known as Inside-Outside [113] because of the two kinds of probabilities we just computed.

The M-step can now aggregate the counts for each rule's usage in every sentence of the corpus in the same manner we would extract frequency counts for a labeled learning problem. When these are normalized, we have a new, updated, value for $p(\text{Y Z} \mid \text{X})$. This model will predict the observed data

with higher probability than the last one (increasing the likelihood of the data under the model). When the change in likelihood between iterations shrinks below a given threshold, we say the model has converged.

There are two important points to remember about EM. First, it provides a training procedure for learning a model without labeled data. Second, every possible instantiation of the hidden structure (i.e. every possible parse in the entire parse forest) contributes to the model updates. The remainder of our discussion on EM will focus on changing this second assumption.

As we have already discussed, the induction procedure is highly ambiguous. As such, most of the parses in our charts will be wrong, and, therefore, they will contribute potentially misleading expected counts for the model.

Viterbi EM One way to avoid accumulating counts from tail phenomena in a parse forest, which are likely to be incorrect, is to perform Inside-Outside using only the single "best" parse per sentence. Instead of accumulating a tremendous number of small expected counts across the corpus, we assume the corpus of N sentences has N analyses, each with probability 1. The hope with this approach is that only very common substructures will appear in the top parses, causing the model to be strongly biased towards descriptive and useful rules in the grammar. Unfortunately, the choice of the "best" parse is determined by the initial, potentially random, model. This style of "winner-take-all" EM [116] is often referred to as hard EM, as it makes a hard assignment of the probability mass of a chart to one parse, or Viterbi EM, in reference to the algorithm for extracting the best parse from a parse forest.

Spitkovsky et. al [50] demonstrated the utility of hard EM for unsupervised grammar induction. We compare standard full (soft) EM, where we use the entire parse forest during estimation, with Viterbi EM, as well as with a smoothed variant of K-best EM which interpolates the probabilities from the top-K parses with those of the full forest.

Interpolated K-Best EM We know from prior work [50] that there is immense ambiguity in the predictions made by an unsupervised grammar induction system. One way to help the model train is to encourage it to reinforce whatever biases or statistics it has found in the data. This can be done by Viterbi-EM. But unfortunately, the model's predictions may not be

very informative, particularly early in training when the model may have been initialized into a poor local optima. This is likely particularly problematic for a model of constituency grammars like ours which has an extremely ambiguous grammar with many parameters.

To handle this sparsity we devised a middle-ground training scenario: Interpolated K-best EM. In this framework, the EM pseudocounts from the top K parses are used to compute statistics c_k, and the pseudocounts of the full forest are used to compute the counts (c_{full}). We then compute two new sets of model parameters: $\hat{p}_k(x \mid y)$ with c_k and $\hat{p}_{\text{full}}$ with c_{full}. This simply requires normalizing the counts for these two sets of expectations separately, the later being the counts used by the normal EM algorithm.

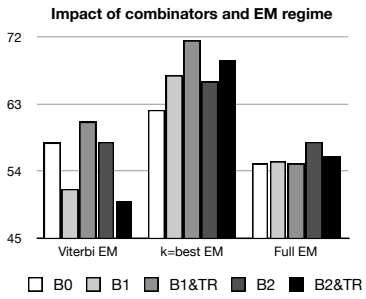
To weight how much the model should trust one set of parameters over the other we interpolate with $\lambda_k = c_k/(c_k+c_{\text{full}})$, and compute a new interpolated probability:

$$\tilde{p}(x \mid y) = \lambda_k \hat{p}_k(x \mid y) + (1 - \lambda_k)\hat{p}_{\text{full}}(x \mid y).$$

If the model is confident about its prediction λ_k will be large, otherwise the model will rely more heavily on the distribution computed from the full set of pseudocounts. We use "Algorithm 3" of Huang and Chiang [117] to compute the K-best parses according to the current model.

While our induction algorithm does strategically limit the number of categories introduced, we find that further biasing of the data set with top-K parsing greatly improves performance. Depending on the size of the initial lexicon and the computational power afforded to the parser, Viterbi parsing often hurts performance while K-best always proved beneficial, occasionally leading to 20-point gains in performance (Figure 5.1). When sampling values for K varying from 5-150 with the an arity two lexicon parsed with B^1 +TR, we found performance varied with a standard-deviation of 0.6, meaning the need for a K is important, but the smoothing is largely robust to the specific value.[2] All initial experiments are on English and all distributions are initialized uniformly to avoid randomness.

[2]We settle on a value of 100 for B^1 +TR.



	Viterbi EM	K-best EM	Full EM
B^0	57.8	**62.2**	55.0
B^1	51.6	**66.8**	55.3
B^1 +TR	60.6	**71.5**	55.0
B^2	57.9	**66.0**	57.9
B^2 +TR	50.0	**68.8**	56.0

Figure 5.1: Impact of the expressiveness of the grammar and training regimen on Section 0 performance directed attachment performance.

5.4 Analysis of PCFG Performance

Before providing a comparison with related work, we set out to evaluate some of the open questions we have just detailed. How much parsing power should be used? What is the effect of the K-best EM? What lexical arity and induction scheme are best?

We begin by addressing the first two questions in tandem and holding the lexical ambiguity constant. We found in section 4.2.1 that arity 2 categories comprised 95% of the English treebank. For this reason, we will run our basic induction algorithm for two iterations in the initial experiment.

In our case study, we will be training the system on sections 02-21 of the WSJ and testing on section 0. Additionally, all models are the result of training only on short sentence (up to 10 words excluding punctuation) and similarly, all evaluations are computed on short sentences of up to 10 words.

5.4.1 Impact of Combinators and Values of K

The combinatory rules allowed during parsing determine the expressiveness of the grammar, and hence the complexity of linguistic phenomena that can be captured. They also has a significant effect on the size of the grammar and number of parses per sentence. The training regime impacts the effective size of the grammar as well: Viterbi training (i.e. only using the highest scoring parse to update the model) effectively prunes the grammar, while our smoothed K-best algorithm spreads the mass out among frequent constructions and categories. We therefore found a strong interaction between these two parameters: grammar size and the choice of training regimen.

Figure 5.1 provides results on the test set for each grammar setting with Viterbi, full EM, and $K = \text{best}_G$ (a grammar-specific setting of K that was found to optimize performance on the development set). Unlike Spitkovsky et al. [50], we found that Viterbi parsing does not in general outperform full EM, but the right choice of K for K-best parsing can yield substantial improvements in performance. We also found that type-raising proved beneficial in both the B^1 and B^2 cases. These results are based on the use of two basic induction steps in addition to a final induction step with derived constituents (Sec 4.2.4).

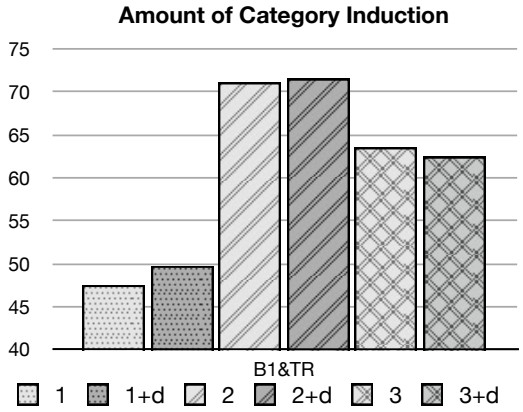
	Error Reduction from:	
	Viterbi EM	Full EM
B^0	*10.4 %*	16.0 %
B^1	**39.6 %**	25.7 %
B^1 +TR	27.7 %	36.7 %
B^2	19.2 %	19.2 %
B^2 +TR	**37.6 %**	29.1 %

We do not have a good explanation for the reason up-weighting K proves so effective beyond noticing that the decreases in error achieved using this technique were greatest for the most ambiguous grammars (those incorporating Type-Raising) and least for the simplest grammar (B^0). This may simply indicate that the model or data have the correct biases but it is split between multiple parses. These biases are aggregated by K-best EM where Viterbi EM forces the model to choose a single, partially correct, analysis.

5.4.2 Number of induction stages

Having found K to be highly influential on the model's performance and B^1 +TR to perform best, we now fix these parameters to explore our final question: the impact of lexical arity on performance (again using a grammar-specific optimal K). Figure 5.2 shows that, for grammars that use B^1 +TR, two iterations of induction perform the best, while induction from derived constituents has a minimal (or slightly detrimental) effect.

We see that arity 2 lexicons greatly outperform the rest and that constituent induction has a very minor effect on performance. We can use the



	1	1 + d	2	2 + d	3	3 + d
B1+TR	47.47	49.71	71.08	71.52	63.51	62.43

Figure 5.2: We use the best performing grammar setting from the previous experiment (B1+TR) to test the impact of inducing differing grammars. Specifically, we look at the impact of the number of induction stages on performance ("d": derived constituents are considered Section 4.2.4).

lessons learned from these experiments to perform multilingual evaluations again the literature.

5.5 Test-Set Performance

We conclude from our experiments in section 5.2 that we should train and evaluate a model that uses an arity 2 lexicon and parse with B^1 +TR. We perform an initial evaluation on section 23 of the WSJ, followed by an analysis of the weighted lexicons, and finally a multilingual evaluation.

5.5.1 Performance Comparison

We present results on section 23 (Table 5.3), when trained on length 10 data from sections 02-21. Starred results were obtained with additional training data: up to length 20 (Naseem '10 [37]) or 45 (Spitkovsky '10 [50]). Almost none of these systems' performances are directly comparable to each other (see Section 2.2.4). The closest comparison to ours (same data splits but potentially different head-finding rules) are Cohn '10 [36] and Headden '09 [34].

What our results seem to indicate is that our system performs the best

	10	20	Inf
Klein & Manning '04	47.5		
Headden '09	68.8		
Spitkovsky Vit '10	65.3*	53.8*	47.9*
Cohn '10	65.9	58.3	53.1
CCG Induction	71.5	**60.3**	**53.3**
Naseem '10 Universal	**71.9**	50.4*	
Naseem '10 English	73.8	66.1*	
Boonkwan '11	74.8		

Figure 5.3: Full table of comparison results for section 23

among the approaches with lack English specific knowledge when evaluated on long sentences (length 20 or the full corpus), and we nearly match Naseem et al. 's performance on short sentences. Despite this strong result, there is still large performance gap between our system and either Naseem's when using English knowledge or Boonkwan's semi-supervised lexicon.

One optimization we did not investigate was a comparison of early versus late stopping during training, or how performance varied through iterations of EM. We simply choose a convergence threshold for the amount of change in log-likelihood (.0001) and reported results at convergence.

5.5.2 The Induced Lexicons

We find that the lexical categories our system induces match very well the commonly assumed CCG categories for English. Figure 5.4 lists common POS tags and their most likely categories (probabilities of category given tag were computed based on the Viterbi parses of our best performing model on the development set). Besides overall accuracy (which depends not just on the lexicon, but also on the kinds of attachment decisions the model makes), this is a very good indicator of how much of the language's basic grammar the model has captured, because CCG encodes all language specific information in its lexicon. We find that most of the mass is centered on exactly those categories that a linguist would include in a basic (C)CG lexicon for English, and that generally little mass is assigned to non-standard categories such as (N/N)/N for NN (common noun) or IN (prepositions and subordinating conjunctions). The only possible exceptions are (S\S)/S for infinitival TO (*to*) and S/N for VB (infinitival verbs), which can both be explained by the

Tag	Category	$p(c \mid t)$	Tag	Category	$p(c \mid t)$
NN	N	0.839	RB	S/S	0.527
	N/N	0.133		(S\S)/(S\S)	0.275
	(N/N)/N	0.021		S\S	0.119
DT	N/N	0.925	VBD	(S\N)/N	0.419
	N	0.034		S\N	0.339
	(N/N)/N	0.011		(S\N)\S	0.339
JJ	N/N	0.861	TO	(S\S)/S	0.498
	S\S	0.114		(S\S)/N	0.437
	(S/S)/N	0.012		N/N	0.012
IN	(S\S)/N	0.678	VB	S/N	0.743
	(N\N)/N	0.148		S	0.151
	(N/N)/N	0.069		N/N	0.031

Figure 5.4: The most likely induced lexical categories for common parts of speech (probabilities based on Viterbi parses of section 00)

fact that infinitives are rarely preceded by subjects (so we are unlikely to have the necessary *noun verb* context required to learn S\N), whereas (S\N)\S for VBD (past tense verbs) is actually required for inversions that are frequently used with direct speech in our domain (*"This was obvious", he said.*).

Why is our model able to induce these linguistically correct categories? Since our induction scheme allows all categories to be modifiers or modifiers of modifiers, one obvious grammar that it permits is one where verbs are S (to fulfill the constraint that sentences containing verbs are analyzed using the TOP $\rightarrow$ S rule), and everything else is either S/S or S\S. The reason that this does not plague us is subtle yet important. Because we do not differentiate between lexical and non-lexical non-terminals but rather have a distribution over expansions (Section 5.1), the frequent use of S throughout the tree in various binary productions leaves little mass for a lexical S. In contrast, a category like (S\N)/N will nearly never appear anywhere but at the lexical level, resulting in a very high probability of being a lexical category.

Specifically, recall that the model has four outcomes in the expansions distribution:

$$p_e(\text{exp}=\text{Lex} \mid \text{S}) \qquad p_e(\text{exp}=\text{Unary} \mid \text{S})$$
$$p_e(\text{exp}=\text{Left} \mid \text{S}) \qquad p_e(\text{exp}=\text{Right} \mid \text{S})$$

These must sum to one, but pressures like $\mathsf{TOP} \rightarrow \mathsf{S}$ prevent the lexical distribution from ever consuming too much of the mass. Even in the simplest, two word, sentences that contain a lexical S, there must be another that is not lexical (e.g. $\mathsf{S/S}\ \ \mathsf{S}$ has one lexical S and one with exp=Right). In contrast $p_e(\text{exp}=\text{Lex} \mid (\mathsf{S\backslash N})/\mathsf{N})$ can get arbitrarily close to one as it will almost always appear as only a lexical category. This is particularly true for parsing settings where the composition arity is limited. For example, in B^1, $\mathsf{S|S}$ will not be able to compose into the transitive verb, so with the exception of coordination the category will always be lexical.

An additional question is why do nouns acquire the English ordering $\mathsf{N/N}$ when they have the equally valid opportunity to be $\mathsf{N\backslash N}$. Specifically, with compound nouns (e.g. NN NN) there are two equally likely analyses: $\mathsf{N/N}\ \ \mathsf{N}$ and $\mathsf{N}\ \ \mathsf{N\backslash N}$. Although we allow the tag $\mathtt{DT}$ (e.g. *this*) to act as a noun (e.g. *This/DT is/VBZ fun/NN*), many noun phrases do not contain a determiner, increasing the relative frequency with which N generates a nominal tag, and decreasing the probability of it generating $\mathtt{DT}$. Further, $\mathtt{DT}$ can almost always be analyzed as $\mathsf{N/N}$ and when the determiner is missing, the adjectives, which tend to precede nouns, must take the category $\mathsf{N/N}$. The result of these factors combined is a very high probability of $p_e(\text{exp}=\text{Right} \mid \mathsf{N})$ and therefore a grammatical bias towards $\mathsf{N/N}$ for nouns, determiners, and adjectives.

5.5.3 Multilingual Performance

Finally, we performed a basic evaluation of this naive approach across ten corpora as part of the PASCAL Shared Task [63, 118]. Participants were allowed to tune their systems on the development set, supplement with additional data, and were encouraged to train on the full union of the train, development, and test sets. We present results for our PCFG system against that of Blunsom and Cohn [42] who also trained a constituency style parser (Tree-Substitution Grammar) and a max over all participating systems in Table 5.1. We discuss this evaluation and the systems being compared again more fully in Section 7.1.1.

We also attempted to tune our system. Our knobs included whether to use a language's coarse or fine tagset, the length of sentences included in the training data and if punctuation should be included during training via the introduction of simple binary rules (e.g. $\mathsf{X} \rightarrow \mathsf{X}\ \ \mathsf{punc}$). Again, the value

of K was tuned, but its exact value had a marginal effect on performance. Participants tuned many of the same values and occasionally entered multiple systems, one with each type of part-of-speech tagset (Fine, Coarse, UPOS). Punctuation was included in Arabic, Childes, Danish, Dutch and Slovene. Additionally, we report the length of sentences included during training. We experimented with sentences of length 10, 15, 20, and in the case of Arabic we included 40 because the data set was so small.

The optimal settings for every language (chosen on the development data) are presented in the top two rows of Table 5.1. The model performances are for sentences of length 10 and 15 (not counting punctuation marks).

What is immediately clear from our results is that even with tuning (which has limited effect), our results fall short of the best performing systems. Most of the best results came from the work of Tu [43] that used the development data to choose the best regularization on dependency types. Despite this, our approach is competitive or the best in many languages. It appears our best results are on languages with more data (right side of the table) as compared to those with very few tokens (left). In the next chapter, we introduce a new model which will greatly outperform our PCFG model without any of this tuning.

As a final note we should mention that the workshop organizers performed a more in-depth analysis of the English results and found that our system performed best overall when different head-finding rules were used for determining dependencies. They compared five types of dependencies: *"standard, CoNLL2007, functional references, lexical, and oldLTH"* [63]. We performed best on the lexical and oldLTH evaluations where Blunsom and Cohn performed best on the standard, CoNLL2007 and functional evaluation.

5.6 Conclusions

What we have demonstrated here is that a simple constituent based model (PCFG), which does not take into consideration the internal shared functional structure of CCG, can still produce dependencies at a level competitive with systems which directly model dependencies. In the next chapter, we will introduce a novel model factorization specific for CCG. In this chapter we introduced a simple technique to improve performance via interpolated K-

	Arabic	**Danish**	**Slovene**	**Swedish**	**Dutch**	**Basque**	**Portuguese**	**WSJ**	**Childes**	**Czech**	**Average**
Sent Len	40	20	10	15	10	20	10	10	20	10	
Tags	Fine	24	Fine	Fine	Fine	Fine	Coarse	Fine	Fine	Fine	
BC	60.8/58.4	44.7/39.4	62.6/57.9	63.2/56.6	**51.8/52.0**	53.0/**48.9**	52.4/50.2	68.6/63.3	47.4/46.1	47.9/43.1	55.2/51.6
Max	**67.2/66.8**	**60.1/56.0**	**65.6/61.8**	**72.8/63.4**	51.1/47.6	**53.7**/47.8	67.0/61.8	**71.2/64.8**	56.0/54.5	**58.3/54.4**	**62.3/57.9**
PCFG	41.6/43.7	46.4/43.8	49.6/43.9	63.7/57.0	49.7/43.6	45.1/39.6	**70.8/67.2**	68.2/59.6	**61.4/59.8**	45.0/38.9	54.2/49.7

Table 5.1: A comparison of participants in the PASCAL Challenge [63]. We compare the performance of our PCFG [118] against Blunsom and Cohn (BC) [42], and a max over all other participants. The numbers reported are for performance on sentences of length 10/15 (not counting punctuation) on the test set.

best training, but in the next chapter we will use a non-parametric bayesian formulation to achieve smoothing whose performance will be more impactful and easily controlled via setting hyperparameters.

Chapter 6

A Hierarchical Non-Parametric Bayesian Model for CCG

Having demonstrated that a grammar can be learned from our very general induction scheme and that the CCG parses once converted to dependency trees are competitive with existing approaches, we now turn our attention to the primary model of this thesis: the Argument Model and its extensions. This model is tailored to the specific constrained nature of CCG derivations.

6.1 A New CCG Argument Factorization

We have just seen that a basic PCFG approach to modeling was competitive with dependency induction algorithms. We experimented with simple ways to augment the training procedure to constrain the grammar and subsequently help address the issue of sparsity. Unfortunately, this factorization does not take advantage of the unique functional nature of CCG. We, therefore, introduced a new factorization we call the Argument Model [119]. This factorization will allow us to define a novel non-parametric model of CCG parses. This underlying insight in this model is the constrained nature of CCG categories. We will train both a parametric and non-parametric variant of the model.

The model factorization exploits CCG's strong strong constraints on a parent category's left and right children since these must combine to create the parent type via one of the combinators. In practice, this means that given the parent $\mathsf{X/Z}$, $\mathsf{X\backslash Z}$ or an atomic X, the choice of combinator c and an argument Y, we can uniquely determine the categories of the left and right children:

Work in this chapter was first published in Y. Bisk and J. Hockenmaier, "An HDP Model for Inducing Combinatory Categorial Grammars," *Transactions of the Association for Computational Linguistics*, pp. 75-88, 2013.[119] and is reprinted here with permission by the copyright holder.

Parent	c	$\rightarrow$	Left	Right
X/Z	$B^0_>$		(X/Z)/Y	Y
	$B^0_<$		Y	(X/Z)\Y
	$B^1_>$		X/Y	Y/Z
	$B^1_<$		Y/Z	X\Y

and correspondingly for X\Z:

Parent	c	$\rightarrow$	Left	Right
X\Z	$B^0_>$		(X\Z)/Y	Y
	$B^0_<$		Y	(X\Z)\Y
	$B^1_>$		X/Y	Y\Z
	$B^1_<$		Y\Z	X\Y

Finally, when the parent is atomic only application is possible:

Parent	c	$\rightarrow$	Left	Right
X	$B^0_>$		X/Y	Y
	$B^0_<$		Y	X\Y

These tables should look very familiar. They are precisely the derivation rules from the CCG grammar definitions, but flipped (section 3.1.2). Now instead of having two categories we wish to combine, as is done during parsing, we have a parent from which we produce children, eventually as part of a generative story.

This formulation easily extends to handle unary rules that arise via type-raising (T, Section 6.4.1) or type-changing (XX, not used in this thesis). We simply treat the argument Y as the unary outcome so that the parent, combinator and argument uniquely specify every detail of the unary rule:

Parent	c	$\rightarrow$	Y
TOP	TOP		$\in \{S, N\}$
S/(S\N)	$T_<$		N
S\(S/N)	$T_>$		N
N\N	XX		S\N

In CCGbank, very few type-raised categories are used, and they are constrained to overlap with the set of lexical categories. Specifically, if type-raising produces a category T/(T\X), T\X must exist in the lexicon. If our

model were trained on gold-standard parses, we would entertain the same set of constraints, but for simplicity in our unsupervised models we will only entertain the two type-raising rules shown in the table above. Future work should explore expanding the search space further.

We still distinguish the same rule types as before (lexical, unary, binary with head left/right), leading us to the following model definition:

$$\begin{aligned} &\text{Given:} \quad \mathsf{P} := \mathsf{X} \\ &\text{where} \quad t \in \{\text{Left,Right,Unary,Lex}\} \end{aligned}$$

$$p(t \mid \mathsf{P}) \times \begin{cases} p(w \mid \mathsf{P}, t) & \text{Lex} \\ \underbrace{p(\mathsf{Y} \mid \mathsf{P}, t)}_{\text{Argument}} \times \underbrace{p(c \mid \mathsf{P}, t, \mathsf{Y})}_{\text{Combinator}} & \text{o.w.} \end{cases}$$

Note that this model generates only one CCG category, but uniquely defines the two children of a parent node. We will see below that this greatly simplifies the development of non-parametric extensions. Specifically, where a CFG must define a distribution over all possible children B and C for any rule A $\rightarrow$ B C, our approach generates a single argument Y given the parent P. When B and C are each extended to range over an infinite set of nonterminal categories, non-parametric PCFG models have to capture a product over two infinite distributions, p(B) and p(C). By constrast, if we allow Y to range over an inifinite set of categories, we only have to model one infinite distribution, $p(\mathsf{Y})$. We will see that this greatly simplifies our approach (Section 6.3.3).

There are a few important notes to consider in our model definition:

- P will take the form $\mathsf{X}|\mathsf{Z}$, $\mathsf{X}|\mathsf{X}$, X and TOP.

- One place the model can leak mass, depending on how distributions are defined and smoothed, is lexical emissions. If the model assigns non-zero probability to emitting any word from any category (due to the base measures), but the grammar only allows parses where a (word/tag, category) pair exists in the induced lexicon, the model will place mass on impossible outcomes.

- In the parametric model (Section 6.2), the space of argument categories Y will be fixed and determined by the set of parses seen when parsing the training data. For this reason, only the non-parametric model (Section 6.3) must properly deal with the infinite possible space

of outcomes allowed by CCG. By constraining the grammar to rules seen during training, not just lexical items, we will not introduce novel instantiations during testing, but we will define the model to score the infinite space of unseen sentences and parses.

6.2 Parametric (Non-Hierarchical) Argument Model for CCG

Having defined a new way to factor CCG categories, we can estimate the model's distributions using EM, the same way we did for the PCFG model (Chapter 5.3). We can use the induced lexicons to parse the corpus, use an initial model to compute pseudocounts, and then update the model's distributions $p(t \mid \mathsf{P})$, $p(w \mid \mathsf{P}, t)$, $p(\mathsf{Y} \mid \mathsf{P}, t)$, and $p(c \mid \mathsf{P}, t, \mathsf{Y})$.

The strength of this model is that it takes CCG's functional nature into account. Specifically, even if the model has weak performance, the model's error analysis may prove more informative than that of CFG based approaches. For example, given some parent X, the CFG approach provides a distribution over possible children B, C which combine to create X, but the distribution does not directly inform us about any commonality between the children. The children are treated as independent.

In contrast, in the argument factorization, if the model learns that the argument N is more likely to be generated from a parent S, we know the model has learned that verbs have a bias towards taking nouns as arguments. In contrast, if the primary argument for S is also S, we know the model is favoring modifier analyses, a potentially problematic result.

One of the weaknesses of this simple approach is that there is no parameter sharing between distributions. It would make sense to extend our current reasoning about arguments further. We might want to ask what the language's argument taking behaviors look like as a whole, and how specific categories diverge from the norm.

Additionally, in a parametric model, all distributions must be defined in advance. The values will change during training, but the set of conditioning variables and possible outcomes must all be defined before the model can be used or trained. These are the *parameters* in the *parametric* model. In contrast, a non-parametric model has the ability to introduce new distribu-

tions, outcomes and conditioning variables automatically. The model will introduce distributions as necessary for the data. It does this, in part, by exploiting shared parameters between distributions.

The ability to handle shared distributions and capture how the model diverges from population norms, as well as the ability to introduce new distributions as necessary are two important strengths of a hierarchical non-parametric approach.

6.3 HDP-CCG: A Non-Parametric Model

Simple generative models such as PCFGs (previous chapter) or the parametric version of the argument model (previous section) are not robust in the face of sparsity, since they assign zero probability to any unseen event. Sparsity is a particular problem for formalisms like CCG that have a rich inventory of object types. Non-parametric Bayesian models, e.g. Dirichlet Processes [120] or their hierarchical variants [121] and generalizations [122] overcome this problem in a very elegant manner, and are used by many state-of-the-art grammar induction systems [37, 42, 38]. They also impose a rich-getting-richer behavior that seems to be advantageous in many modeling applications. Earlier, in Section 5.3, we attempted to mimic this rich-get-richer behavior for our PCFG model in an ad-hoc manner via a top-k reweighting version of EM.

The argument model introduced above lends itself particularly well to non-parametric extensions such as Hierarchical Dirichlet Processes (HDP) [121]. In this thesis, the size of the grammar and the number of productions are fixed because they are constrained by the induced grammar, but we present the formulation as infinite to allow for easy extension in the future. Specifically, this framework allows for extensions which grow the grammar during parsing/training, or fully lexicalize the productions. The strength of the non-parametric approach is its ability to introduce new, previously unseen, outcomes to distributions. Lexical distributions are one place where this is particularly important. The space of words that a category might emit is both large and, in general, unbounded, making it necessary to have a mechanism which can handle sparsity and assign probability mass to novel data.

Additionally, again, while our current work uses a restricted fragment of

CCG that has only a finite set of categories, i.e. those induced or produced during parsing of the training data, CCG allows for generalized composition [53], which makes it possible to generate categories of unbounded arity. We therefore believe that this is a very natural probabilistic framework for CCG since HDPs make it possible to consider a potentially infinite set of categories that can instantiate the Y slot while allowing the model to capture language-specific preferences for the set of categories that can appear in this position.

It is important to note that the categories which serve as arguments may themselves have internal structure. For example, in the rule

$$\mathsf{X/(Y/Z)} \;\; \mathsf{(Y/Z)/W} \rightarrow \mathsf{X/W}$$

the argument takes the form $\mathsf{Y/Z}$. This means that any generative story which includes the production of CCG arguments as defined here must be able to generate and assign probabilities to a possibly unbounded number of categories with internal structure. We will return to this in more detail later.

6.3.1 Incorporating the HDP

In Bayesian models, multinomial distributions are drawn from a corresponding n-dimensional Dirichlet distribution prior. Multinomials are n-dimensional distributions over a discrete set of outcomes. For example, a fair 6-sided die has the distribution $M = [\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}]$. Each dimension of the vector corresponds to the probability of rolling the numbers one through six. In general, dice might be rigged to express an infinite number of distributions. Formally, the value of each dimension can take any value in the range $[0, 1]$ as long as the sum of all dimensions is equal to one: $\sum_i M[i] = 1$.

Due to this constraint, if we plot the location of every multinomial of dimensionality n, they fall within an (n-1)-dimensional simplex. An n-dimensional Dirichlet distribution is defined over the (n-1)-dimensional simplex:

$$p(x_1, ..., x_K; \alpha_1, ..., \alpha_K) = \frac{\prod_{i=1}^{K} \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{K} \alpha_i)} \prod_{i=1}^{K} x_i^{\alpha_i - 1}$$

The Dirichlet distribution assigns probabilities to any point on the simplex and therefore gives the probability of choosing any particular multinomial distribution, as each point on the simplex $p = [p_1...p_{n-1}]$ (with $0 \leq p_i \leq 1$ $\forall i$, and $\sum_i p_i = 1$) defines an n-dimensional multinomial distribution. The

Dirichlet distribution is a conjugate prior distribution to the multinomial, as it defines a distribution $p(x)$ over models x prior to the incorporation of any observed data y. According to Bayes' rule, $p(x \mid y)$, the posterior probabilty of the model given the data is proportional to the prior $p(x)$ times the likelihood of the data under the model, $p(y \mid x)$.

$$p(x \mid y) = \frac{p(y|x)p(x)}{p(y)}$$

A conjugate prior has the same mathematical form as the posterior $p(x \mid y)$. That is, if the prior $p(x)$ is a Dirichlet distribution, and $p(y \mid x)$ is a multinomial to be estimated from data, the posterior $p(x \mid y)$ is again a Dirichlet distribution, which can be multiplied by the prior, $p(y)$, to produce a posterior, $p(x \mid y)$, which can be used as the new (updated) prior. This description is, again, parametric in nature because we assume knowledge of n, the space of outcomes, in advance.

(Hierarchical) Dirichlet Processes

The Dirichlet Process (DP) generalizes the Dirichlet distribution to an infinite number of possible outcomes, allowing us to deal with a potentially infinite set of categories or words. DPs are defined in terms of a base distribution H over the space Θ that corresponds to the mean of the DP, and a concentration or shape parameter α. G is Dirichlet process distributed with base distribution H and concentration parameter α, written $G \sim \mathrm{DP}(\alpha, H)$, if $(G(A_1), ..., G(A_r)) \sim \mathrm{Dir}(\alpha H(A_1), ..., \alpha H(A_r))$ for every finite measurable partition $A_1, ..., A_r$ of Θ [123, 124, 125, 120]. There are several popular ways to define the DP. We present the stick-breaking construction of Suethuraman (1994) [126].

Our aim is to create a discrete distribution over an infinite space (of words or categories). Naively, extending a multinomial to infinity does not make sense as $\frac{1}{\infty} = 0$. Further, we will only emit an infinite set of words or categories in the limit, so there is a tremendous amount of wasted probability mass in such a naive characterization. The incremental generation of data, coupled with the fact that common observations are likely to be observed first, is captured within the DP formulation and the stick-breaking construction. The basic metaphor behind the stick-breaking construction is that of a (potentially unbounded) number of pieces that are being successively broken

off of a unit-length stick. The length of each piece is determined probabilistically as a fraction of the length of the currently unbroken part of the stick.

Formally, H defines a base distribution over the infinite space of outcomes. We sample an atom δ_{ϕ_k} (outcome) with probability ϕ_k from H. Next, we assign this outcome some probability. Remember that the total probability mass that can be assigned to the union of all outcomes is one, so we need a mechanism for choosing how much of that mass (i.e. β_k) to assign to our new outcome ϕ_k. This is achieved with a Beta distribution (the conjugate prior for the Bernoulli distribution), $Beta(1, \alpha)$, which defines a distribution over the range [0,1].

$$p(x; 1, \alpha) = \frac{\Gamma(1+\alpha}{\Gamma(\alpha)}(1-x)^{\alpha-1}$$

If this is the first outcome generated ($k = 1$), we say the probability of choosing the outcome ϕ_1 is $\beta'_1 \sim Beta(1, \alpha)$. For the next unique outcome ($k = 2$), the mass that remains to be allocated is $1 - \beta'_1$. We therefore assign ϕ_2 a portion $\beta'_2 \sim Beta(1, \alpha)$ of the remaining mass, or $\beta'_2 \times (1 - \beta'_1)$. In general, for draw k we define β_k, i.e. the overall probability mass of the k-th outcome, as the product of the size of the remaining stick ($\prod_{l=1}^{k-1}(1-\beta'_l)$) and the fraction of that remaining stick that is broken off ($\beta'_k \sim Beta(1, \alpha)$):

$$\beta_k = \beta'_k \prod_{l=1}^{k-1}(1 - \beta'_l)$$

What we have just defined is a mechanism for generating and weighting an infinite set of discrete outcomes. We can put this all together to define a Dirichlet Process measure G by assigning probabilities β_k to each point δ_{ϕ_k} from 1 to infinity:

$$G = \sum_{k=1}^{\infty} \beta_k \delta_{\phi_k}$$

We can now draw multinomials from this process in the same way we previously drew multinomials from a Dirichlet distribution.

The most powerful part of this approach is the ability to define a hierarchy of Dirichlet Processes. In a Hierarchical Dirichlet Process (HDP) [121], there is a hierarchy of DPs, such that the base distribution of a DP at level n is another DP at level $n - 1$. This means that just as we drew new outcomes

from H for the DP, now we draw outcomes from G for the HDPs at the next level of the hierarchy. We will make this more concrete throughout the discussion of our model, but we can quickly convey the intuition here.

Imagine a distribution over the potentially infinite space of English words: $p(w)$. If we want to model bi-gram probabilities, $p(w_i \mid w_{i-1})$, we need to define an infinite set of distributions (one for every w_{i-1}), each of which have an infinite space of outcomes (every possible w_i). The HDP allows us to define a prior G whose mean is the unigram distribution $p(w)$. This allows us to draw similar $p(w_i \mid w_{i-1})$ distributions from a shared prior. In other words, before any evidence is acquired to help estimate $p(w_i \mid w_{i-1})$ we can specify how much variation exists between every distribution by specifying the shape of the shared base measure G. Further, once information is acquired, it can be used to inform the shared measure G to influence all other sampled distributions.

A perhaps simpler way to understand this is that the HDP allows us to specify that all bi-gram probability models for words should, initially, take a form very similar to the unigram distributions, until evidence is accumulated that indicates otherwise. Second, any information learned about the shape of the bi-gram should be propagated back to inform the unigram distribution. In these two complementary ways, the HDP allows for an elegant solution to smoothing and parameter sharing in the face of infinite distributions. This technique has been demonstrated as being very effective for language modeling [122].

HDP Formulation of the Argument Model

This intuition for the shared parameters between distributions in a bi-gram language model applies to our approach to modeling CCG parses. Specifically, we aim to capture the same sharing of distributions over words between the lexical categories emitting them. Additionally, the factorization allows us to capture parameter sharing between the argument generating distributions, $p(\mathsf{Y} \mid \mathsf{Z}, t)$. These produce a CCG argument Y given a parent Z and expansion type ($t \in \{\text{Left, Right, Unary}\}$). We discuss this more fully in section 6.1.

The HDP-CCG (Figures 6.2 and 6.1) is a reformulation of the Argument

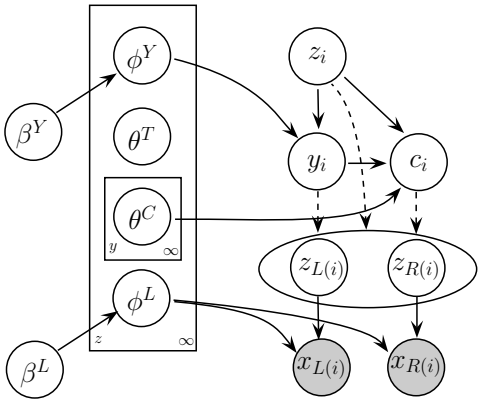


Figure 6.1: This is the plate diagram for our model. It allows for an infinite space of categories and lexical emissions. Because we are working with CCG, the parent z_i, argument y_i and combinator c_i uniquely define the two children categories $(z_{L(i)}, z_{R(i)})$. The dashed arrows here represent the deterministic process used to generate these two categories.

Model introduced above in terms of Hierarchical Dirichlet Processes.[1] The model has two main families of distribution: 1. Infinite lexical emissions 2. Infinite generation of CCG categories.

In both cases, a shared base distribution defines the global distribution over either lexical emissions (β^{L}) or CCG categories which serve as arguments (β^{Y}). In the face of sparsity, there may be insufficient evidence to estimate a good lexical or argument distribution for a category. In these cases, the shared base distributions allow the model to "fall back" on the global mean. The ease with which our factorization allows for implementing this parameter sharing and non-parametric extensions is at the heart of what makes the model so attractive.

An argument y_i is drawn for a specific CCG category from $\phi^{\mathrm{Y}}_{z_i}$. By combining a stick breaking process with a multinomial over categories we can define a DP over CCG categories whose stick weights (β^{Y}) correspond to the frequency of the category in the corpus. Next we build the hierarchical component of our model by choosing an argument distribution (ϕ^{Y}), again over the space of categories, for every parent $\mathsf{X/Z}$. This argument distribution is drawn from the previously defined base DP, allowing for an important level

[1] An alternative HDP model for CCG semantic parsing was proposed by Kwiatkowski et al. (2012) [127], but it does not take advantage of our CCG specific argument factorization and instead models any child of a parent in the grammar (unary, binary, lexical or lambda calculus) as a tuple to be emitted in a manner more akin to a CFG.

HDP-CCG

1) Draw global parameters
- Define MLE root parameter: θ_{TOP}
- Draw top-level symbol weights: $\beta^{\text{Y}} \sim \text{GEM}(\alpha^{\text{Y}})$
- Draw top-level lexical weights: $\beta^{\text{L}} \sim \text{GEM}(\alpha^{\text{L}})$
- For each grammar symbol (full CCG category) $z \in \{1, 2, ...\}$:
  - Define MLE rule type parameters: θ^{T}_z
  - Draw argument parameters: $\phi^{\text{Y}}_z \sim \text{DP}(\alpha^{\text{Y}}, \beta^{\text{Y}})$
  - Draw lexical emission parameters: $\phi^{\text{L}}_z \sim \text{DP}(\alpha^{\text{L}}, \beta^{\text{L}})$
  - For each grammar symbol $y \in \{1, 2, ...\}$:
    - Define MLE combinator parameters: $\theta^{\text{C}}_{z,y}$

2) For each parse tree:
- Generate root node: $z_{\text{TOP}} \sim \text{Binomial}(\theta_{\text{TOP}})$
- For each node i in the parse tree:
  - Choose rule type: $t_i \sim \theta^{\text{T}}_{z_i}$
  - If $t_i ==$ Lex:
    - Emit terminal symbol: $x_i \sim \phi^{\text{L}}_{z_i}$
  - If $t_i ==$ Left/Right/Unary:
    - Generate argument category: $y_i \sim \phi^{\text{Y}}_{z_i}$
    - Generate combinator: $c_i \sim \theta^{\text{C}}_{z_i, y_i}$
    - Deterministically create $z_{L(i)}$ (and $z_{R(i)}$ if binary)
- Recurse on children

Figure 6.2: The HDP-CCG has two base distributions, one over the space of categories and the other over words (or tags). For every grammar symbol, an argument distribution and emission distribution is drawn from the corresponding Dirichlet Processes. In addition, there are several MLE distributions tied to a given symbol for generating rule types, combinators and lexical tokens.

of parameter tying across all argument distributions. We discuss this further in section 6.3.1. First, we will describe how words are generated.

Generating Words

In our model, we generate words w conditioned on lexical categories z. This corresponds to first defining a random probability measure over words β^{L}, which we use as the base measure for a Dirichlet Process from which we draw a distribution over words for a specific lexical category z_i: $p(w \mid z_i) = \phi^{\mathrm{L}}_z$. Finally, the actual word x_i is drawn from $\phi^{\mathrm{L}}_{z_i}$.

The stick breaking distribution over β^{L} is abbreviated in this thesis as $\beta^{\mathrm{L}} \sim \mathrm{GEM}(\alpha^{\mathrm{L}})$, where GEM stands for Griffiths, Engen and McCloskey [128]. The stick-breaking process is parameterized by α^{L}, a single scalar which shapes the Beta distribution in the stick breaking process.

$$\begin{aligned}
\beta^{\mathrm{L}} &\sim \mathrm{GEM}(\alpha^{\mathrm{L}}) \\
\phi^{\mathrm{L}}_z &\sim \mathrm{DP}(\alpha^{\mathrm{L}}, \beta^{\mathrm{L}}) \\
x_i &\sim \phi^{\mathrm{L}}_{z_i}
\end{aligned}$$

To properly construct an HDP for lexical emissions requires that we define a distribution over the full, infinite, space of word spellings (β^{L}). β^{L} is a vector whose dimensions correspond to word spellings. In this way, the model can sample every new word from this distribution. One way this can be done is with a character based language model where the set of characters is fixed (perhaps to a specific language). The simplest possible model would be a unigram character model. This requires we have a distribution over characters in the language, and then the probability of any sequence is defined by the product of the probabilities for each individual character: $p(S) = \prod_{c \in S} p(c)$. This very simplistic model assigns a probability to every string in a language and that probability decreases with the length of the word. Using this distribution (or a more sophisticated one) for the base measure in our DP might have significant effects on the performance of the model.

In practice, we take a shortcut and assume we have seen all words in the training data and use an UNK token for rare words. Initial experiments will generate POS tags in lieu of actual words, in which case the entire space of tags will be seen during training. But once words are emitted, we will replace words with fewer than five occurrences with an UNK token. In this way, we

can compute the frequency of words in our corpus and use this distribution as the basis for β^{L}. This provides a very simple way to initialize and implement the model based on the available data, but means our implementation is actually a Hierarchical Dirichlet. The Hierarchical Dirichlet is an HDP over a finite set of atoms. Each level of the hierarchy still shares parameters through shared base measures but these measures are finite and we define them parametrically using the data.

In our implementation of the Hierarchical Dirichlet we will use a unigram word distribution over the corpus for initializing β^{L}. There are other initializations (e.g. random, uniform, etc.) for distributions over the vocabulary, which we did not evaluate. Additionally, as ϕ_z^{L} is the lexical distribution for the category z, it inherits from β^{L} and will therefore also be finite in our experiments.

Generating CCG Categories

The second generating process in our model is that of category arguments Y given a parent z:

$$\begin{aligned}\beta^{\mathrm{Y}} &\sim \mathrm{GEM}(\alpha^{\mathrm{Y}})\\ \phi_z^{\mathrm{Y}} &\sim \mathrm{DP}(\alpha^{\mathrm{Y}}, \beta^{\mathrm{Y}})\\ x_i &\sim \phi_{z_i}^{\mathrm{Y}}\end{aligned}$$

These distributions parallel those of the lexical emissions. We first define a stick breaking process (parameterized by α^{Y}) from which we draw a base distribution over argument categories in the corpus: β^{Y}. This serves as the base measure for drawing parent category specific argument distributions, ϕ_z^{Y}, from which a specific argument, x_i, is drawn.

While similar to the lexical distribution, the argument distribution presents two additional challenges to being infinite: first, categories are highly constrained and structured objects, and second, we cannot initialize β^{Y} with observed counts because the parses are not observed.

Structured Outcomes Producing structured objects requires that there be a meaningful relationship between categories which share internal structure. We therefore require a mechanism for providing them probabilities which is informed by the structure and ensures a shared reference between the same category when used in different contexts.

To handle the first issue, we will treat the space of categories as constrained by those observed when parsing the training data. But, one could create a model which would sample an unbounded set of novel categories by defining the HDPs base measure in terms of a simple weighted CFG. This allows for sampling new categories and assigning them a probability. We will illustrate this here:

$$
\begin{aligned}
&0.15 \quad \mathsf{Cat} \rightarrow (\ \mathsf{Cat} \setminus \mathsf{Cat}\ ) \\
&0.05 \quad \mathsf{Cat} \rightarrow (\ \mathsf{Cat} / \mathsf{Cat}\ ) \\
&0.40 \quad \mathsf{Cat} \rightarrow \mathsf{N} \\
&0.40 \quad \mathsf{Cat} \rightarrow \mathsf{S}
\end{aligned}
$$

Given a simple weighted CFG like the one above (written here with arbitrary probabilities) we can generate a CCG category with probability p by randomly sampling rewrite rules from the above grammar (and multiplying their rule probabilities to obtain p) until the derivation terminates in a string over the terminal alphabet $\Sigma = \{\mathsf{N}, \mathsf{S}, \ /, (,)\}$. In practice, atomic arguments are more common than complex ones and complex results are more common than complex arguments. Further research might investigate using a larger grammar which allows for capturing these interactions and assigning them different probabilities. Because there are two rules in the grammar which allow a Cat to introduce new Cats as results and arguments, these rules can be applied an arbitrary number of times to produce arbitrarily complex or recursively deep categories. In this way, there is no longest or most complex category licensed by the grammar, allowing for a distribution over the infinite space of categories.

Initializing the Argument Distributions Second, unlike lexical items (words or POS tags, which we can observe in the corpus to compute an initial distribution, the sentences are unlabeled, and as such, there is no gold data from which to estimate initial argument distributions. Unlike our baseline model which was initialized with a uniform distribution over outcomes (section 5.3), we initialize this distribution by assuming a uniform distribution over the sentences in the training data and a uniform distribution over the parses for a given sentence produced using the induced lexicon. Although a uniform distribution over outcomes is a simple baseline, it does not incorporate any of the grammar's biases into the model. A uniform distribution

over parses will bias the initial model towards the most common/useful constructions in the grammar.

Using this assumption, we compute pseudocounts for every argument in every training sentence and normalize. Specifically, the pseudocount for a given argument production within a specific chart item in a chart's forest is computed analogously to the standard inside-outside algorithm. Unlike inside-outside, which uses probabilities to compute pseudocounts, our initialization will recurse through the parse forest to count, for every chart item, the number of parses that involve this item. These "observed" counts will be then be divided by the total number of parses for the sentence to compute an initial distribution.

We know the total number of parses in the chart (*Total*), so our goal is to assess how much influence a specific chart item has on the chart as a whole. So, for each chart item *Parent* in each cell chart$[i][j]$, we consider each split point k ($i \leq k < j$), and each rule *Parent* $\rightarrow$ *Left Right* (if chart$[i][k]$ contains a chart item *Left* and chart$[k+1][j]$ an item *Right*). We first compute the number of inside parses each child has (*Left.parses* and *Right.parses*). The inside parses are computing by summing over all split points, and all left and right children at that split point. Next, analogously to the outside probability computation, we compute the number of parses that contain *Parent* at the given span. More precisely, a chart item's outside parses equal the product of its parent's outside parses and sister's parses summed over every (parent, sister) pair the given chart item has in the forest. We multiply the number of outside parses (*Parent.outside*) with the number of inside parses (*Left.parses* and *Right.parses*) and divide by the total number of parses in the chart (*Total*).

$$Count_{Parent \rightarrow Left\ \ Right} = Parent.outside \ \times \ Left.parses \ \times \ Right.parses$$

We then renormalize this count (divide it by *Total*) to get the fraction of parses that use a particular rule instantiation. These fractional counts can then be summed over all instantiations of a particular rule (i.e. all splits in the data i, k, j) to compute the expected count for the rule in this sentence. These expected counts are used to initialize the argument distribution. It computes the relative frequency of a specific rule in the grammar being used within a specific chart. In this way, chart items which are used by a larger

percentage of the parses in a chart will contribute more than those used infrequently.

MLE Parameters

Finally, we describe how to initialize and update the MLE distributions over the categories generated by TOP, over rule types (T), and over the combinators (C).

Initialization The distribution over categories generated by TOP is simply set to a uniform distribution over the only two allowed outcomes: N and S. For the rule types we need to define a distribution conditioned on each of the CCG categories. We assume these distributions are uniform (over the possible expansions Lex, Unary, Left, Right). For the distribution over combinators, we need a distribution conditioned on every (parent, argument) pair. Again, we assume this is a uniform distribution, but we define it over the set of combinators seen with the parent P and the argument X in the parse forests created during training. Because the set of combinators is fixed, this could easily be replaced with a uniform distribution over all combinators which the model then quickly learns to refine.

Training Updating these distributions is done via the inside-outside algorithm. We compute pseudocounts for every outcome of every distribution and normalize between rounds. This is in contrast to the variational EM that will be used for all other distributions (section 6.3.3).

6.3.2 Hyperparameters

When defining the HDP, new categories/words are drawn from H and given a probability β_k. We denote these distributions β^{Y} for arguments, and β^{L} for lexical emissions. These are then used to define the mean for the next DP in the hierarchy. The DP also requires a notion of variance, or precision, which determines how similar individual draws will be. This precision is determined by the magnitude of the hyperparameter α^{Y}. We have an identical parameter α^{L} for controlling variance in the lexical distribution β^{L}). α^{L} controls how much evidence is necessary for a distribution of lexical productions to diverge from a unigram base DP over terminal symbols.

These α parameters can have a significant effect on the performance of a model, but since each distribution could have its own set of hyperparameters, we cannot feasibly optimize them individually. For this reason, we follow the example of Liang et al. [129] and use schemes for specifying their values. For simplicity, we use the same scheme for setting the values for α^{L} as for α^{Y}. A proper search or optimization over parameters may yield better results, but our primary goal is to limit the number of tunable parameters in our model. We present initial experiments where we vary the value of α^{Y} as a function of the number of outcomes allowed by the grammar for argument categories or the corpus in the case of terminal symbols in section 7.1. Specifically, we set $\alpha^{\mathrm{Y}} = n^p$ for conditioning contexts with n outcomes, following Liang et al. Similarly, we set $\alpha^{\mathrm{L}} = n^p$ for the lexical emissions where n is the number of lexical types (part-of-speech initially) in the corpus. Since Liang et al. [129] found that the ideal value for α appears to be super-linear but sub-quadratic in n, we present results where p takes the values 0, 1.0, 1.5, and 2.0 to explore the range from uniform to quadratic. This setting for p is the only free parameter in the model. By controlling precision, we can tell the model to what extent global corpus statistics should be trusted.

One obvious concern with this scheme for setting the hyperparameters is that as the space of outcomes n grows, the value of the hyperparameter grows polynomially in n, with $p \leq 2$. Because the hyperparameter limits the variance between draws from the base measure and specifies the amount of evidence that is required to diverge from the base distribution, the larger the value, the less variance is permitted and the harder it is to learn an empirical distribution. In the extreme case, all draws are forced to be identical to the mean of the base distribution, and learning is made effectively impossible, as the amount of empirical observations required to diverge from the base measure will also grow in n^2. This becomes a problem when lexicalizing the emissions, as we replace the set of $\sim$30-50 part-of-speech tags with the size of the vocabulary of a language (V), and V^2 will be many orders of magnitude larger than we have sentences in our training data. For this reason, experiments later in this thesis (Chapter 7.2.1) will set α^{L} and α^{Y} to a constant (2500). It is possible that curriculum learning techniques [49, 130] or grid search might provide a better mechanism for choosing and tuning this value.

6.3.3 Variational EM

One advantage of the argument model is that it only requires a single distribution over categories for each binary tree. In contrast to similar proposals for CFGs [129], which impose no formal restrictions on the non-terminals A, B, C that can appear in a rewrite rule $\mathsf{A} \rightarrow \mathsf{B}\ \mathsf{C}$, this greatly simplifies the modeling problem (yielding effectively a model that is more akin to non-parametric HMMs), since it avoids the need to capture correlations between different base distributions for Y and Z. In contrast, Liang et al. [129] use a CFG and attempt to avoid modeling correlations by factorizations like $p(\mathrm{B}\ \mathrm{C}) = p(\mathrm{B})p(\mathrm{C})$.

HDPs need to be estimated with approximate techniques. As an alternative to Gibbs sampling [121], which is typically very slow and is only exact in the limit, variational inference algorithms [131, 132] traditionally estimate the parameters of a truncated model to maximize a lower bound of the log-likelihood of the actual model. This allows for factorization of the model and a training procedure analogous to the inside-outside algorithm [113], allowing training to run very quickly and in a trivially parallelizable manner.

As briefly mentioned earlier, we initialize our base measure for lexical emissions with the empirical unigram counts, and our base measure for CCG arguments with frequency counts from the parses produced on the training data. Training is non-convex and, therefore, there are many local optima the model may converge to. We did not explore random initialization and how the model's performance might be affected. Instead, we explored two options: first, we initialized all distributions to be uniform over observed outcomes, and second, we initialized all distributions with empirical frequency counts from the training data (Section 6.3.1). We found the latter performed better. During training, all distributions are updated, including the re-estimation of the base DPs.

In variational inference, multinomial weights W take the place of probabilities [129]. Specifically, the probability of outcome Y given parent P, $P_{\mathsf{P}}(\mathsf{Y})$, is replaced with $W_{\mathsf{P}}(Y)$. The weights for an outcome Y with conditioning variable P are computed by summing pseudocounts with a scaled mean vector from the base DP. The computation involves moving in the direction of the gradient of the Dirichlet distribution, which results in the following (non-exponentiated) difference of Digammas where $\Psi(x) = \frac{d}{dx}\log\,\Gamma(x)$:

Without HDP:

$$W_P(Y) = E\ \log\phi_P(Y) = \Psi(C(\mathsf{P},\mathsf{Y})) - \Psi(C(\mathsf{P},*))$$

With HDP:

$$W_P(Y) = \Psi(C(\mathsf{P},\mathsf{Y}) + \alpha^P\beta_Y) - \Psi(C(\mathsf{P},*) + \alpha^P)$$

Here, we have used $C(\mathsf{P},\mathsf{Y})$ to denote the pseudocounts (expected counts) for seeing argument Y with parent P. These are computed using the standard inside-outside algorithm. Next, the normalization is performed over the sum of the pseudocounts for parent P with any argument. We denote this as $C(\mathsf{P},*)$ where the * indicates that any and all arguments should be counted.

Importantly, the Digamma and multinomial weights comprise a rich-get-richer scheme, biasing the model against rare outcomes. In addition, since variational inference requires the same two step (1. compute counts 2. normalize) process as EM, it is trivially parallelizable. The counts are computed by the inside-outside algorithm on a per sentence level and then aggregated. This means the computation can be split into N/c chunks for N sentences and c cores. This divide-and-conquer approach lets us compute the counts in small parallel batches before aggregating them and updating the distributions via the equation above to complete one iteration of training. In practice, by limiting ourselves to training and testing our models on the short sentences (up to 15 words not counting punctuation) in each of the corpora, training takes between one minute to at most three hours on a single 12-core machine (with 96GB of RAM) depending on corpus size. The size of each corpus is presented in Table 7.1. This computation time will grow as a function of the grammar's complexity, amount of data, and sentence length in the upcoming sections.

6.4 Capturing More Complicated Phenomena

In our experiments with a PCFG model (Chapter 5), we found that simpler grammars were the best performing. We therefore kept a simple paradigm for initial experiments with our new argument factored CCG model (Chapter 7.1). Given that our new model greatly outperforms the PCFG, it is appropriate to re-examine the model's performance when higher arity categories, composition, and type-raising are included. Further, since the HDP better

handles sparsity and smoothing, we will investigate the effects of lexicalization and try to capture constraints imposed by punctuation [133].

The next few sections detail how the model can be extended to handle these phenomena. These extensions will give rise to the final and best performing model in this thesis, which will allow us to revisit our original motivation, the creation of a cog to replace supervised parsing. Specifically, in section 7.2 we will perform a novel labeled dependency evaluation of these extensions.

6.4.1 Increasing Grammatical Complexity

We are interested in allowing the model to explore the use of composition of arity two (B^2) and three (B^3), and in allowing complex arguments (section 4.4.3) as well as type-raising. This translates to removing restrictions on parse operations and induction, while increasing the number of rounds of lexicon induction performed.

These changes present the model with a much broader and more ambiguous search space than we entertained for the PCFG (Figure 5.1). Additionally, unlike in that model, we will investigate allowing the semantically necessary preposition vs. possessive ambiguity of categories of the form $(\mathsf{X}\backslash\mathsf{X})/\mathsf{X}$ and $(\mathsf{X}/\mathsf{X})\backslash\mathsf{X}$ (removing restriction 7 of Section 4.2.2). This introduces the possessive for $(\mathsf{N}/\mathsf{N})\backslash\mathsf{N}$ for English and a preposition for Japanese $(\mathsf{N}\backslash\mathsf{N})/\mathsf{N}$, which uses postpositions. This also introduces both $(\mathsf{S}\backslash\mathsf{S})/\mathsf{S}$ and $(\mathsf{S}/\mathsf{S})\backslash\mathsf{S}$ for every language.

We only introduce a a limited set of complex arguments: $\mathsf{S}\backslash\mathsf{N}$ and S/N. This means we are not allowing non-modifier categories with arguments of the form $\mathsf{S}|\mathsf{S}$ or $\mathsf{N}|\mathsf{N}$. Categories that take modifiers as arguments but are not themselves modifiers (e.g. $(\mathsf{S}\backslash\mathsf{N})/(\mathsf{N}\backslash\mathsf{N})$) do exist in CCGbank, but are very rare. In particular, the only category of this form in CCGbank is $((\mathsf{S}\backslash\mathsf{N})/(\mathsf{N}\backslash\mathsf{N}))/\mathsf{N}$ which only occurs a total of 12 times (for the words is and was).

6.4.2 Punctuation

Spitkovsky et al. [51] performed a detailed analysis of punctuation for dependency-based grammar induction and proposed a number of constraints that aimed to capture the different ways in which dependencies might cross

constituent boundaries implied by punctuation marks.

A constituency-based formalism like CCG allows us instead to define a very simple, but effective Dirichlet Process (DP) based Markov grammar that emits punctuation marks at the maximal projections of constituents. We note that CCG derivations are binary branching and that virtually every instance of a binary rule in a normal-form derivation combines a head X|Y or X with an argument Y or modifier X|X.

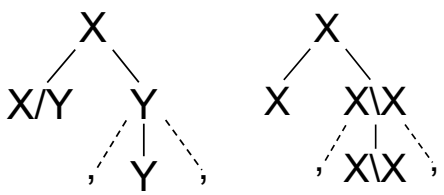
In these two configurations the punctuation may appear between the two combining categories or after them. this allows for four possible attachment points in each case. For X/Y Y, the punctuation between them can attach to either category, and the punctuation following to either the argument Y or result X. Analoguously, for X X\X, the punctuation between them can attach to either category, and the punctuation that follows can attach to either the modifier X\X or the result X.

The important insight is that when a punctuation mark appears before a category, it can only belong in one of three states:

1. Either the category will be taken as an argument of (or serve as a modifier to) a constituent that immediately precedes it
2. The category will be a constituent's argument or modifier
3. The category spans the entire sentence

By insisting that categories attach at the maximal projection, we eliminate extraneous attachments. In the final case, the punctuation mark is attached before applying the rule TOP → X. We have the same constraint for all unary rules (type-raising/type-changing), that is, they should only be applied after punctuation has been attached.

Without reducing the set of strings generated by the grammar, we can, therefore, assume that in the binary case, punctuation marks can only be attached to the argument Y or the adjunct X|X. Here, Y and X|X are maximal projections, since the head word of the parent X comes from the child X/Y (or X).



This constraint does not impose any restrictions on the set of allowed strings, but only reduces redundant, ambiguous analyses. In every parse produced by our grammar, every instantiation of a binary rule includes one category being taken as an argument or being modified by another category. If we had only the first comma in the two examples above, there would be two identical analyses for each string: the comma could attach to the left, yielding ((X/Y ,) Y) or ((X/X ,) X), or to the right, yielding (X/Y (, Y)) or (X/X (, X)). This restriction removes that spurious redundancy by forcing attachment to the argument (X/Y (, Y)) or the modifier ((X/X ,) X). This restriction also biases the grammar towards analyses where punctuation marks bracket meaningful constituents.

To model this, for each maximal projection (i.e. whenever we generate a non-head child) with category C, we first decide whether punctuation marks should be emitted ($M = \{true, false\}$) to the *left* or *right* side (Dir) of C. Since there may be multiple adjacent punctuation marks (... ."), we treat this as a Markov process in which the *history* variable captures whether previous punctuation marks have been generated or not. We should note that nothing about this restriction requires adjacent marks to be generated by or attached to the same tree. They might both attach to different, adjacent, constituents which combine later in the parse. Finally, we generate an actual punctuation mark w_m:

$$
\begin{aligned}
&p(M \mid Dir) && \sim DP(\alpha, p(M)) \\
&p(M \mid Dir, Hist) && \sim DP(\alpha, p(M \mid Dir)) \\
&p(w_m \mid Dir, Hist, M = True) && \sim DP(\alpha, p(w_m)) \\
&p(w_m \mid Dir, Hist, \mathsf{C}, M = True) && \sim DP(\alpha, p(w_m \mid Dir, Hist))
\end{aligned}
$$

The base distributions are $p(M)$, the global probability of a constituent emitting punctuation, and $p(w_m)$, the observed probabilities of punctuation marks.

The only exception to this punctuation treatment are the symbols # and $. These are treated as ordinary lexical items for which CCG categories will be induced by the regular induction algorithm. All other punctuation follows this scheme, including quotes and brackets. Commas and semicolons (,, ;) can act both as punctuation marks generated by this Markov grammar, and as conjunctions with lexical category conj.

6.4.3 Lexicalization

As discussed earlier (section 6.3.1), our work until now, in keeping with most work in grammar induction, treats POS tags t rather than words w as the terminals generated by lexical categories c. The advantage of this approach is that tag-based emissions $p(t \mid c)$ are a lot less sparse than word-based emissions $p(w \mid c)$. It is, therefore, beneficial to first train a model that emits tags rather than words [134], and then to use this simpler model to initialize a lexicalized model that generates words instead of tags.[2]

To switch our lexical emissions, expected lexical counts for words are computed using their tag probabilities, $p(t \mid c)$, during the E-step. Those counts, having been allocated to specific (word, category) pairs, can be normalized during the M-Step to estimate $p(w \mid c)$. Inside-outside can then continue as before. The only effect on the variational inference discussed is that we are now computing multinomial weights for individual (word, category) pairs, including an UNK token, rather than over (tag, category) pairs.

Many words, e.g. prepositions and verbs, differ systematically in their preferred syntactic role from that of their part-of-speech tags. For example, both *of* and *with* are tagged as **IN**, but our models will correctly discover that *of* is far more likely to be generated as a noun attaching preposition, $(\mathsf{N}\backslash\mathsf{N})/\mathsf{N}$, and *with* by a verb attaching category, $(\mathsf{S}\backslash\mathsf{S})/\mathsf{N}$. We see a similar effect with the words *said* and *fell* which are both tagged as **VBD**. Both words have a strong bias to be transitive but *said* takes a sentence as its second argument, while *fell* takes a noun:

	IN			**VBD**	
Word	Category	$p(c \mid w)$	Word	Category	$p(c \mid w)$
of	$(\mathsf{N}\backslash\mathsf{N})/\mathsf{N}$	0.60	said	$(\mathsf{S}\backslash\mathsf{N})/\mathsf{S}$	0.35
	$(\mathsf{S}\backslash\mathsf{S})/\mathsf{N}$	0.17		$\mathsf{N}\backslash\mathsf{N}$	0.12
	N/N	0.09		$\mathsf{S}\backslash\mathsf{N}$	0.11
with	$(\mathsf{S}\backslash\mathsf{S})/\mathsf{N}$	0.56	fell	$(\mathsf{S}\backslash\mathsf{N})/\mathsf{N}$	0.44
	$(\mathsf{N}\backslash\mathsf{N})/\mathsf{N}$	0.17		$\mathsf{S}\backslash\mathsf{N}$	0.28
	N/N	0.13		$(\mathsf{N}\backslash\mathsf{N})/\mathsf{N}$	0.01

[2]Our choice of this scheme which switches to generating words rather than generating both tags and words ($p(t \mid c) \times p(w \mid t, c)$) was based on poor empirical results where we found it difficult to have the model diverge from $p(t \mid c)$ given the new lexical evidence.

These results are for the model $\mathbf{B_3^{P\&L}}$, discussed and evaluated fully in the next chapter (section 7.2.1).

6.5 Conclusions

This chapter presented a novel factorization of CCG (the argument model). We present MLE and HD(P) formulations of the model. We outline both our current training procedures with variational EM as well as a description for how non-parametric extensions might be implemented. Finally, we provide three ways in which the argument model can be extended to incorporate additional grammatical complexity, punctuation and lexicalization. In the next chapter, we evaluate the MLE and HD(P) formulations of our model in 10 languages and demonstrate the utility of our extensions when evaluated against English CCGbank.

Chapter 7

Multilingual Evaluation

As is standard for this task, we evaluate our systems against a number of different dependency treebanks, and measure performance in terms of the accuracy of directed dependencies (i.e. the percentage of words in the test corpus that are correctly attached). To demonstrate the performance across a number of languages we use the data from the PASCAL challenge for grammar induction [63], the data from the CoNLL-X shared task [135] and Goldberg's Hebrew corpus [136].

7.1 Unlabeled Dependency Evaluation

We evaluate our system on 13 different languages. In each case, we follow the test and training regimes that were used to obtain previously published results in order to allow a direct comparison. We compare our system to the results presented at the PASCAL Challenge on Grammar Induction [63, 118],[1] as well as to Gillenwater et al. [137] and Naseem et al. [138]. We use Nivre's [61] Penn2Malt implementation[2] of Collins' [64] head rules to translate the WSJ Penn Treebank [1] into dependencies. Finally, when training the MLE version of our model, to prevent issues with numerical precision we define a small rule probability (e^{-15}) that prevents any rule used during training from

Work in this chapter was first published in Y. Bisk and J. Hockenmaier, "An HDP Model for Inducing Combinatory Categorial Grammars," *Transactions of the Association for Computational Linguistics*, pp. 75-88, 2013.[119] and Y. Bisk and J. Hockenmaier, "Probing the linguistic strengths and limitations of unsupervised grammar induction," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Beijing,China, July 2015. [133] and is reprinted here with permission by the copyright holder.

[1] Numbers are from personal correspondence with the workshop organizers. The previously published numbers are not comparable to literature due to an error in the evaluation. `http://wiki.cs.ox.ac.uk/InducingLinguisticStructure/ResultsDepComparable`

[2] `http://w3.msi.vxu.se/~nivre/research/Penn2Malt.html`

shrinking to zero. When performing analysis on English and Portuguese we found no effect on performance when varying the small rule value between e^{-7} to e^{-20}.

7.1.1 PASCAL Challenge on Grammar Induction

In Table 7.1, we compare the performance of the basic Argument model (MLE), of our HDP model with four different settings of the hyperparameters (Section 6.3.2) and of the systems presented in the PASCAL Challenge on Grammar Induction [63]. The systems in this competition were instructed to train over the full data set, including the unlabeled test data. The competing systems include our CCG PCFG model (PCFG) (Section 5.5.3) from the previous chapter, Cohn and Blunsom's [36] re-implementation of Klein and Manning's [41] DMV model in a tree-substitution grammar framework (BC), as well as three other dependency based systems which either incorporate Naseem et al.'s [37] rules in a deterministic fashion [139], rely on extensive tuning on the development set [43] or incorporate additional tokens from Wikipedia to estimate model parameters [140]. We ignore punctuation for all experiments reported in this section of the thesis, but since the training data (but not the evaluation) includes punctuation marks, participants were free to choose whether to include punctuation or ignore it.

While our PCFG is the only other system with directly interpretable linguistic output, we also include a direct comparison with Blunsom and Cohn (BC), whose Tree-Substitution Grammar (TSG) representation is equally expressive to ours. Finally, we present a row with the maximum performance among the other three models. As we have no knowledge of how much data was used in the training of other systems, we simply present results for systems trained on length 15 (not including punctuation) sentences and then evaluated at lengths 10 and 15.

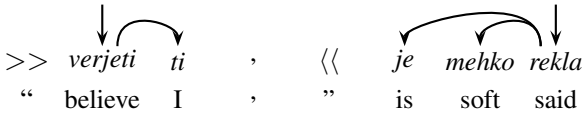
		Arabic	Danish	Slovene	Swedish	Dutch	Basque	Portuguese	WSJ	Childes	Czech
	# Tokens	5,470	25,341	54,032	61,877	78,737	81,345	158,648	163,727	290,604	436,126
	# Tags	20	24	36	30	304	14	23	36	69	62
PASCAL	**BC**	60.8/58.4	44.7/39.4	62.6/57.9	63.2/56.6	**51.8/52.0**	53.0/**48.9**	52.4/50.2	68.6/63.3	47.4/46.1	47.9/43.1
PASCAL	**Max**	**67.2/66.8**	**60.1/56.0**	**65.6/61.8**	**72.8/63.4**	51.1/47.6	**53.7**/47.8	67.0/61.8	**71.2/64.8**	56.0/54.5	**58.3/54.4**
PASCAL	**PCFG**	41.6/43.7	46.4/43.8	49.6/43.9	63.7/57.0	49.7/43.6	45.1/39.6	**70.8/67.2**	68.2/59.6	**61.4/59.8**	45.0/38.9
Arg Model	**MLE**	41.6/42.9	43.4/39.2	46.1/41.1	70.1/59.7	52.2/47.2	29.6/26.5	62.2/59.7	59.5/52.4	53.3/51.9	50.5/45.8
Arg Model	**HDP**$_{0.0}$	48.0/50.0	**63.9/58.5**	44.8/39.8	67.6/62.1	45.0/33.9	41.6/39.1	71.0/66.0	59.8/52.9	56.3/55.2	54.0/49.0
Arg Model	**HDP**$_{1.0}$	45.6/47.1	45.7/42.3	53.9/46.9	**74.5/66.9**	**58.5/54.4**	50.1/44.6	65.1/60.6	64.3/56.5	71.5/70.3	**55.8/50.7**
Arg Model	**HDP**$_{1.5}$	49.6/50.4	58.7/54.4	53.2/48.2	74.3/67.1	57.4/54.5	**50.6/45.0**	70.0/64.7	65.5/57.2	69.6/68.6	55.6/50.3
Arg Model	**HDP**$_{2.0}$	**66.4/65.1**	56.5/49.5	**54.2/46.4**	71.6/64.1	51.7/48.3	49.4/43.3	**76.3/70.5**	**70.7/62.9**	**74.1/73.3**	54.4/48.5
	+/−	-0.8/-1.7	+3.8/+2.5	-11.4/-15.4	+1.7/+3.5	+6.7/+2.4	-3.1/-3.9	+5.5/+3.3	-0.5/-1.9	+12.7/+13.5	-2.5/-3.7

Table 7.1: A comparison of the basic Argument model (MLE) and four hyper-parameter settings of the HDP-CCG against two syntactic formalisms that participated in the PASCAL Challenge [63], PCFG [118] and Blunsom and Cohn (BC) [42], in addition to a max over all other participants. We trained on length 15 data (punctuation removed), including the test data as recommended by the organizers. The last row indicates the difference between our best system and the best of the competition [139, 43, 140]. Results are presented at length 10/15.

The MLE version of our model shows rather variable performance: although its results are particularly bad on Basque (Eu), it outperforms both the PCFG and BC on some other settings. By contrast, the HDP system is always better than the MLE model. It outperforms all other systems on half of the corpora. On average, it outperforms the PCFG and BC by 10.3% and 9.3% on length $\leq$10, or 9.7% and 7.8 % on length $\leq$15 respectively.

The initialization is not random (Section 6.3.1) and so there is no variability between runs of the HDP unless the hyperparameters are changed. Unfortunately, it is not clear how to automatically choose between models or hyperparameter settings as no clear pattern is immediately apparent, nor does performance correlated with the model's likelihood. Making these decisions automatically is a potentially fruitful direction for future work [45].

The main reason why our system does not outperform BC by an even higher margin is the very obvious 11.4%/11.5% deficit on Slovene. However, the Slovene dependency treebank seems to follow a substantially different annotation scheme. In particular, the gold standard annotation of the 1,000 sentences in the Slovene development set treats many of them as consisting of independent sentences (often separated by punctuation marks that our system has no access to), so that the average number of roots per sentence is 2.7:



When our system is presented with these short components in isolation, it oftentimes analyzes them correctly, but since it has to return a tree with a single root, its performance degrades substantially. We did not investigate ways to determine if punctuation indicates that a verb should be detached, but using a small amount of seed knowledge about the corpus might provide large gains in performance.

We believe the HDP performs so well as compared to the MLE model (Figure 7.1) because of the influence of the shared base distribution, which allows the global category distribution to influence each of the more specific distributions. One example of this effect is when choosing what argument Y to produce a parent category P, i.e. the distribution $p(\mathsf{Y} \mid \mathsf{P})$. While

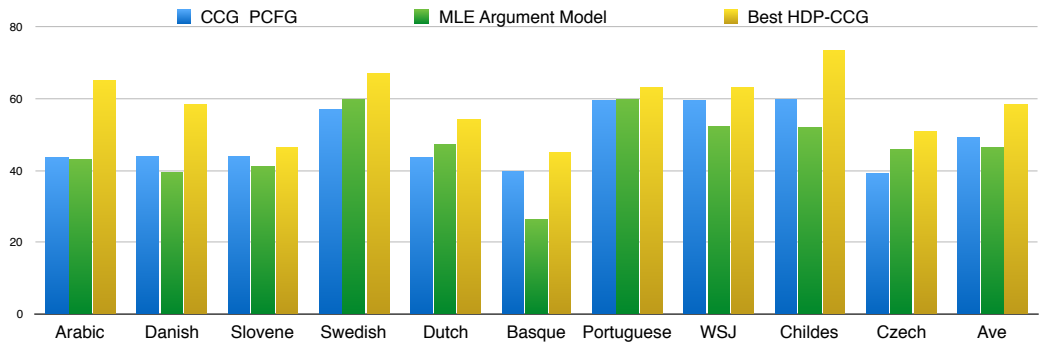


Figure 7.1: Comparison of our PCFG, MLE argument model and the best of our HDP models' performances on directed accuracies (length 15) for 10 languages.

S is a more common category, N is a more common argument. This parameter sharing provides an informative bias to rare categories being used to complete a parse. The same is true with lexical distributions where an infrequent lexical category can fall back on the unigram distribution when generating. Further, hyperparameters provide a very simple knob that has a substantial effect on performance. A side effect of the hyperparameters is that their strength also determines the rate of convergence. This may be one of the reasons for the high variance seen in the four settings tested, although we again should note that since our initialization is always uniform (Section 6.3.1) in the parse forest, and not random, so consecutive runs do not introduce variance in the model's performance.

7.1.2 Systems with Linguistic Constraints

Since our induction algorithm is based on the knowledge of which POS tags are nouns and verbs, we compare our approach to both that of Naseem et al. (Section 4.3) who use universal knowledge to constrain the learner, and against Boonkwan and Steedman (Section 4.3), who incorporate knowledge about the lexicon into their system.

Naseem et al.: Universal Knowledge In Table 7.2 we compare our system to Naseem et al. [37], who present a non-parametric dependency model that incorporates thirteen universal linguistic constraints. Three of these constraints correspond to our rules that verbs are the roots of sentences and may take nouns as dependents, but the other ten constraints (Sec. 4.2.2) have

no equivalent in our system. Although our system has less prior knowledge, it still performs competitively.

	Sl	**Es**	**Da**	**Pt**	**Sv**
∼#Tokens	3.8K	4.2K	9.5K	15K	24K
N10	50.9	**67.2**	**51.9**	71.5	63.3
HDP	**56.6**	62.1	51.5	**74.7**	**69.8**

Table 7.2: A comparison of our system with Naseem et al. (2010), both trained and tested on the length 10 training data from the CoNLL-X Shared Task.

On the WSJ, Naseem et al. demonstrate the importance and effect of the specific choice of syntactic rules by comparing the performance of their system with hand crafted universal rules (71.9), with English specific rules (73.8), and with rules proposed by Druck et al. [141] (64.9).

The performance of Naseem et al.'s system drops very significantly as sentence length (and presumably parse complexity) increases, whereas our system shows significantly less decline, and outperforms their universal system by a significant margin. We saw a similar stability in the PCFG model from before (Figure 5.3).

	≤ 10	≤ 20
Naseem Universal Rules	71.9	50.4
Naseem English Rules	**73.8**	**66.1**
HDP-CCG	68.2	64.2
HDP-CCG (train ≤ 20)	71.9	

In contrast to Spitkovsky et al. [49], who reported that performance of their dependency based system degrades when trained on longer sentences, our performance on length $\leq$10 sentences increases to 71.9 when we train on sentences up to length $\leq$20. This is a particularly promising result. We believe that this correlates to introducing far more evidence for prepositional phrases and their internal noun-phrases, a pair of very common construction in Newswire text.

Boonkwan and Steedman: Knowledge from a Linguist Another system that is also based on CCG, but captures significantly more linguistic

knowledge than ours, was presented by Boonkwan and Steedman [38], who achieve an accuracy of 74.5 on WSJ10 section 23 (trained on sections 02-22). When evaluating our model on the same train/test split, our system achieves an accuracy of 68.4. Unlike our approach, Boonkwan and Steedman do not automatically induce an appropriate inventory of lexical categories, but use a questionnaire that defines prototype categories for various syntactic constructions, and requires manual engineering of which POS tags are mapped to what categories to generate a language-specific lexicon. However, their performance degrades significantly when only a subset of the questions is considered. Using only the first 14 questions, covering facts about the ordering of subjects, verbs and objects, adjectives, adverbs, auxiliaries, adpositions, possessives and relative markers, they achieve an accuracy of 68.2, which is almost identical to ours, even though we use significantly less initial knowledge. However, the lexicons we present below (Table 7.4) indicate that we are in fact learning many of the very exact details that in their system are constructed by hand. The remaining 14 questions in Boonkwan and Steedman's questionnaire cover less frequent phenomena such as the order of negative markers, dative shift, and pro-drop. The obvious advantage of this approach is that this allows them to define a much more fine-grained inventory of lexical categories than our system can automatically induce. We also stipulate that for certain languages knowledge of pro-drop could play a significant role in the success of their approach: if complete sentences are allowed to be of the form $S\backslash N$ or S/N, the same lexical category can be used for the verb regardless of whether the subject is present or has been dropped.

	Sl	**Es**	**Da**	**Pt**	**Sv**
#Tokens	3,857	4,230	9,549	15,015	24,021
G10	51.2	62.4	47.2	54.3	48.6
HDP-CCG	**57.9**	**65.4**	**49.3**	**73.5**	**73.2**
	Bg	**WSJ**	**Nl**	**Ja**	**De**
#Tokens	38,220	42,442	43,405	43,501	77,705
G10	59.8	64.4	47.5	60.2	47.4
HDP-CCG	**66.1**	**70.3**	**56.2**	**64.1**	**68.4**

Table 7.3: A comparison of our system with Gillenwater et al. (2010), both trained on the length 10 training data, and tested on the length 10 test data, from the CoNLL-X Shared task.

7.1.3 Additional Languages

In order to provide results on additional languages, we present in Table 7.3 a comparison to the work of Gillenwater et al. [142] (G10), using the CoNLL-X Shared Task data [135]. Following Gillenwater et al. , we train only on sentences of length 10 from the training set and evaluate on the test set. Since this is a different training regime, and these corpora differ for many languages from that of the PASCAL challenge, numbers from Table 7.1 cannot be compared directly with those in Table 7.3.

7.1.4 Automatic Tagging/Segmentation in Hebrew

Finally, we are unaware of any existing work on grammar induction in Hebrew. Goldberg (2011) [136] introduced a small Hebrew corpus which is available in two formats: 1. Gold POS tags and morpheme segmentations 2. Automatically tagged and segmented morphemes. As a morphologically rich language, the segmentation of words into morphemes is often difficult and an automatic segmentation should introduce far more noise than automatic tagging in language like English. We applied our model to the gold annotated version of Goldberg's Hebrew corpus and achieved an accuracy of 62.1 (trained and tested on all sentences length 10; 7,253 tokens) and 59.6 (length 15; 21,422 tokens). In this discussion, token refers to morpheme, not the original white-space delimited words.

In the Shared Task experiments in Portuguese some automatically tagged data was included as part of the corpus, but Hebrew provides us the opportunity to test both automatic tagging and segmentation on a low resource language. Due to the limited amount of data, we present results where we train and test on the union of the data at lengths 10 and length 15. This corresponds to ∼6,000 and ∼18,000 tokens respectively. As there does not exist a Universal POS mapping for Hebrew, we constructed one with help from Yoav Goldberg (reproduced in Appendix A.1).

	Gold Tags	Auto Tags
0.0	57.4/56.7	55.9/48.8
1.0	**62.1**/**59.6**	**59.2**/54.3
1.5	57.6/57.4	52.6/**54.9**
2.0	33.9/51.1	31.7/48.7

We notice that while performance does degrade it is not a substantial drop, which is encouraging for our next steps, which further limit our access to part-of-speech tags. In the future, this experiment should be scaled up to a larger class of languages with varying morphological richness.

7.1.5 The Induced Lexicons

Since our approach is based on a lexicalized formalism such as CCG, our system automatically induces lexicons that pair words (or, in our case, POS-tags) with language-specific categories that capture their syntactic behavior. If our approach is successful, it should learn the basic syntactic properties of each language, which will be reflected in the corresponding lexicon. In Figure 7.4 one sees how verbs subcategorize differently, how word ordering differs by language, and how the attachment structures of prepositions are automatically discovered and differ across languages.

In the induction algorithm a tremendous number of incorrect verbal categories are introduced for every language, and the model typically converges converges on one that assigns most of its probability mass to the correct word order. Interestingly, in contrast, in Arabic the model learns that word order is variable, and therefore the verb must allow for both SVO and VOS style constructions and splits the mass appropriately. We generally learn that adpositions (prepositions or postpositions) take nouns as arguments. In Czech, PPs can appear before and after the verb, leading to two different categories ($(\mathsf{S}\backslash\mathsf{S})/\mathsf{N}$ and $(\mathsf{S}/\mathsf{S})/\mathsf{N}$). Japanese has postpositions that appear in preverbal position ($(\mathsf{S}/\mathsf{S})\backslash\mathsf{N}$), but when this category is assigned to nominal particles that correspond to case markers, it effectively absorbs the noun, leading to a preference for verbs that do not take any arguments (S), and to a misanalysis of adjectives as verb modifiers (S/S).

Our lexicons also reflect differences in style: while Childes and the WSJ are both English, they represent very different registers. We learn that subjects are mostly absent in the informal speech and child-directed instructions contained in Childes, leading to categories such as S for intransitive verbs and S/N for transitive verbs, while effectively mandatory in the Wall Street Journal, allowing us to infer $\mathsf{S}\backslash\mathsf{N}$ for intransitives and $(\mathsf{S}\backslash\mathsf{N})/\mathsf{N}$ for transitive verbs instead. In principle, it is possible to have the model capture these missing subjects by allowing parses to terminate in $\mathsf{TOP} \rightarrow \mathsf{S}|\mathsf{N}$. We did not

	Arabic	%	**Swedish**	%	**WSJ**	%	**Childes**	%	**Japanese**	%	**Czech**	%
VERB	(S\N)/N	56	S	45	S\N	52	S/N	44	S	84	S	26
	(S/N)/N	29	S\N	20	(S\N)/N	19	S	37			S\N	25
ADP	N\N	68	(S\S)/N	49	(S\S)/N	46	(S\S)/N	45	(S/S)\N	44	(S\S)/N	42
	N/N	21	(N\N)/N	25	(N\N)/N	20	N/N	25	N\N	23	(S/S)/N	26
NOUN	N\N	50	N	91	N	79	N	89	N	73	N	74
	N	35			N/N	12						
ADJ	N\N	82	N/N	50	N/N	70	N/N	46	S/S	64	N/N	55

Table 7.4: Partial lexicons demonstrating language specific knowledge learned automatically for five languages. For ease of comparison between languages, we use the universal tag label (Verb, Adposition, Noun and Adjective). Shown are the most common categories and the fraction of occurrences of the tag that are assigned this category (according to the Viterbi parses).

explore the effects of allowing these parses into our model. Later experiments with complex arguments may indicate that the models presented here are not powerful enough to learn when to use such an analysis.

7.2 Labeled Evaluation Against CCGbank

Hyperparameter Settings As noted previously, all of the following experimental results will be reported with hyperparameters (α^{L} and α^{Y}) set to a constant value of 2500. We are making this change from the schemes explored in section 6.3.2 and Figure 7.1 because we are introducing words as lexical emissions. This means it no longer makes sense to have hyperparameters set as a function of the size of the output. Instead, we tested three constant values (1000, 2500, 5000) and found that the basic model we are about to extend performed closest to the dependency evaluation in Section 7.1.2 with a constant of 2500. We will fix this hyperparameter setting for experimental simplicity, but a more rigorous grid search might find better parameters for the complex models.

Finally, all numbers henceforth, unless otherwise specified, will be based on our labeled evaluation for CCGbank (Section 3.4). Therefore, they will be lower, but much more informative than those we reported previously to compare with the existing literature. This evaluation will allow for an in-depth analysis in Section 7.2.2.

7.2.1 Evaluation

For our experiments, we will follow the standard practice in supervised parsing of using WSJ Sections 02 through 21 for training, Section 22 for development and error analysis, and a final evaluation of the best models on Section 23. As noted in the induction section, the grammars induced grow rapidly (Table 4.3) as complexity is added/allowed. Correspondingly, the memory footprint required to keep all parse forests for the training data in RAM quickly grows beyond the RAM we have available (96GB). For this reason, we only train on sentences that contain up to 20 words (as well as an arbitrary number of punctuation marks). All analyses and evaluation are performed with sentences of all lengths unless otherwise indicated.

		Base	+ Lexicalization	+ Punctuation	+ Punc & Lex	+ Allow (X|X)|X
Only Atomic Arguments	B^1	**34.2 / 60.2**	35.2 / 61.0	36.3 / 63.7	36.9 / 63.7	36.8 / 63.5
(S, N)	B^3	34.4 / 59.2	35.1 / 60.4	33.8 / 61.9	38.9 / 65.7	**38.8 / 65.7**
Allow Complex Arguments	B^1	33.0 / 58.7	34.9 / 61.7	33.2 / 61.2	35.7 / 63.4	**35.8 / 63.5**
(S, N, S|N)	B^3	29.4 / 52.2	29.5 / 53.6	31.2 / 54.7	31.2 / 55.0	31.2 / 54.7

Table 7.5: The impact of our changes to the our previous model (henceforth: B^1, top left) on English CCGbank dependencies (LF1/UF1, Section 22, all sentences). The best overall model uses B^3, punctuation and lexicalization. The best model with complex arguments uses only B^1.

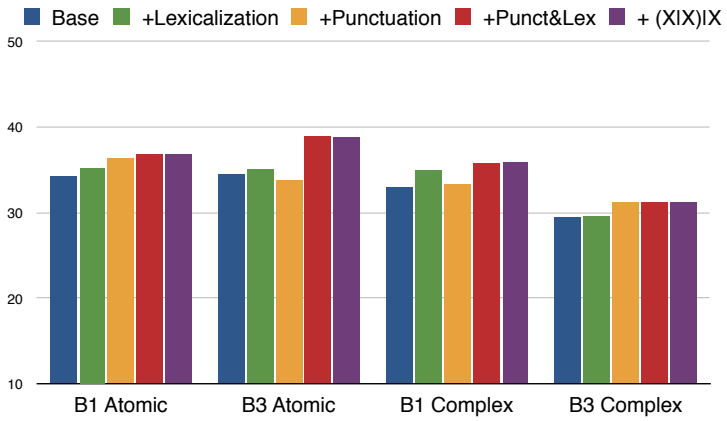


Figure 7.2: Labeled F1 performance for our model with and without complex arguments, and our discussed enhancements for B^1 and B^3. Full results are in Table 7.5.

Table 7.5 and Figure 7.2 show the performance of 20 different model settings on Section 22 under the simplified labeled CCG-based dependency evaluation proposed in Section 3.4.2 (and undirected unlabeled evaluation), starting with our original model (henceforth: $\mathbf{B_1}$, top left) which outperformed the PCFG model on all languages (Table 7.1). We are now using our more informative labeled evaluation to evaluate the effect of adding three model extensions: increasing grammatical complexity (Section 6.4.1), punctuation (Section 6.4.2), and lexicalization (Section 6.4.3).

We see that modeling punctuation and lexicalization both increase performance. We also show, as noted previously, that removing the induction restriction on $(\mathsf{X}/\mathsf{X})\backslash\mathsf{X}$ does not lead to a noticeable decrease in performance. We also see that an increase in grammatical and lexical complexity is only beneficial for the grammars that allow only atomic arguments, and only if both lexicalization and punctuation are modeled. Allowing complex arguments is generally not beneficial, and performance drops further if the grammatical complexity is increased to B^3. Future work might try to allow these categories in the grammar but discourage them in the prior, something we did not explore here. Our further analysis will focus on the three bolded models, $\mathbf{B_1}$, $\mathbf{B_1^C}$ (the best model with complex arguments) and $\mathbf{B_3^{P\&L}}$ (the best overall model), whose supertag accuracy, labeled (LF1) and unlabeled undirected CCG dependency recovery on Section 23 are shown in Table 7.6. We see that $\mathbf{B_1^C}$ and $\mathbf{B_3^{P\&L}}$ both outperform $\mathbf{B_1}$ on all metrics, although the unlabeled metric (UF1) perhaps misleadingly suggests that $\mathbf{B_1^C}$ leads to a

Model	Supertagging	LF1	UF1
$\mathbf{B_1}$	59.2	34.5	60.6
$\mathbf{B_1^C}$	59.9	34.9	63.6
$\mathbf{B_3^{P\&L}}$	62.3	37.1	64.9

Table 7.6: Test set performance of the final systems discussed in this Chapter (Section 23)

greater improvement than the supertagging and LF1 metrics indicate.

Finally, to compare our models directly to a comparable unsupervised dependency parser [37], we evaluate using the unlabeled dependencies produced by Yamada and Matsumoto's [143] head rules for Sections 02-21 of the Penn Treebank (Table 7.7).[3] Naseem et al. [37] only report performance on sentences of up to length 20 (without punctuation marks) and train and test on the same data.

Comparing these numbers to labeled and unlabeled CCG dependencies on the same corpus (all sentences, hence, @∞), we see that performance increases on CCGbank (the successive rows in the table) do not translate to similar gains on these unlabeled dependencies.

While we have done our best to convert the predicate argument structure of CCG into dependencies, there are many constructions which have vastly different analyses and assumptions. This becomes obvious when naively attempting to compare undirected attachments between CCGbank and those obtained via Matsumoto's head-finding rules, the two gold treebanks would only obtain an F-score of 81.9%.

Effect of the Normal Form

Finally, our work has assumed parsing with a normal form: Eisner [88] when using limited composition and without type-raising, or Hockenmaier and Bisk [89] for the new extensions presented here. As normal forms constrain ambiguity by eliminating redundant semantic analyses, they should have an important impact on both the size of the search space and the performance of the model.

In Table 7.8, we evaluate the effect of the normal-form parsing algorithm

[3]One may recall that in our previous comparison we use hyperparameter schemes and report 64.2@20.

Model	CCGbank 02-21 LF1	UF1	WSJ2-21 DA @10	@20	@∞
Naseem (Universal)			71.9	50.4	
Naseem (English)			73.8	66.1	
$\mathbf{B_1}$	33.8	60.3	70.7	63.1	58.4
$\mathbf{B_1^C}$	34.4	62.0	70.5	65.4	61.9
$\mathbf{B_3^{P\&L}}$	38.3	66.2	71.3	65.9	62.3

Table 7.7: To perform a valid comparison to Naseem (2010) [37] we train and test on the same data (Sections 02-21). Our goal is to compare performance on CCGbank dependencies @∞ (left side) and CoNLL-style directed attachments (right side).

	NF?	Performance ST	LF1	UF1	# of Parses Mean	Median
$\mathbf{B_1}$	No		28.8	53.6	2.1E73	5.1E13
	Yes		**34.2**	**60.2**	3.7E71	2.1E13
$\mathbf{B_3^{P\&L}}$	No	57.9	33.1	58.8	8.4E82	3.1E15
	Yes	**63.1**	**38.8**	**65.7**	5.6E79	6.6E14
$\mathbf{B_1^C}$	No	57.9	33.9	62.8	8.0E80	1.6E15
	Yes	**59.3**	**35.8**	**63.5**	2.1E79	6.7E14

Table 7.8: We also evaluate the same three models without the normal form. Normal-form parsing (NF) leads to significantly better performance and fewer parses on section 22.

on B^1 and our best model ($\mathbf{B_3^{P\&L}}$). We see that normal-form parsing is fundamental to performance, and decreases the average number of parses by up to three orders of magnitude. The importance of constraining the grammar might be alleviated if it was possible to evaluate the 1-best dependency structure (which would require computing marginal probabilities over all distinct dependency structures in the parse forest), rather than the 1-best derivation as we are doing here. For this reason it is difficult to know if the lack of a normal form disadvantages training, testing or both. It is possible the model learns much of the correct structure, but the prediction mass gets fragmented during evaluation. This is an open question, as we cannot recover this information from the model's distributions, but the utility of the normal form is clear.

7.2.2 English CCGbank Analysis

By using our CCGbank simplification (section 3.4.2), we can perform a more detailed analysis on the results we have just presented. While the performance of the models with lexicalization and punctuation clearly improved, the question remains as to which constructions did improve and which are still being lost. We first perform an error analysis based on supertag accuracy to look for missing categories from our complex model. Recall that supertag accuracy (Section 3.4) looks at whether a word is given the correct CCG category and upper-bounds labeled dependency evaluation. Finding many categories conspicuously absent, we try to isolate where the gains are coming from in a dependency evaluation, and finally perform a corpus analysis to identify categories missing entirely from our search space.

Supertagging error analysis

We first consider the lexical categories that are induced by the models. Table 7.9 shows the accuracy with which they recover the most common gold lexical categories, together with the category that they most often produced instead. We see that the simplest model ($\mathbf{B_1}$) performs best on N, and perhaps overgenerates $(\mathsf{N}\backslash\mathsf{N})/\mathsf{N}$ (noun-modifying prepositions) while the overall best model ($\mathbf{B_3^{P\&L}}$) outperforms both other models only on intransitive verbs.

The most interesting component of our analysis is the long tail of constructions that must be captured in order to produce semantically appropriate representations. We can inspect the confusion matrix of the lexical categories that the model fails to use to obtain insight into how its predictions disagree with the ground truth, and why these constructions may require special attention when developing models in the future or augmenting the input. Table 7.10 shows the most common CCGbank categories that were in the search space of some of the more complex models (e.g. $\mathbf{B_3^C}$) but were never used by any of the parsers in a Viterbi parse. These include possessives, relative pronouns, modals/auxiliaries, control verbs and ditransitive. We show the categories that the $\mathbf{B_1^C}$ model uses instead. The gold categories shown correspond to the bold words in Table 7.10.

We can now easily analyze some of the simple mistakes the model has made. Row one in Table 7.10 shows the model confusing the headedness of possessives. Row two requires the model learn that *Before*, or prepositions,

Correct Category	B_1 LR	B_1 Used instead (%)		B_1^C LR	B_1^C Used instead (%)		$B_3^{P\&L}$ LR	$B_3^{P\&L}$ Used instead (%)	
N	**82.6**	N/N	7.5	74.5	N/N	8.3	77.4	N/N	9.8
N/N	78.5	(S\S)\(S\S)	9.8	71.9	(S\S)\(S\S)	8.7	**80.6**	N	7.7
S\N	17.3	S\S	43.5	**22.1**	S\S	27.6	18.3	S\S	39.5
S\S	38.1	N	24.3	34.9	N	16.0	**39.4**	N	22.7
S/S	37.8	N\N	20.8	41.1	N/N	16.3	**57.2**	(S\S)/S	13.8
(N\N)/N	**64.3**	(S\S)/N	20.8	60.5	(S\S)/N	13.8	53.1	(S\S)/N	23.8
(S\N)/N	25.6	S/N	27.0	26.0	(S/N)/N	23.5	**29.4**	S/N	22.3
(S\S)/N	51.0	(N\N)/N	23.1	48.0	(N\N)/N	18.2	**62.6**	N/N	10.1
(S\N)/S	**60.7**	S\N	12.1	55.7	S\N	12.4	57.9	S\N	11.0
(S\S)/S	38.0	(N\N)/N	35.2	50.8	S/S	14.4	**61.5**	N	7.5

Table 7.9: Detailed supertagging analysis: Labeled Recall (LR) scores of B^1, B_1^C, and $B_3^{P\&L}$ on the most common recovered (simplified) lexical categories in Section 22, along with the most commonly produced error.

Category	**Example usage**	Used instead by $\mathbf{B_1^C}$ (%)					
(N/N)\N	The woman **'s** company ...	(N\N)/N	89.9	N/N	3.7	N	2.9
(S/S)/N	**Before** Monday, ...	S/S	69.9	N/N	14.8	(N\N)/N	8.2
(N/N)/(N/N)	The **very** tall man ...	N/N	38.0	(S\S)\(S\S)	33.9	(S\S)/N	10.1
(N\N)/(S\N)	John, **who** ran home, ...	(S\S)/(S/N)	26.5	N\N	23.3	S/S	14.9
(S\N)/(S\N)	I **promise to** pay ...	S\N	32.6	(S\S)/(S/N)	21.5	(S\N)/(S/N)	12.4
((S\N)/N)/N	I **gave** her a gift.	(S\N)/N	34.6	(S/N)/N	34.6	N/N	7.7
((S\N)/(S\N))/N	I **persuaded** her to pay ...	(S\N)/N	24.8	(S/N)/N	22.0	N/N	11.0

Table 7.10: Categories that are in the search space of the induction algorithm, but do not occur in any Viterbi parse of any sentence, and what $\mathbf{B_1^C}$ uses instead. Example sentences are shown for demonstration but do not represent the specific contexts in which the incorrect categories are used.

can transform an introductory clause into a sentence modifier. This is particularly tricky in news text which has many sentences and clauses linked by conjunctions or punctuation.

Row three contains two common mistakes. The system is tasked with producing a modifier of noun modifiers ((N/N)/(N/N)). The most common mistake is easy to understand as the system produces a simple adjective category (N/N) for both *very* and *tall* in the sentence *very tall man*. This simple set of modifiers would be the correct analysis for the very similar sentence *big green ball*. The second most common mistake uses the category (S\S)\(S\S), which could be used to modify a VP or sentential modifier (S\S). An example context for this mistake is the phrase: *estimated reserves of 32 million barrels*. Here, *32* should modify *million* which in turn modifies *barrels*. Unfortunately, numbers come very often after verb attaching prepositions in the corpus, and so the model has discovered that it can use what should be a rare double verb modifier to compose into the preposition.

The most interesting and difficult type of error is that of recovering non-local dependencies (Section 7.2.3). The recovery of non-local dependencies requires the use of lexical categories with complex arguments and coindexation. This makes their recovery beyond the scope of both standard dependency-based approaches and our original induction algorithm. But the parser does not learn to use lexical categories with complex arguments correctly even when the algorithm is extended to induce them. For example, $\mathbf{B}_1^{\mathbf{C}}$ prefers to treat auxiliaries or equi verbs like *promise* as intransitives rather than as an auxiliary that shares its subject with *pay*. The surface string supports this decision, as it can be parsed without having to capture the non-local dependencies (top row) present in the correct (bottom row) analysis:

I	promise	to	pay	you
N	S\N	(S\S)/S	S/N	N
N	(S\N)/(S\N)	(S\N)/(S\N)	(S\N)/N	N

As this is a particularly interesting problem that we are able uncover with our approach and is problematic for progress on grammar induction, we provide several examples from the development set of places where we predict the wrong category.

	... earnings,	which	have	marched	steadily ...
Gold		$(N\backslash N)/(S\backslash N)$	$(S\backslash N)/(S\backslash N)$	$S\backslash N$	
Predicted		S/S	$S\backslash N$	$S\backslash S$	

	... million,	which	have	helped	launch ...
Gold		$(N\backslash N)/(S\backslash N)$	$(S\backslash N)/(S\backslash N)$	$(S\backslash N)/N$	
Predicted		$(S\backslash S)/(S/N)$	S/S	$(S/N)/N$	

	... write-off	could	help	solidify	...
Gold		$(S\backslash N)/(S\backslash N)$	$(S\backslash N)/(S\backslash N)$	$(S\backslash N)/N$	
Predicted		$(S\backslash N)/(S/N)$	$(S/N)/(S/N)$	$(S/N)/N$	

Learning strong lexical statistics for the main verbs might help, or it might be the case that to address these might require actually modeling the non-local dependencies.

We also see that this model uses seemingly non-English verb categories of the form $(S/N)/N$, both for ditransitives, and object control verbs. Perhaps it chooses this analysis for object control verbs because the spurious $/N$ argument can be swallowed by other categories that take arguments of the form S/N, like its (incorrect) treatment of subject relative pronouns (row 4 of Table 7.10), or because of the pervasive use of auxiliaries and modals which separate the verb from its subject:

	... bill that	would	give	the secretary	authority
Gold		$(S\backslash N)/(S\backslash N)$	$((S\backslash N)/N)/N$		
Predicted		$(S\backslash S)/(S/N)$	$(S/N)/N$		

One possible lesson we can extract from this is that practical approaches for building parsers for new languages might need to focus on injecting semantic information that is outside the scope of our learner from text alone.

Dependency error analysis

Table 7.11 shows the labeled recall of the most common dependencies. We see that both new models typically outperform the baseline, although they yield different improvements on different dependency types. $\mathbf{B}_1^{\mathbf{C}}$ is better at

	1st Argument			2nd Argument		
	$\mathbf{B_1}$	$\mathbf{B_1^C}$	$\mathbf{B_3^{P\&L}}$	$\mathbf{B_1}$	$\mathbf{B_1^C}$	$\mathbf{B_3^{P\&L}}$
N/N	68.4	69.7	**71.6**			
S\N	12.2	**24.9**	14.6			
S\S	17.0	16.2	**18.7**			
S/S	24.0	27.1	**33.8**			
(N\N)/N	49.7	**54.4**	51.2	41.0	**46.2**	42.4
(S\N)/N	26.6	32.9	**34.4**	30.6	33.2	**33.8**
(S\S)/N	21.6	19.2	**24.7**	24.0	24.9	**29.3**
(S\N)/S	23.9	**50.3**	32.5	25.2	**59.1**	35.0
(S\S)/S	6.1	**22.7**	14.1	9.5	**34.6**	19.5

Table 7.11: LF1 scores of B^1, $\mathsf{B}_1^{\mathsf{C}}$ and $\mathbf{B_3^{P\&L}}$ on the most common dependency types in Section 22.

recovering the subjects of intransitive verbs (S\N) and verbs that take sentential complements ((S\N)/S) while $\mathbf{B_3}$ is better for simple adjuncts (N/N, S/S, S\S) and transitive verbs.

7.2.3 Dealing with Non-Local Dependencies

While the methodology used here is restricted to CCG-based algorithms, we believe the lessons to be very general. The aforementioned constructions involve optional arguments, non-local dependencies, and multiple potential heads. Even though CCG is theoretically expressive enough to handle these constructions, they present the unsupervised learner with additional ambiguity that will pose difficulties independently of the underlying grammatical representation.

For example, although our approach learns that subject NPs are taken as arguments by verbs, the task of deciding which verb to attach the subject to is frequently ambiguous. This most commonly occurs in verb chains, and is compounded in the presence of subject-modifying relative clauses (in CCGbank, both constructions are in fact treated as several verbs sharing a single subject). To illustrate this, we ran the $\mathsf{B}_1^{\mathsf{C}}$ and $\mathbf{B_3^{P\&L}}$ systems on the following three sentences:

1. The woman **won** an award
2. The woman **has won** an award
3. The woman **being promoted has won** an award

The single-verb sentence is correctly parsed by both models, but they flounder as distractors are added. Both treat *has* as an intransitive verb, *won* is given a category traditionally used for adverbs and *an* ends up with a low probability preposition category:

	The	woman	**won**	an	award
$B_3^{\mathbf{P\&L}}/B_1^{\mathbf{C}}$:	N/N	N	(S\N)/N	N/N	N

	The	woman	**has**	**won**	an	award
$B_3^{\mathbf{P\&L}}/B_1^{\mathbf{C}}$:	N/N	N	S\N	S\S	(S\S)/N	N

It appears that despite the adverb analysis for *won* and the preposition for *an* having low probabilities, they are still higher under these models than analyses using complex arguments. Future work might try and investigate enforcing additional consistency of analyses across sentences.

To accommodate the presence of two additional verbs, both models analyze *being* as a noun modifier that takes *promoted* as an argument. $B_1^{\mathbf{C}}$ (correctly) stipulates a non-local dependency involving *promoted*, but treats it (arguably incorrectly) as a case of object extraction:

...	**being**	**promoted**	**has**	**won**	an	award
$B_3^{\mathbf{P\&L}}$	(N\N)/S	S	S\N	S\S	(S\S)/N	N
$B_1^{\mathbf{C}}$	(N\N)/(S/N)	S/N	S\N	S\S	(S\S)/N	N

Given that there is enough signal for our model to try and capture non-local dependencies, we would expect these grammatical constructions to pose even greater learning difficulties for dependency formalisms which do not have non-local dependencies in their search space.

Discovering these, and many of the other systematic errors describe here, may be less obvious when analyzing unlabeled dependency trees. But we would expect similar difficulties for any unsupervised approach when sentence complexity grows without a specific bias for a given analysis.

7.2.4 Wh-words and the Long Tail

To dig slightly deeper into the set of missing constructions, we tried to identify the most common categories that are beyond the search space of the current induction algorithm. To do this we needed to compute the set of frequent categories in the treebank. We do so by using CCGbank to compute the set of categories assigned to each part of speech tag. We then take

the counts from the corpus to compute what percentage of the corpus uses each (category, tag) assignment. We threshold the lexicon to only contain the categories that comprise 95% of token occurrences for each tag. Finally, to discover what categories lie outside the search space of our approach, we removed both the categories that contain PP, as we assume our approach does not have access to a labeled set of preposition, and those categories that our algorithm can induce with complex arguments and three rounds of induction. What remains are the categories shown in Table 7.12. These are constructions that motivate the need to improve our induction algorithm. The tags that are missing categories are predominantly wh-words required for wh-questions, relative clauses or free relative clauses. Some of these categories violate the assumptions made by the induction algorithm: question words return a sentence (S) but are not themselves verbs. This violates our seed knowledge assumption that only allows verbs to have the category S. Another example is that free relative pronouns return a noun, but take arguments (violating constraint 1 in Section 4.2.2). However, this is a surprisingly small set of special function words and, therefore, perhaps a strategic place for supervision. Questions, in particular, pose an interesting learning question – how does one learn that these constructions indicate missing information which only becomes available later in the discourse?

Additional Category	$p(cat \mid tag)$		Explanation (example)
((N\N)/(S\N))/N	.93	WP$	Possessive Wh-pronoun (whose)
N/(S/N)	.14	WP	Wh-pronoun (what)
N/(S\N)	.08	WP	Wh-pronoun (what)
((N\N)/S)\((N\N)/N)	.07	WDT	Wh-determiner (which)
((S\S)\(S\S))\N	.04	RBR	Adverb (earlier)
S/(S\N)	.04	WP	Wh-pronoun (whoever)
S/(S/N)	.02	WP	Wh-pronoun (whom)

Table 7.12: Common categories that the algorithm cannot induce.

7.2.5 Lessons Learned from Labeled Analysis

We introduced labeled evaluation metrics for unsupervised CCG parsers and showed that these expose many common syntactic phenomena that are currently out of scope for any unsupervised grammar induction system. We focused our analysis on English for simplicity, but many of the same types of

problems exist in other languages, and can be easily identified as stemming from the same lack of supervision. For example, in Japanese we would expect problems with post-positions, in German with verb clusters, in Chinese with measure words, or in Arabic with morphology and variable word order.

We believe that one way to overcome the issues we have identified is to incorporate a semantic signal. Lexical semantics, if sparsity can be avoided, might suffice; otherwise learning with grounding [144] or an extrinsic task could be used to bias the choice of predicates, their arity and in turn the function words that connect them. Alternatively, a simpler solution might be to follow the lead of Boonkwan and Steedman [38] or Garrette et al. [103] where gold categories are assigned by a linguist or treebank to tags and words. It is possible that more limited syntactic supervision might be sufficient if focused on the semantically ambiguous cases we have isolated.

More generally, we hope to initiate a conversation about grammar induction which includes a discussion of how these non-trivial constructions can be discovered, learned, and modeled. Relatedly, in future extensions to semi-supervised or projection-based approaches, these types of constructions are probably the most useful to get right despite comprising the tail, as analyses without them may not be semantically appropriate. Further, because the semantics of a sentence is so heavily dependent on many of these constructions grounding or active learning may be ideal mechanisms for learning these categories. In summary, we hope to begin to pull back the veil on the types of information that a truly unsupervised system, if one should ever exist, would need to learn, and we pose a challenge to the community to find ways that a learner might discover this knowledge without hand-engineering it.

Chapter 8

Grammar Induction with Automatically Induced Clusters

As mentioned, one goal of our work is to lessen the reliance of the grammar induction literature on gold POS-tagged text. We show here, for the first time, that very limited human supervision may be enough to induce *labeled* dependencies from automatically induced word clusters. Thus far we have assumed access to POS tags and defined our seed knowledge by attaching S to verbs and N to nouns. However, assuming gold POS tags is highly unrealistic for most scenarios in which one would wish to use an otherwise unsupervised parser.

In joint work with Christos Christodoulopoulos [145], we demonstrate how our universal seed knowledge can be easily applied to induced clusters given a small number of words labeled as *noun*, *verb* or *other*, and that this small amount of knowledge is sufficient to produce labeled syntactic structures from raw text. Specifically, we provide a labeled evaluation of induced CCG parsers against the English [70] and Chinese [93] CCGbanks. To provide a direct comparison to the dependency induction literature, we also provide an unlabeled evaluation on the 10 dependency corpora that were used for the task of grammar induction from raw text in the PASCAL Challenge on Grammar Induction [63].

The system of Christodoulopoulos et al. [146] was the only participant competing in the PASCAL Challenge that operated over raw text (instead of gold POS tags). However, their approach did not outperform the six baseline systems provided. These baselines were two versions of the DMV model [41, 137] run on varying numbers of induced Brown clusters (described in section 8.1). We will, therefore, use these baselines in our evaluation.

Work in this chapter was first published in Y. Bisk, C. Christodoulopoulos, and J. Hockenmaier, "Labeled grammar induction with minimal supervision," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Beijing,China, July 2015. [145] and is reprinted here with permission by the copyright holder.

Outside of the shared task, Spitkovsky et al. [147] demonstrated impressive performance using Brown clusters but did not provide evaluation for languages other than English.

The system we propose here will use a coarse-grained labeling comprised of three classes, which makes it substantially simpler than traditional tagsets, and uses far fewer labeled tokens than is customary for weakly-supervised approaches [107, 103].

The parsing model is our new punctuation and lexicalization aware HDP-CCG ($\mathsf{B}_{\mathbf{3}}^{\mathbf{P\&L}}$ defined in Section 6.4 and evaluated in Section 7.2).

8.1 Inducing Word Clusters

We will evaluate three clustering approaches, briefly summarized here:

Brown Clusters: Brown clusters [148] assign each word to a single cluster using an agglomerative clustering that maximizes the probability of the corpus under a bi-gram class conditional model:

$$P(w_i|w_{i-1}) = P(w_i|c_i) \times P(c_i|c_{i-1})$$

Brown clustering is a greedy algorithm that defines n classes and then moves through the corpus assigning words to whichever cluster maximizes the probability of the corpus. We use the implementation at `https://github.com/percyliang/brown-cluster` for our experiments. Because the clustering is agglomerative, a cluster hierarchy is formed where classes further down the tree correspond to smaller word groupings with finer grained distinctions.

In what is perhaps unfair to this technique, we simply ran Brown clustering with the desired number of clusters. We did not attempt to form a fine-grained clustering which we could then prune the desired size. We avoided this potential optimization simply to avoid adding another parameter to the approach that required tuning.

BMMM: The Bayesian Multinomial Mixture Model (BMMM, [149]) is also a hard clustering system, but has the ability to incorporate multiple types of features either at a token level (e.g. ± 1 context word) or at a type level (e.g. morphology features derived from the Morfessor system [150]). The combination of these features allows BMMM to better capture morphosyntactic information.

The BMMM clusters word types by clustering their features. Every latent syntactic class has some prior probability and some distribution over the features it extracted from the data. Inference then searches for model probabilities that maximize the observed features under a set of latent classes. We use the implementation at `https://github.com/christos-c/bmmm`.

Bi-gram HMM: We also evaluate unsupervised bi-gram HMMs, since the soft clustering they provide may be advantageous over the hard Brown and BMMM clusters. A bi-gram HMM has the same probability model for a word as the Brown clusters except that instead of maximizing this conditional distribution by greedy assignment of word types to cluster we aim to maximize the probability of the complete sequences (sentences) in our data. This is accomplished via the Forward-Backward algorithm [151] which computes posterior marginals for sequence data via message passing forward and backward through the lattice of possible clusters for any given token in the corpus. These marginals can then be used to re-estimate the data and are optimized via the Expectation-Maximization algorithm.

The important conceptual point is that distributions over the lexical emissions and class transitions keep their full support throughout inference. This, combined with optimizing complete sequence likelihood, means that during Viterbi decoding at test time each individual token is assigned a cluster, rather than each lexical type, and that the choice is informed by both the preceding and following words. This often leads to choosing a cluster assignment which assigns a low probability to any specific token in the sequence but a higher overall score to the full sentence. Despite this flexibility in the model, unsupervised HMMs may not find good POS tags [152], and in future work, more sophisticated models (e.g. [153]), might outperform the systems we use here.

In all cases, we assume that we can identify punctuation marks, which are moved to their own cluster and ignored for the purposes of tagging and parsing evaluation.

8.1.1 Identifying Noun and Verb Clusters

To induce CCGs from induced clusters, we need to attach our seed knowledge (Section 4.1) to each cluster. This requires attaching a single label (*noun*, *verb*, or *other*) to each cluster. We did not investigate using a softer labeling

that allows for assigning a cluster to two or all three of our seed classes. Because this knowledge forms the basis of our grammar, the labeling must be done judiciously. For example, the model performs very poorly when every class is given the label *verb*. If allowed to, the system will choose to analyze prepositions as the main sentential predicates instead of verbs.

We demonstrate here that labeling three frequent words per cluster is sufficient to outperform state-of-the-art performance on grammar induction from raw text in many languages.[1] We emulate having a native speaker annotate words for us by using the universal tagset [2] as our source of labels for the most frequent three words per cluster (we map the tags NOUN, NUM, PRON to *noun*, VERB to *verb*, and all others to *other*). The final labeling is a majority vote, where each word type contributes a vote for each label it can take. One possible extension of this approach would be to use a plurality vote with where each vote was weighted by the type's frequency rather than the equal weighting we used here.

This approach can be easily scaled to allow more words per cluster to vote. But we will see that three per cluster is sufficient to label most tokens correctly. Future work may be better spent improving the underlying clustering instead of adding additional human annotation.

8.2 Experimental Setup

We focus first on producing CCG labeled predicate-argument dependencies for English and Chinese, and will then apply our best settings to produce a comparison with the tree structures of the languages of the PASCAL Shared Task. All languages will be trained on sentences of up to 20 words (not counting punctuation). All cluster induction algorithms are treated as black boxes and run over the complete data sets in advance. This alleviates having to handle tagging of unknown words.

To provide an intuition for the performance of the induced word clusters, we provide two standard metrics for unsupervised tagging:

Many-to-one (M-1): A commonly used measure that relies on mapping each cluster to the most common POS tag of its words. Among the tokens tagged with a given cluster, whichever gold tag is most common overall is

[1] Though their labels were much more informative, the basic idea is similar to that of Haghighi and Klein (2006) [154]

assigned to the cluster as its label. More than one cluster may be mapped to a give POS tag. M-1 reports the percentage of correctly labeled tokens if the tags were then propagated back to the clusters that chose them. However, M-1 can be easily inflated by simply inducing more clusters. In the extreme, one could split the clusters until each cluster only corresponded to a single POS tag.

V-Measure: Proposed by [155], V-Measure (VM) measures the information-theoretic distance between two clusterings and has been shown to be robust to the number of induced clusters. Both of these metrics are known [156] to be highly dependent on the gold annotation standards they are compared against, and may not correlate with downstream performance at parsing [157].

Of more immediate relevance to our task is the ability to accurately identify nouns and verbs.

Noun, Verb, and Other Recall: We measure the (token-based) recall of our three-way labeling scheme of clusters as *noun*/*verb*/*other* against the universal POS tags of each token. As noted above, we assume *noun* corresponds to the tags NUM, PRON, and NOUN. We assume only VERB corresponds to *verb*, and that all other tags map to *other*.

8.3 Experiment 1: CCG-based Evaluation

8.3.1 Experimental Setup

For our primary experiments, we train and test our systems on the English and Chinese CCGbanks, and, as with our previous work, report labeled F1 (LF1) and undirected unlabeled F1 (UF1) over CCG dependencies. For the labeled evaluation, we use our simplification of CCGbank. For Chinese we also map both M and QP to N.

We use the published train/development/test splits, using the development set for choosing a cluster induction algorithm, and then will present final performance on the test data. We induce 36 tags for English and 37 for Chinese to match the number of non-symbol and punctuation tags present in the treebanks.

		Tagging		Labeling			Parsing	
		M-1	VM	N	V	O	LF1	Gold
English	Brown	62.4	56.3	**85.6**	59.4	81.2	23.3	
	BMMM	**66.8**	**58.7**	81.0	**81.2**	**82.7**	**26.6**	38.8
	HMM	51.1	41.7	76.3	63.3	**82.6**	25.8	
Chinese	Brown	**66.0**	**50.1**	88.9	28.6	**91.3**	10.2	
	BMMM	64.8	50.0	**94.4**	**48.7**	87.0	**10.5**	16.6
	HMM	46.3	30.8	68.0	44.6	76.7	3.13	

Table 8.1: Tagging evaluation (M-1, VM, N/V/O Recall) and labeled CCG-Dependency performance (LF1) as compared to the use of Gold POS tags (Gold) for the three clustering algorithms.

8.3.2 Results

Table 8.1 presents the parsing and tagging development results on the two CCG corpora. In terms of tagging performance, we can see that the two hard clustering systems significantly outperform the HMM, but the relative performance of Brown and BMMM is mixed.

More importantly, we see that, at least for English, despite clear differences in tagging performance, the parsing results of all models (LF1) are much more similar. In Chinese we see that the performance of the two hard clustering systems is almost identical, again, not representative of the differences in the tagging scores. The N/V/O recall scores in both languages are equally poor predictors of parsing performance. However, these scores show that having only three labeled tokens per class is sufficient to capture most of the necessary distinctions for the HDP-CCG. All of this confirms the observations of Headden et al. [157] that POS tagging metrics are not correlated with parsing performance. However, since BMMM seems to have a slight overall advantage, we will be using it as our clustering system for the remaining experiments.

Since the goal of this work is to produce labeled syntactic structures, we also want to evaluate our performance against that of the HDP-CCG system that uses gold-standard POS tags. As we can see in the last two columns of our development results in Table 8.1 and in the final test results of Table 8.2, our system is within 2/3 of the labeled performance of the gold-POS-based HDP-CCG.

Figure 8.1 shows an example labeled syntactic structure induced by the

	LF1/UF1	Gold
English	26.0 / 51.1	37.1 / 64.9
Chinese	10.3 / 33.5	15.6 / 39.8

Table 8.2: Comparison on the test sets of our CCG parsing performance to using gold tags.

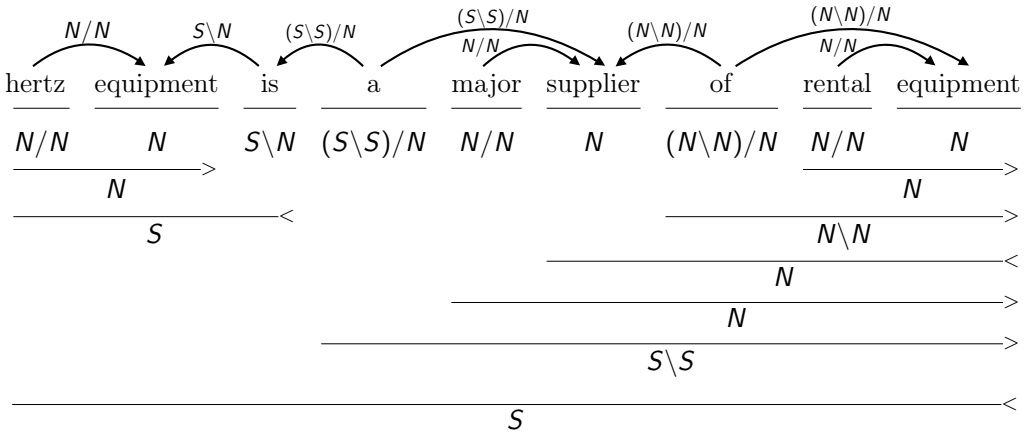


Figure 8.1: A sample derivation from the WSJ Section 22 demonstrating the system is learning most of the correct categories of CCGbank but has incorrectly analyzed the determiner as a preposition.

model. We can see the system successfully learns to attach the final prepositional phrase, but mistakes the verb for intransitive. The labeled and unlabeled undirected recall for this parse are 5/8 and 7/8 respectively.

8.4 Experiment 2: PASCAL Shared Task

8.4.1 Experimental Setup

During the PASCAL shared task, participants were encouraged to train over the complete union of the data splits. We do the same here, use the development set for choosing a HDP-CCG hyperparameter, and then present final results for comparison on the test section. We vary the hyperparameter for this evaluation because the data sets fluctuate dramatically in size from 9K to 700K tokens on sentences up to length 20. Rather than match all of the tagsets we simply induce 49 (50 if you include punctuation) classes for every language. The actual tagsets vary from 20 to 304 tags (median 39, mean 78) so we chose 49 as a round midpoint.

8.4.2 Results

We now present results for the 10 corpora of the PASCAL shared task (evaluated on all sentence lengths). Table 8.3 presents the test performance for each language with the best hyperparameter chosen from the set $\{100, 1000, 2500\}$. As before, we convert all of our parses (Section 3.2.4) to match the CoNLL style dependencies. This includes transformation like having the modifiers depend on heads, unlike in CCG. Also, as with our previous results for the PASCAL data (Section 7.1.1), the evaluation reported is for unlabeled attachments.

The languages are sorted by the number of non-punctuation tokens in sentences of up to length 20. Despite our average performance (37.8) being higher than the shared task (31.8), the variance in gains and losses are substantial ($\sigma = 15.2$). It appears evident from the results that while data sparsity may play a role in affecting performance, the more linguistically interesting thread appears to be morphology. Czech is perhaps a prime example as it has twice the data of the next largest language (700K tokens vs 336K in English), but our approach still performs poorly. It is possible that

	VM	N / V / O	This	ST	@15	BH
Czech_{2500}	42	86 / 67 / 67	28.3	**33.2**	32.4	50.7
English_{2500}	59	87 / 76 / 85	**43.8**	24.4	51.6	62.9
CHILDES_{2500}	68	84 / 97 / 89	**47.2**	42.2	47.5	73.3
Portuguese_{2500}	55	88 / 81 / 69	**55.5**	31.7	55.8	70.5
Dutch_{1000}	50	81 / 81 / 82	**39.9**	33.7	43.8	54.4
Basque_{1000}	52	2 / 78 / 95	**31.1**	28.7	35.2	45.0
Swedish_{1000}	50	89 / 74 / 85	**45.8**	28.2	52.9	66.9
Slovene_{1000}	50	83 / 75 / 79	18.5	**19.2**	23.6	46.4
Danish_{100}	59	95 / 79 / 82	**33.9**	31.9	37.7	58.5
Arabic_{100}	51	85 / 76 / 90	34.5	**44.4**	43.7	65.1
Average	54	78 / 78 / 82	**37.8**	31.8	42.4	59.4

Table 8.3: Tagging VM and N/V/O Recall alongside Directed Attachment for our approach and the best shared task baseline. Subscripts below the language show the hyperparamter constant (section 6.3.2) used during training. Additionally, we provide results for length 15 to compare to our previously published results (Section 7.1.1).

this might be addressed, at least in part, by training the parser on words split by a morphological analyzer.

Additionally, it is clear that in some languages only very basic properties are being learned, as must be the case in Basque where noun recall was abysmal (2%). This is masked by the unlabeled dependency metric, because the model learns to treat nouns as adverbs. This creates a dependency arc from the verb to the noun, which is indistinguishable (according to the unlabeled CoNLL-style metric) from the arc that would be drawn between a predicate and its argument. This is precisely the type of information loss discussed in Section 3.4.1, though perhaps the fact that the grammar learns this behavior can be used as the basis for future work in tag re-estimation.

Finally, while we see that the hard clustering systems outperform the HMM for our experiments, this is perhaps best explained by analyzing the average number of gold fine-grained tags per lexical type in each of the corpora. We find, that the "difficult" languages have lower average number of tags per lexical type, surface form, (1.01 for Czech, 1.03 for Arabic) than English (1.17) which is the most ambiguous. This is likely due to morphology distinguishing otherwise ambiguous lemmas. Where words like *drink* are ambiguous in English, between noun and verb, in languages which conjugate the verb form with prefixes/suffixes it is far less likely to see the same surface string for a

verb as the bare noun.

8.5 Conclusions and Directions for Future Work

Based on our final PASCAL results, there are several languages where our performance greatly exceeds the previously published results, but many where we fall short. It also appears to be the case that this problem correlates with morphology (e.g. Ar, Sl, Eu, Cs) and some of the lowest performing intrinsic evaluations of the clustering and labeling (Cs and Eu).

In principle, the BMMM is taking morphological information into account, as it is provided with the automatically produced suffixes of Morfessor. Unfortunately, BMMM's treatment of them simply as features from a "black-box" appears to be too naive for our purposes. Properly modeling the relationship between prefixes, stems and suffixes both within the tag induction and parsing framework is likely necessary for a high performing system.

We have produced the first labeled syntactic structures from raw text. As there remains a noticeable performance gap due to the use of induced clusters, this lends credence to our claim that moving forward we may need to remove some of the pipeline enforced abstraction barriers between tagging and parsing to allow for the two tasks to benefit each other [146].

Chapter 9

Semantics With Induced Grammars

In Section 3.1.5 we discussed the clean, and transparent, relationship between CCG and semantics. A CCG syntactic derivation provides the structure for a semantic derivation. This relationship is commonly exploited in tasks like Semantic Parsing for Question Answering [82]. In this context, a natural language question is parsed into a database query, which when executed returns an answer to the original question. Most systems that use CCG for semantic tasks assume they have access to the mapping between semantic predicates and words [78, 79, 80, 81, 82, 83] (Section 3.1.5). In this formulation, the combinatory rules of CCG can then be used to build a semantic interpretation in tandem with the syntactic derivation the syntactic and semantic derivation are built together.

A novel insight exploited by Reddy et al. (2014) [72] is that the expense of creating a mapping between database predicates and text can be avoided by splitting up the semantic parsing process into two stages. First, syntactic CCG derivations can be used to create an ungrounded semantic representation (Section 9.2). Second, existing resources can be used to learn the grounding without explicit supervision (Section 9.3). The reason this approach works is because the correct CCG derivation correctly captures and represents the semantics of the sentence, allowing for the application-specific meanings to be learned separately and later.

Section 7.2.2 analyzed missing syntactic categories and their effects on producing syntactic derivations that can support the correct semantics (Section 7.2.3). In this chapter, we will perform a direct semantic evaluation of our approach. By using our parser, and a semi-supervised extension, to produce semantic analyses, we will be able to measure the strength of our approach in a downstream task, demonstrate the novel result that an unsupervised

Work in this chapter was performed jointly with Siva Reddy, John Blitzer, and Mark Steedman.

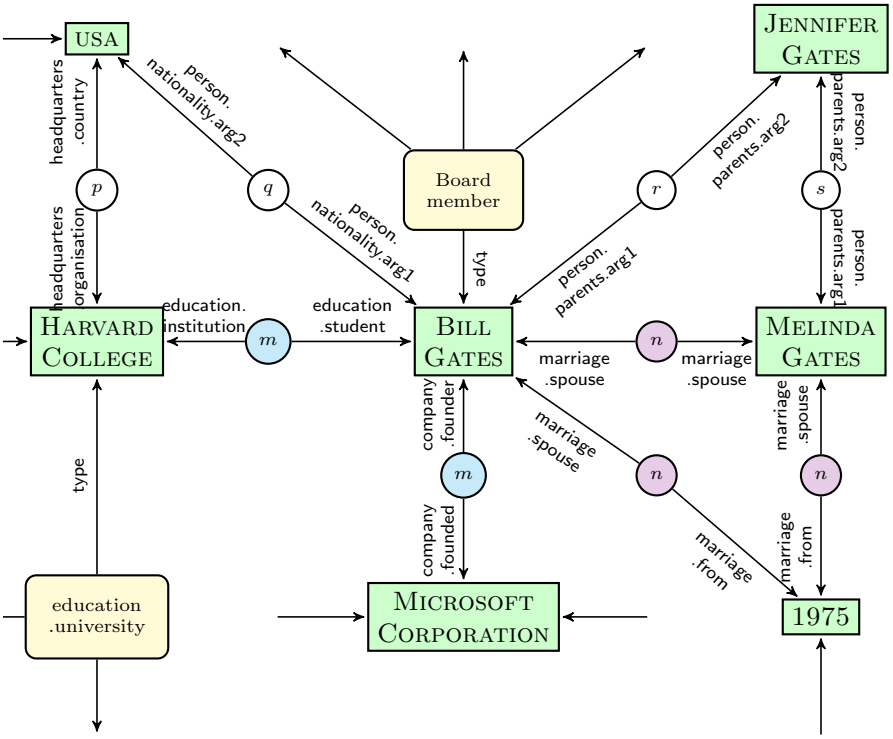


Figure 9.1: An example snippet from a knowledge graph centered on Bill Gates. Here Bill Gates is linked to other entities and types.

parser can create meaningful semantic representations, and test our claims about the utility of limited supervision. This is joint work with Siva Reddy, John Blitzer and Mark Steedman.

9.1 Database Semantics

Relational databases [158] represent structured data in tables whose rows and columns correspond to different aspects of this structure. The values of several such tables can be intersected or reasoned about using first-order logic. This innovation allowed for the introduction of query languages like the Structured Query Language (SQL) for asking questions of and manipulating data. As many databases exist in this form, parsing a natural language query into this format proves very useful for many applications.

Recently, very large data-stores of facts like Freebase [159, 160] or Wiki-Data[1] have been constructed as tuple-stores with graph architectures (Figure

[1] `https://www.wikidata.org/`

9.1). The architecture assumes that the nodes of the graph corresponds to entities or events, which are linked to one another via some relation. The data can, therefore, be represented by tuples of the form $(entity_1, relation, entity_2)$. In this way, facts, entities and relations are easily added to the graph without needing to define new tables or schemas. The strength of this approach becomes apparent when extracting facts from the web. Far more is known or written about celebrities and politicians from the United States, than other countries. The tuple store allows for every new fact (or revelation) to be quickly and easy added. Additionally, these resources are just as easily queried.

These facts and knowledge graphs will be the basis for the semantics in our approach.

9.2 Ungrounded Database Semantics

As discussed previously, every syntactic rule in CCG has a corresponding semantic counterpart that can be used to build a semantic interpretation for the sentence, given a CCG lexicon with semantic interpretations (section 3.1.5). These semantic predicates do not have to correspond to those in a particular database or knowledge graph, but can be arbitrary tokens: $predicate_1$, ... Such dummy predicates can then be used to produce (ungrounded) semantic representations. We can see this in the following derivation:

$$\dfrac{\dfrac{\textit{Google}}{\mathsf{N} : \texttt{Google}} \quad \dfrac{\dfrac{\textit{acquired}}{(\mathsf{S}\backslash\mathsf{N})/\mathsf{N} : \lambda y.\lambda x.acquired(x, y)} \quad \dfrac{\textit{Nest}}{\mathsf{N} : \texttt{Nest}}}{\mathsf{S}\backslash\mathsf{N} : \lambda x.acquired(x, \texttt{Nest})}>}{\mathsf{S} : acquired(\texttt{Google}, \texttt{Nest})}<$$

The "semantic" result of this derivation is: *acquired*(Google, Nest). If repeated across a corpus, a series of "facts" can be extracted from the data:

acquired(Google, Nest)	*acquired*(Microsoft, Skype)
founded(Bill_Gates, Microsoft)	*married*(Bill_Gates, Melinda_Gates)
...	

This semantic representation is ungrounded because we do not know if the predicates correspond to any relations in a database. What remains is to

String	Freebase ID
Google	/m/045c7b
Nest	/m/0hr65bq
Microsoft	/m/04sv4
Skype	/m/026wfg
Bill Gates	/m/017nt
Melinda Gates	m/0dmp4

Table 9.1: The corresponding Freebase node IDs for several people and companies in our data.

Predicate	Freebase Relation
acquired	/organization/organization/companies_acquired
founded	/organization/organization_founder/organizations_founded
married	/people/person/spouse_s

Table 9.2: The corresponding Freebase relation names for our example data.

attach both the entities in our examples to nodes in the knowledge graph and to find the correspondence, if any exist, between our dummy predicates (constructed from the surface string) and the edges that exist in Freebase.

9.3 Grounding Semantics

There are two steps to grounding our representation: 1. Grounding the names of entities to database IDs (Table 9.1) and 2. Finding the correct Freebase relation for every predicate (Table 9.2).

Both of these require ambiguities to be resolved. Microsoft, for, example might refer to Microsoft Corporation or Microsoft Research, both of which have different IDs. Similarly, Bill Gates's relation to Microsoft is as founder, board member, and employee. In this work, our focus is on evaluating the parses and their predicted semantics so we will only be concerned with grounding predicates and assume that entities have already been grounded. We achieve this by using an annotated version of ClueWeb (a large crawl of the internet). The annotated version: [161], which includes freebase entity IDs.

Reddy, Lapata, and Steedman (2014) To learn the mapping between predicates and Freebase relations, we use a system introduced by Reddy, Lapata and Steedman (2014) [72]. The basic intuition behind their approach is that a sentence can be parsed to an ungrounded semantic representation from which features are extracted to learn its correspondence to Freebase. Learning this mapping requires that they generate a large set of training examples of the form (ungrounded graph, known freebase equivalent). While they cannot automatically generate data of this form, they can use declarative sentences to produce synthetic queries. We will demonstrate this process for the following two sentences:

`Bill Gates` *founded* `Microsoft`
`Tony Fadell` *founded* `Nest Labs`

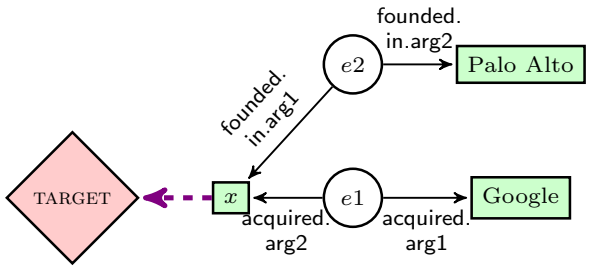
Both sentences are first parsed into simple ungrounded representations:

founded(`Bill Gates`, `Microsoft`)
founded(`Tony Fadell`, `Nest Labs`)

The ungrounded representation does not tell us about the underlying Freebase relation but it does indicate that the same predicate (*founded*) links two pairs of entities. Because the nodes those entities correspond to are known, we can choose to hide one, and ask the system to predict it. Doing so will produce two "queries" with known answers:

Query	**Answer**
$founded(x,$ `Microsoft`$)$	`Bill Gates`
$founded($`Tony Fadell`$, x)$	`Nest Labs`

The system can now extract a number of features (Section 3 of Reddy et al. [72]) from the query to use in a perceptron for predicting the missing entity. These features include type information about the entities and graph connectivity. Unlike Reddy et al., we do not use lexical similarity features. By treating the vocabulary of predicates and freebase relations as disjoint, we can test an approach which should generalize to new languages beyond English. This means that despite the string similarity between the English word "founded" and the Freebase relation name /organization/organization founder/organizations_founded there is no information about the surface string provided to the model. Reddy et al. report that



$\textsc{target}(x)$

$\wedge$ founded.in.arg1$(e2, x)$ $\wedge$ founded.in.arg2$(e2,$ Palo Alto$)$

$\wedge$ acquired.arg1$(e1,$ Google$)$ $\wedge$ acquired.arg2$(e1, x)$

`Google` *acquired* `Nest Labs`, *which was founded in* `Palo Alto`

Figure 9.2: By removing an entity (Nest Labs) from a declarative statement, the analyzed statement can be transformed into a query. The graph and logical expression above are the resultant queries that will be generated by a successful syntactic parse.

these similarity features we are ignoring account for between 1.5 and 3 points of accuracy for semantic parsing.

The importance of synthetic data is that it provides gold training data for the system. If the system predicts the wrong freebase relation as the meaning of *founded*, the system will not be able to predict the two missing entities correctly. This failure can be used as feedback to update the classifier. In this way, two simple declarative sentences have been transformed into labeled data for predicting grounded semantic predicates.

In our discussion, we have focused on very simple sentences with a single predicate and two entities, but this approach generalizes seamlessly to an arbitrary number of entities or predicates. In the case of more complicated sentences, several entities can be used to help make the prediction. In the following sentence, there are three entities, and both Google and Palo Alto have relations connecting them to Nest Labs:

`Google` *acquired* `Nest Labs`, *which was founded in* `Palo Alto`

In this sentence, to predict Nest Labs, there are two pieces of evidence and two predicates in use. First is the acquisition by Google and second is the location where it was founded. We can see precisely this information encoded in the graph and logical form in Figure 9.2.

Often, more than one word is required to link entities. For example, the predicate argument dependencies for *Nest Labs ... founded in Palo Alto* link

Nest Labs as the subject of *founded* and Palo Alto as the argument of *in*. In these cases, the preposition is modifying the verb, and this attachment is translated into a single predicate, *founded.in*, whose first argument was the arg1 of *founded* and whose second argument is the arg2 of *in*. One practical concern of such a simple approach to the creation of ungrounded semantic predicates is that we do not share information betwen predicates with the same verb. For example *acquire*, *acquired*, *acquired.by*, etc. are all unique and distinct tokens that need to be learned individually.

Finally, one technical note is that Reddy et al. assume neo-Davidsonian semantics [162]. This simply means that we assume there exists an event e that corresponds to each of the relations in freebase. We see this in Figure 9.2, where e1 is the shared event linking arg1 and arg2 of *acquired*, and e2 links the two arguments of *founded.in*. This translates into the inclusion of an extra variable in the logical expressions to link the arg1 and arg2 of a given predicate. We note this here simply for completeness.

9.3.1 Copulas and Special Semantic Rules

Our discussion has focused on the production and mapping of ungrounded semantic predicates to database relations. Though outside the scope of the unsupervised work presented here, there are many cases where a word or phrase correspond to a mathematical operation, not a relation. These mathematical operations (e.g. sum, max, etc.) operate over the set of entries returned by the rest of the query. Within supervised semantic parsing, knowledge of these operations is very useful for performing reasoning on top of a query. For example, to answer the question:

How many countries are members of the UN?

There is unlikely to be a numerical entry in the knowledge graph that answers this question. The list of member countries will, however, be linked or included in the graph. Executing a query that retrieves the list and computes its size allows semantic parsers to answer these type of question. We will not be handling this style of question, but feel it is a fruitful direction for future work to investigate how special predicates of this form might be learned automatically.

The one additional rule we will be encoding is for copulas. Following the

example of Reddy et al. we will produce two ungrounded graphs for sentences that contain a copula. It is best to explain why using an example.

`Obama` ***is*** *the* `US President` *and lives in* `DC`.

A copula creates an equality relation between its arguments. *"Obama is the US president"* implies that any statement that is true of Obama is also true of the US president, and vice-versa. We, therefore, add a rule that any predicate that acts on an argument of a copula should be duplicated also to hold for the copula's other argument:

Without copula rule:	**Added with copula:**
is(`Obama`, `US President`)	
lives.in(`Obama`, `DC`)	*lives.in*(`US President`, `DC`)

This additional semantic knowledge is not propagated to the parser. If the syntactic parser does not link Obama and President, then the copula rule will not fire. In this way, the semantics is enhanced to match the domain, but the parser is not given access to this supervision.

9.4 Evaluating Grounded Semantics

We have outlined how dummy predicates can be attached to a syntactic CCG parse to create an ungrounded semantic representation. Further, as noted previously, given access to (an annotated version of) ClueWeb and the Freebase, we have a procedure for learning correspondences from ungrounded predicates to grounded relations. What remains is to evaluate if the joint system is learning to map the text to the correct database relations.

Reddy et al. evaluate their system using Question Answering. They use a supervised parser to analyze questions, use the groundings they have learned to map the question to a query, and evaluate if the query returns the correct answer. Unfortunately, parsing questions is outside the scope of our parser, but a more direct semantic evaluation is possible over sentences instead.

Evaluation: Given a sentence, one entity can be removed at random, and our system will be tasked with correctly predicting it.

Sentence:

Google *acquired* Nest Labs, *which was founded in* Palo Alto.

Query:

Google *acquired* _____, *which was founded in* Palo Alto.

Answer:

Nest Labs

Performing well at this task requires that the syntactic derivation has successfully linked the relevant entities to their predicates both during training (to learn the semantic mapping) and at test time. As before, we can compute precision, recall and F1 scores. We follow the lead of Reddy et al. and use a loose metric for correctness: the correct answer is in the set of predicted entities.

Precision can be computed by dividing the number of sentences for which we predicted the correct entity by the total number of sentences for which we made a prediction. Recall is the number of correct predictions divided by the full test set, and the harmonic mean is the same formula as before: $\frac{2pr}{p+r}$.

To analyze the effect of sentence complexity on our performance, we report overall scores as well as performance for subsets of the data set broken down by the number of entities per sentence (2, 3, or 4). In addition, we previously discussed how small amounts of knowledge about the lexicon may prove very beneficial when creating a parser (Section 7.2.2). To evaluate this claim and situate our results, we will compare our model to semi-supervised and supervised parsers. Finally, we include a Bag-of-Words (BoW) baseline that does not model any syntax, to evaluate whether the syntax we discover is capturing semantically useful information.

For the supervised comparison, we will use the state-of-the-art Clark and Curran [85] (C&C) parser. This is the same parser used in the experiments by Reddy et al. (2014). As discussed before, the correct CCG syntactic parse of a sentence specifies exactly which words are predicates and how every argument slot is filled. In this way, and correspondingly in the simple resultant ungrounded semantic parse, the syntactic ambiguities of the sentence are eliminated, and a simple, unambiguous representation is provided as input to learn the semantic mapping of the sentence to the database.

9.4.1 Supervised and Bag-of-Words Comparisons

To provide a counterpoint, the BoW model (inspired by Yao (2015) [163]) entertains the possibility that every word might act to link every pair of entities in the sentence. To make the strengths of a syntax approach more explicit, we revisit our example sentence from earlier.

`Google` *acquired* `Nest Labs`, *which was founded in* `Palo Alto`.

Supervised Parser:	$acquired($`Google`$, x) \wedge founded.in(x,$ `Palo Alto`$)$
Bag-of-Words:	{`Google`, `Palo Alto`} + {acquired, founded, in, was, which}

Supervised Parser The syntactic parse provided by a supervised parser will identify that there are two predicates, each attached to distinct entities (`Google` and `Palo Alto`) which are looking for either an initial or second argument. This is a very precise relation, and intersection of entities.

Bag-of-Words In contrast, the BoW model only has access to two pieces of knowledge: 1. There is a set of entities in the sentence: {`Google`, `Palo Alto`}, and 2. One or two of the other words in the sentence should predict the missing entity. We show these two sets above.

The BoW model is therefore presented with the union of entities linked to either entity in the graph, and must find the most discriminating word(s) in the remainder of the sentence to use when predicting the answer (e.g. `Nest Labs`).

As compared to the supervised parser, the BoW model has a tremendous amount of freedom in choosing relations to predict the missing entity. In this case, it might only learn the acquisition relation, but still get the answer correct. In contrast, the supervised parser would require correct knowledge of both *acquired* and *founded.in* to make the correct prediction.

This lack of information about the structure of the sentence is also a weakness. If the task were to predict `Palo Alto`, the BoW model should have more difficulty choosing the right city as it is equally like to predict a location based on knowledge of `Google` (Menlo Park) as based on `Nest Labs` (Palo Alto).

9.5 Semi-Supervised Grammars

We have outlined the strongest (supervised parser) and weakest (BoW model) approaches to the task of learning semantic groundings. We are also interested in evaluating how the unsupervised model we have been developing throughout the thesis will compare to these approaches. We hope its parses will constrain the semantic groundings better than a BoW model, but with equally little supervision.

The final approach we compare is a semi-supervised version of our model. We claimed earlier in our analysis of the unsupervised parser's failings (Section 7.2.2) that a small set of categories might be sufficient to drastically improve the model's parsing performance. Further, we also noted that our unsupervised models did not even have access to the PP category.

9.5.1 The Impact of Supervision

To test how knowledge of the lexicon of the category PP affects our model we created six lexicons from the training section of the English CCGbank.

Supervised parsers have three main advantages over a system like ours. First, they have access to a grammar and lexicon that are both (much more) complete[2] and correct[3] than what the induction algorithm returns. Second, they are provided with correct parses (including lexical categories and attachments) during training. Third, their models have access to richer features and more data than our models are currently trained on. To assess the importance of these differences, we compare our unsupervised parser with a weakly supervised variant, which uses the same probability model, but is trained on gold lexicons derived from CCGbank, and with the fully supervised parser (HWDep) of Hockenmaier and Steedman (2002) [98]. We chose to compare against the HWDep model because it is a simple generative model which models lexical dependencies where our approach does not. In this way, HWDep is better suited to take advantage of the full annotated syntactic parse than our model.

[2]Lexical coverage is a problem even for supervised CCGbank parsers.

[3]CCGbank also contains some rare categories and rules that are probably incorrect.

Weakly supervised parsing with partial gold lexicons						
	Without PP			With PP		
	90%	**95**%	**99**%	**90**%	**95**%	**99**%
# Categories	32	36	65	36	44	94
Ave # Cats / Tag	3.2	4.3	7.4	3.6	5.0	9.4
Token Coverage	89.4	91.2	94.4	92.7	95.2	98.7
Type Coverage	20.9	23.6	41.9	23.6	29.1	58.1
Type Precision	96.9	97.2	95.4	97.2	97.7	95.4
Sentence Coverage	61.8	66.4	76.3	64.6	73.2	90.4
Parse Coverage	78.8	92.0	99.9	80.0	92.4	99.9
ST Accuracy	59.1	**69.3**	67.5	60.0	**69.5**	63.3
LF1	48.2	**50.5**	41.3	49.3	**51.9**	37.4
UF1	61.2	**65.0**	58.0	63.6	**69.5**	61.0

Table 9.3: Performance (Section 7.2) of the weakly supervised parser on Section 22 of the English CCGbank. In addition we report the same ambiguity metrics used for induced lexicons in Section 4.4.3. We computed partial lexicons based both on the total token distribution (right) and for words whose gold tag did not include the PP category.

9.5.2 The Weakly Supervised Parser

We now evaluate our model when trained in a weakly supervised fashion. Instead of using the automatically induced lexical categories to create the parse forests that the model is trained on, we use (partial) gold lexicons derived from CCGbank. This is similar in spirit to Boonkwan and Steedman's [38] semi-supervised approach to CCG induction, which requires a linguist to help construct the lexicon. Since the categories considered by our parser are defined by a word's POS tag, we define our CCGbank gold lexicons also in terms of tags rather than words. For each tag, we find the set of categories that cover 90%, 95% and 99% of its occurrences in our training sentences. Since the induction algorithm does not consider the category PP, we also consider only tokens whose lexical categories do not contain a PP result or argument. We again evaluate against simplified CCGbank categories (with PPs), and parse with the same $\mathsf{B}_3^{\mathsf{C}}$ settings as in the fully unsupervised case with complex arguments (allowing restricted type-raising which only includes the categories S/(S\N) and S\(S/N), and punctuation, but no type-changing).

There are two important caveats to note about this experimental setup.

First, the lexicons are constructed based on token coverage, not sentence coverage. Therefore, in Table 9.3 we see that even when 99% of a tag's gold categories are included, only 90.4% or 76.3% (with and without PP) of gold parses are recoverable. Second, the model is provided an unweighted lexicon. We are not experimenting with providing additional information from the corpus to the system (e.g. frequency counts for categories).

The numbers presented here parallel those computed for induced grammars (Section 4.4.3), but because these are gold (tag based) lexicons, the average number of categories per tag is much smaller (an order of magnitude) than the ambiguous categories introduced by induction. Further, all metrics reported are higher than in the case of the induced lexicon.

9.5.3 Performance of the Weakly Supervised Parser

Table 9.3 shows the performance of the weakly supervised parser (B^3 with punctuation, lexicalization and complex arguments, trained and tested on the same data as in Table 7.5) with CCGbank lexicons.[4] In contrast to the induced lexicons which has no parse failures, only the lexicons with 99% token coverage can parse the unseen data without (basically) any parse failures. The 90% lexicons yield too many parse failures ($> 20\%$) to achieve good results. But the model seems to fail to properly utilize the additional categories provided by the 99% lexicon. We see this in the fact that the correct sequence of categories exist in 10-15% more of the sentences with 99% coverage than 95% coverage, but the model's performance drops.

The lexicon with 95% coverage and PPs (G95%) is the only one that outperforms the unsupervised parser ($\mathsf{B}_{\mathbf{3}}^{\mathbf{P\&L}}$) on all metrics. What stands out is the very small increase in labeled dependency recovery (LF1) that the 95% lexicons have over the 90% lexicons, even though they have far fewer parse failures (7.6% vs. 20.0%), and much higher (labeled) supertagging accuracies (69.5 vs. 60.0). This is presumably due to the simplicity of our model (which does not capture word-word dependencies) and to the fact that raw sentences do not carry enough signal for an unsupervised system to learn correct attachments.

The benefit of knowing the additional categories provided by a semi-supervised lexicon becomes apparent in Table 9.4, which reproduces some

[4] All results reported with $\alpha = 5$

	First Argument						Second Argument					
	∼DEP	**%S22**	**B_1**	**$B_3^{P\&L}$**	**G95%**	**HWDep**	**∼DEP**	**%S22**	**B_1**	**$B_3^{P\&L}$**	**G95%**	**HWDep**
N/N_1	NMOD	*30.6*	68.4	71.6	75.0	92.4						
$S\backslash N_1$	SUB,NMOD	*3.2*	12.2	14.6	28.6	68.9						
$S\backslash S_1$	VMOD	*2.0*	17.0	18.7	21.4	74.3						
S/S_1	VMOD,SBAR	*1.9*	24.0	33.8	30.4	80.9						
$(N\backslash N_1)/N_2$	NMOD	*5.5*	49.7	51.2	58.7	74.9	PMOD	*6.1*	41.0	42.4	53.1	79.9
$(S\backslash N_1)/N_2$	SUB	*4.4*	26.6	34.4	38.3	73.1	OBJ	*4.9*	30.6	33.8	42.1	78.3
$(S\backslash S_1)/N_2$	VMOD	*3.1*	21.6	24.7	6.7	64.0	PMOD	*3.5*	24.0	29.3	8.2	69.2
$(S\backslash N_1)/S_2$	SUB	*1.1*	23.9	32.5	49.8	82.6	VMOD	*1.1*	25.2	35.0	38.8	85.1
$(S\backslash S_1)/S_2$	VMOD	*0.5*	6.1	14.1	20.8	67.9	SBAR	*0.6*	9.5	19.5	31.9	76.1
PP/N_1	PMOD	*2.2*			33.6	69.2						
$(S\backslash N_1)/PP_2$	SUB	*1.2*			40.0	62.2	VMOD	*1.2*			42.8	68.0
$(N/N_1)\backslash N_2$	NMOD	*1.2*			83.2	92.7	NMOD	*1.2*			87.0	91.6
$(S/S_1)/N_2$	VMOD	*0.8*			56.7	81.3	PMOD	*0.8*			54.6	82.9
$(N/N_1)/(N/N)_2$							AMOD	*0.9*			18.9	66.7
$(N\backslash N_1)/(S\backslash N)_2$	NMOD	*0.8*			59.0	69.7	SBAR	*0.8*			62.0	87.5
$(S\backslash N_1)/(S\backslash N)_2$	SUB	*4.1*			51.1	86.8	VC,VMOD	*6.3*			68.0	92.1

Table 9.4: Detailed error analysis: LF1 scores of unsupervised B^1 and $B_3^{P\&L}$, the best weakly supervised model (G95%), and the supervised model (HWDep), on the most common 90% of (simplified) dependency types in Section 22. For each label, we report the fraction of dependencies covered (%S22) and the CoNLL dependency labels that account for 75% of undirected dependency overlap (∼DEP). The first and second arguments of the category are marked by subscripts. These correspond to the performance being reported in the left (first) and right (second) columns. Categories that cannot be induced or are ignored by the unsupervised models are presented in the bottom half.

Model	Supervision	UF1	LF1
$\mathbf{B_1}$	POS tags	60.6	34.5
$\mathbf{B_3^{P\&L}}$	+ Punc & Words	63.6	37.1
G95%	+ Partial Lexicon	70.0	52.6
HWDep	Fully supervised	88.5	80.3

Table 9.5: Overall performance on Section 23 of the systems discussed in this Chapter.

of the detailed error analysis from Table 7.11 in Section 7.2.2 but also includes the semi-supervised and fully supervised models trained on the same data. We see how categories like the possessive, relative pronouns and auxiliaries are nearly unambiguous constructions. The model still does not capture bilexical dependencies that might be necessary for improved attachment decisions. Additionally, the most dramatic failing of the semi-supervised lexicon is the attachment of verb modifying prepositions ($(\mathsf{S}\backslash\mathsf{S})/\mathsf{N}$) (6.7 LF1 vs. 24.7 LF1 for $\mathbf{B_3^{P\&L}}$), presumably due to argument-adjunct ambiguity with PP/N.

Finally, we can provide the first side-by-side labeled comparison of an unsupervised, semi-supervised and supervised system on Section 23 in Table 9.5. Once again this table makes clear that, while many of the attachments learned by unsupervised models are correct, we still have immense progress to be made on labeled evaluation metrics.

9.6 Slot Filling Performance

We now evaluate our unsupervised and semi-supervised systems (trained on length 20 sentences from the WSJ) against a Bag-of-Words (BoW) baseline (one in which any word, or pair of words in the sentence can act as a relation to link entities) and against the state-of-the-art supervised syntactic parser of Clark and Curran [85] (C&C). For each of the four syntactic representations (BoW, unsupervised, semi-supervised, supervised), we train the system described in Section 9.3 on nearly 85,000 declarative sentences and test on just under 10,000 sentences in which one entity has been randomly removed. The full dataset of 100,000 sentences was randomly sampled to create a train-development-test split of 85-5-10. The development data was used to discard perceptron gradient steps which hurt accuracy during training. The results

	Prec	Recall	F1
Clark & Curran	38.7	38.1	38.4
Semi-Supervised	37.5	35.0	36.2
Unsupervised	32.6	30.5	31.5
Bag-of-Words	33.1	33.1	33.1

Table 9.6: Slot filling performance (Section 9.4) of different syntactic models. For 10,000 with a randomly dropped entity, we are computing what percentage of our predictions are correct (precision), what percentage of the data set we correctly predict (recall), and the harmonic mean of these values (F1).

are presented in Table 9.6.

Recall that the syntactic parse constrains the possible semantic interpretation of a sentence. Our goal when analyzing this table is to see whether the constrained semantics licensed by the syntactic parsers are correct and allow the system to learn. Not surprisingly, the fully supervised and semi-supervised approaches outperform the baseline, but the unsupervised system appears to fall short. In particular, we see that with very little annotation the semi-supervised system nearly matches the performance of a fully supervised state-of-the-art parser (36.2 vs. 38.4), while the unsupervised system performs worse than the Bag-of-Words baseline (31.5 vs. 33.1).

When reasoning about this apparently negative result for our unsupervised approach, we realized that the strengths and weaknesses of the BoW and unsupervised approach should be complementary. In sentences in which there are only two entities, there is likely a single predictive verb, which, if improperly analyzed by the syntax, will cause the system to fail. A common example being our system's inability to capture the non-local dependencies of the auxiliary (Section 7.2.3):

`Obama` *was* *born* *in* `Hawaii`

We find upon analyzing the output of our system that we are predicting *was* as the ungrounded semantic predicate: *was.in*(`Obama`, `Hawaii`). When this mistake is aggregated across a large corpus, the semantics of founded.in, born.in, etc. are all mapped to the increasingly ambiguous semantic predicate *was.in*(). This one-to-many mapping makes it nearly impossible for the semantic grounding to succeed. In contrast, the BoW model just picks out *born* as the most discriminating word in the sentence, as it is the best

	Overall		2		3		4	
BoW	**33.1**		**38.8**		**25.1**		12.8	
C&C	38.4	5.3	44.4	5.6	28.7	3.6	26.0	13.2
Semi-Supervised	36.2	3.1	41.5	2.7	27.0	1.9	23.9	11.1
Unsupervised	31.5	-1.6	35.8	-3	**24.9**	-0.2	**17.9**	5.1

Table 9.7: Slot filling performance of different syntactic models as a function of the number of entities in the sentence. We report F1 for each approach and the absolute gains/losses of the syntax-based models as compared to the BoW approach.

predictor of the missing entity. In doing so, it ignores the syntax of the sentence entirely to extract the one descriptive word. For this reason, we would expect the BoW model to shine on simple sentences.

At the other end of the extreme are sentences with three or four entities. For example, in our example from before:

`Google` *acquired* `Nest`, *which was founded in* `Palo Alto.`

If we randomly select Palo Alto to be predicted, the clause attachment becomes crucially important. Both Google and Nest were founded in cities, but only one was founded in Palo Alto. For this reason, we would expect the BoW model to perform at chance, drastically lowering its performance. In contrast, while syntax-based techniques also suffer from decreased performance due to ambiguity in longer sentences, we would expect their relative performance to increase on three and four entity sentences.

To investigate this further, we split up the evaluation sentences by the number of entities in the sentence (including the removed entity). We then computed the prediction F1 for each subset of the test set and report the results in Table 9.7 and relative performance in Figure 9.3.

Upon analyzing the results of these experiments, it becomes apparent that the real strength of syntax is on more complex sentences. The most exciting result being the final column where the performance of the unsupervised system on sentences with four entities nicely outperforms the BoW model.

We should note that even the supervised system is far from perfect. This is likely due to the nature of the data. The data is web text that we believe to have linked entities, but there are no assurances that the sentences in our corpus refer to the set of relations in Freebase. Addressing this concern and

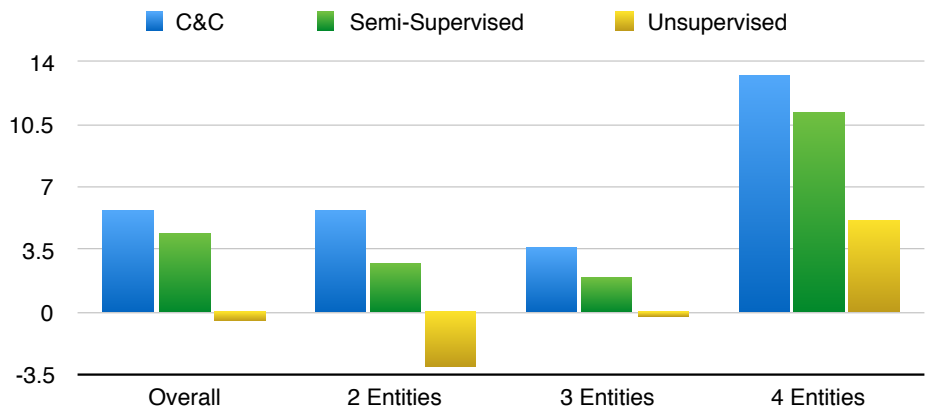


Figure 9.3: Relative absolute F1 performance of our three syntax-based approaches as compared to the Bag-of-Words model.

building better semantic parsers is outside the goal of our work. We have demonstrated the utility of syntax, even when unsupervised, for semantic grounding.

9.7 Conclusions

There are two primary contributions in these experimental results. The first is that, unsurprisingly, a somewhat limited gold lexicon (44 categories) can lead to the creation of a semi-supervised model that nearly matches supervised performance despite not having access to full treebank. It is possible that the questionnaire used by the semi-supervised approach of Boonkwon and Steedman [164] might be sufficient for building an effective semantic parser in new languages.

The second and perhaps most significant and surprising contribution is that an unsupervised approach discovers enough syntactic structure automatically from the text to beat a Bag-of-Words model on long sentences. From a practical standpoint, the complementary errors of the two systems might be combined in future work to produce a very successful semantic parser. Additionally, because the BoW model is better at isolating the primary semantic predicate on short sentences, future work might integrate this signal into the unsupervised grammar induction process to improve the syntactic parser.

A final point worth noting is that we have now created an information

extraction system from very limited resources. Our approach assumes access to an initial database of facts and a corpus of automatically tagged text. To deploy this system in a new language would only require retraining one of our unsupervised or semi-supervised parsers and translating the name of entities. We are therefore slightly closer to being able to quickly deploy information extraction systems in languages where there are no treebanks.

Chapter 10

Conclusions

This thesis introduces a state-of-the-art unsupervised grammar induction procedure. We use Combinatory Categorial Grammars to produce labeled structures. This allows us to perform an in-depth linguistic analysis of the approach and a direct head-to-head performance comparisons with supervised parsers on both syntactic and semantic evaluations.

Early work in grammar induction produced simpler, less descriptive, and discriminating structures. This was essential for starting the field, but fundamentally limiting for its future success. Labeled structures, particularly those that carry semantic content, are necessary for transitioning unsupervised grammar induction from an intellectual curiosity to a standard tool in the NLP toolkit.

In chapter 4, we introduced a novel means of inducing a minimally supervised grammar. We only assume knowledge of the basic distinction between nouns and verbs, and demonstrated how this distinction can be leveraged into the creation of a complete language specific grammar. Our approach performs at or better than state-of-the-art in over a dozen languages (Chapters 5 and 6). We achieve this with a novel factorization of CCG and nonparametric model.

By diverging from previous work and using CCG (Chapter 3), we gleaned important insights into necessary changes to the field of unsupervised grammar induction (Chapter 7). We demonstrated how these structures can be learned in a minimally supervised setting (Chapter 8), and how unsupervised syntax can be used for semantic grounding (Chapter 9). All of these contributions were made possible by our novel use of a rich syntactic formalism, Combinatory Categorial Grammars, for producing an unsupervised parser.

Over the last decade grammar induction transitioned from parsing short sentences in a few languages to reporting impressive performance on full length sentences in over a dozen languages. We demonstrate in this the-

sis that the numbers reported in the literature paint a rosier picture of the utility of grammar induction approaches than what could be expected to achieve in practice due to the common practice of evaluating an unlabeled representation. Our work is the first to both automatically induce linguistic labels (corresponding to adjective, transitive verb, verb/noun attaching preposition, etc.) and accurately use them in an unsupervised framework.

These labels and the distinctions they make have long been essential to the creation of supervised NLP systems. We demonstrated their utility for semantics in Chapter 9, but believe the scope of their influence is much greater and encompasses any current NLP or Data Mining system that includes a syntactic parser.

Future Work Despite progress, both within this thesis and the field, there is more work that needs to be done in both decreasing supervision and improving grammar induction. Future work should harness more naturalistic sources of supervision and make advances in the following areas:

1. Remove any reliance on Part-of-Speech tags
2. Integrate semantic feedback from the world
3. Model lexical semantics

We have demonstrated initial results on performing grammar induction with knowledge of a three-way split between nouns, verbs, and others (Chapter 8). There are distributional properties of these classes that may allow for their discovery without labeling. But if our belief that nouns are truly semantic primitives that are easiest to learn from the world is correct, we should also be able to utilize object detection in vision, entities in databases, or beings in a virtual world as sources of nouns/entities for learning. Further, if verbs are special semantic predicates that form queries or describe the interactions of people in the world, they too should be discoverable from the environment without the need for explicit annotation.

One aspect of grammar induction that has become clear is how interconnected syntax and semantics are. The fundamental limitation in progress in grammar induction and closing the performance gap with supervised syntactic parsers the need to capture semantics. Starting from the basic vocabulary

of the grammar, up to the attachment decisions we make, and the form of the categories, nearly every aspect of learning syntax is informing or informed by semantics. It, therefore, seems silly to continue to treat the acquisition of grammar as disjoint from semantics. Unfortunately, semantics is a vague term within NLP and takes many forms. We will focus on two very concrete definitions when discussing future work: Lexical and Database Semantics.

Lexical semantics attempts to use the distributional properties of word co-occurrences to find low-dimensional vector spaces that maintain semantic relations as spatial relations [165, 166, 167, 168, 169, 170]. These properties can be automatically extracted from the text and do a very good job at finding typological information. Our approaches do not model lexical dependencies. One might imagine that given clusters that separate the many types of actors found in text (e.g. companies, people, animals) and their respective actions (mergers and acquisitions versus run and play), a model could be built to disambiguate many attachment decisions, particularly if the model and representation can be trained jointly. This representation, without grounding in an environment, will always be shallow, but potentially very powerful.

Database/Logical semantics provide a less ambiguous representation of world knowledge which can be directly queried. In the case of a robot in the world, they can take an action and receive feedback from the environment. In a database, the feedback comes as an answer to a query or a failure to execute. By building off the work we have performed on grounding language to Freebase, one can easily imagine re-incorporating that signal to improve our model. By restructuring the training objective to take input from the world, syntax can make predictions about the world, or content of a sentence, and have the predictions verified experimentally through interaction with the environment [171, 172].

While these are three very explicit next steps, our stance more holistically about language learning and grammar induction is that the syntax and semantics must be learned jointly. POS tags are syntactically and semantically informative, syntax relies on and informs semantics, and semantics is not recoverable in isolation. Whether the way forward is a loop [40] or a joint model, we do not know, but future work should aim to build systems

that learn from the environment and use as little supervision as possible. We believe this will be possible by exploiting the natural supervision that exists in each of these tasks alone: distributional regularities and feedback from the environment.

Appendix A

UPOS/Seed Knowledge

The following are cut-and-paste copies of the seed knowledge files used in the thesis experiments. There are a number of places where performance can be improved by removing the verb seed knowledge from specific tags (e.g. Gerunds, some participles, etc) which should not act as heads of sentences.

A.1 Hebrew UPOS tags

POS Tag	UPOS	Explanation
!!MISS!!	X	means analysis is not in the lexicon
!!SOME_!!	X	
!!UNK!!	X	means the word is not in the lexicon
!!ZVL!!	X	
ADVERB	ADV	
AT	X	Accusative Marker
BN	VERB	
BN_S_PP	VERB	
BNT	VERB	Gerund
CC	CONJ	Coordinating conjunction
CC-COORD	CONJ	Coordinating conjunction
CC-REL	CONJ	
CC-SUB	CONJ	Subordinating conjunction
CD	NUM	Numeral (definite)
CDT	NUM	Numeral determiner (definite)
CONJ	CONJ	
COP	VERB	Copula
COP-TOINFINITIVE	VERB	to be
DEF	DET	The (H)
DEF@DT	DET	All (HKL)
DT	DET	Determiner
DTT	DET	All, how many
EX	VERB	Existential?
IN	ADP	Preposition (EL)
INTJ	X	Interjection
JJ	ADJ	Adjective (definite)
JJT	ADJ	Construct state adjective
MD	VERB	Modal
NCD	NUM	Date/Time
NN	NOUN	Noun (definite — definite-genetive)
NNP	NOUN	Proper noun
NNT	NOUN	Construct state noun
NN_S_PP	NOUN	Possessive noun (paney-hem)
P	X	Prefix
POS	PRT	Possessive item (shel)
PREPOSITION	ADP	
PRP	PRON	Personal Pronoun
PRP-REF	PRON	
PRP-PERS	PRON	
PRP-DEM	ADP	
PRP-IMP	NOUN	
PUNC	.	
QW	X	Question/WH word
RB	ADV	Adverb
REL	ADP	Relativizer
REL-SUBCONJ	ADP	(she)-lifnei etc

A.1 cont'd

POS Tag	UPOS	Explanation
S_ANP	PRON	Pronoun (suffix)
S_PRN	PRON	Pronoun (suffix)
TEMP-SUBCONJ	CONJ	WH / Conj e.g. when KF
TTL	DET	Title
VB	VERB	Verb
VB-BAREINFINITIVE	VERB	
VB-TOINFINITIVE	VERB	

A.2 Arabic (Coarse)

POS Tag	Seed
A	
C	conj
D	
G	punct
F	
I	
-	
N	noun
Q	noun
P	
S	noun
V	verb
Y	
Z	noun
X	

A.3 Arabic

POS Tag	Seed	Explanation
Z-	noun	Proper nouns?
D-		Adverb
Y-		List item?
N-	noun	Noun
C-	conj	Conjunction
–		the
VI	verb	Imperfect - Pain, Feel, Be, Riding
A-		Adjective
VP	verb	Perfect - Ask, return, answered, achieved
VC	verb	Imperative - Check, called, note, let the
I-		Hello, both, of course
FN		PRT (e.g. no,but,not)
S-	noun	Pronoun
SR	noun	Pronoun
FI		PRT (e.g. what,of,are)
G-	punct	Punctuation
SD	noun	Pronoun
Q-	noun	Number
F-		PRT (e.g. the,has,any)
P-		Preposition
X		Lots of FW + ?

A.4 Basque (Coarse)

We found several mappings to be inconsistent in the Shared Task data. Some of the other possible interpretations are listed:

POS Tag	Seed	Alternatives
ADT	verb	X VERB
BEREIZ	punct	
ITJ	noun	
IZE	noun	ADV NOUN DET PRON X ADJ
BST	noun	X DET
DET	noun	
IOR	noun	
PRT		
ADB		
LOT	conj	
ERL		
PUNT_MARKA	punct	
ADL	verb	
HAOS		
ADJ		
ADI	verb	X VERB

A.5 Basque

Again, there is some some inconsistency in the mappings

POS Tag	Seed	Alternatives/Explanation
BEREIZ	punct	”
DZG	noun	DET
DZH	noun	DET
ADT	verb	VERB
PERIND	noun	PRON
PUNT_KOMA	punct conj	, Comma
ERL		PRT
ERKARR	noun	DET
PUNT_PUNT	punct	.
PUNT_GALD	punct	?
BAN	noun	DET
ADI_IZEELI	verb	VERB
SIN	verb	ADI_SIN (VERB common) ADJ_SIN (adj, 1 in train)
IZE_IZEELI	noun	NOUN
HAOS		X (are → are being)
LIB	noun	Proper? Noun
PRT		Yes, No
BST		Usually, Then, Of
IZGGAL	noun	Who, Whom, ... Pronoun
ADT_IZEELI	verb	VERB
PUNT_ESKL	punct	!
IOR_IZEELI	noun	our, mine, ... Pronoun
FAK	verb	VERB (ADI_FAK)
ADB_IZEELI		ADV
PUNT_BI_PUNT	punct	
PERARR	noun	We, our, I ... Pronoun
PUNT_HIRU	punct	:
LOK		Also, In addition, ...
ERKIND	noun	DET
ELK	noun	PRON
IZB	noun	NOUN
ZKI	noun	NOUN (IZE_ZKI)
ADP	verb	VERB (ADI_ADP)
IZGMGB	noun	PRON (IOR_IZGMGB)
MEN		Although, and (... so SC and CC ?)
DET_IZEELI	noun	DET
PUNT_PUNT_KOMA	punct	;
ADL	verb	VERB (was?)
ADJ		ADJ
ADK	verb	VERB (ADI_ADK)
ARR		(ADJ_ARR = ADJ) (IZE_ARR = NOUN) (ADB_ARR = ADV)
ITJ		X
JNT	conj	conj (LOT_JNT)
ADJ_IZEELI		ADJ
NOLGAL	noun	DET (DET_NOLGAL)

A.5 cont'd

POS Tag	UPOS	Explanation
GAL		(ABD_GAL = ADV) (ADJ_GAL = ADJ)
ORO	noun	DET (DET_ORO)
ADL_IZEELI	verb	VERB
ORD	noun	DET (DET_ORD)
NOLARR	noun	DET (DET_NOLARR)

A.6 Bulgarian

POS Tag	Seed	Alternatives/Explanation
A		Adjective
Af		Adjective, feminine
Am		Adjective, masculine
An		Adjective, neuter
Cc	conj	Conjunction, coordinative
Cp	conj	Conjunction, subordinative
Cr	conj	Conjunction, repetitive coordinative
Cs	conj	Conjunction, single and reptitive coordinative
D		Adverb
Dd		Adverb, model
Dl		Adverb, location
Dm		Adverb, manner
Dq		Adverb, quantity and degree
Dt		Adverb, time
H		Hybrid Adjective
Hf		Hybrid Adjective, feminine
Hm		Hybrid Adjective, masculine
Hn		Hybrid Adjective, neuter
I		Interjection
Mc	noun	Cardinal numerals
Md		Adverbial numerals
Mo	noun	Ordinal numerals
My	noun	Fuzzy numerals about people (few,many)
N	noun	Noun
Nc	noun	Common nouns
Nm	noun	Noun masculin
Np	noun	Proper nouns
P	noun	Pronoun
Pc	noun	Collective pronouns
Pd	noun	Demonstrative pronouns
Pf	noun	Indefinite pronouns
Pi	noun	Interrogative pronouns
Pn	noun	Negative pronouns
Pp	noun	Personal pronouns
Pr	noun	Relative pronouns
Ps	noun	Possessive pronouns
Punct	punct	
R		Preposition
Ta		Particle, affirmative
Te		Particle, emphasis
Tg		Particle, gradable
Ti		Particle, interrogative
Tm		Particle, modal
Tn		Particle, negative
Tv		Particle, verbal
Tx		Particle, auxiliary

A.6 cont'd

POS Tag	UPOS	Explanation
V	verb	Verb
Vii	verb	Auxiliary
Vni	verb	Verb, impersonal, imperfective
Vnp	verb	Verb, impersonal, perfective
Vpi	verb	Verb, personal, imperfective
Vpp	verb	Verb, personal, perfecive
Vxi	verb	Auxiliary (to be)
Vyp	verb	Auxiliary

A.7 CHILDES (coarse)

POS Tag	Seed
co	
int	noun
pro	noun
tag	punct
inf	verb
conj	conj
adv	
neo	
neg	
.	punct
rel	conj
test	
aux	verb
adj	
prep	
fil	
v	verb
part	verb
L2	
fam	
post	
poss	
mod	verb
on	
det	noun
n	noun
wplay	
bab	
chi	
unk	
qn	noun

A.8 CHILDES

POS Tag	Seed	Explanation
!	punct	
+”/.	punct	Quotation Follows (content from story reading)
+...	punct	Trailing off
+/.	punct	Interruption
.	punct	
?	punct	
0prep		
0pro	noun	
L2		Second=language form
adj		Adjective
adj:n		Adjective, (Careless, Squirrely, ...)
adj:v		Adjective, (Double, gently, ...)
adv		Adverb (well)
adv:adj		Adverb, ending in ly (quickly)
adv:int		Adverb, intensifying (very, rather)
adv:loc		Adverb, locative (here,then)
adv:tem		Adverb, time (then, today)
adv:wh		Adverb, (when, why)
aux	verb	verb, modal auxiliary
bab		babbling
chi		child-invented form
co		Communicator (aha)
co#v	verb	(copilot)
co:voc		(Dear) Adverb?
conj:coo	conj	Conjunction, coordinating (and,or)
conj:sub		Conjunction, subordinating (if, although)
conj:subor		Conjunction,
det	noun	Determiner
det:num	noun	Number
det:wh	noun	(Which, Who)
dis#n:gerund	noun	(disappearing)
dis#part	verb	(disappeared)
dis#v	verb	(disappear)
fam		Family-specific form (buba, oy)
fil		Filler (hmm)
inf	verb	Infinitive market _to_
int		interjection, interaction
mis#part	verb	(misplaced, misunderstood)
mod	verb	(did,do,may,will)
n	noun	Noun common
n:adj	noun	(-ness)
n:gerund	noun	-ing (missing, doing, ...)
n:let	noun	Multiple letters
n:prop	noun	Proper noun
n:pt	noun	(Pants, clothes)
n:v	noun	(-er)
neg		negation

A.8 cont'd

POS Tag	UPOS	Explanation
neo		neologism (becauses)
on		onomatopoeia
part	verb	Participle (doing)
poss		Possessive
post		(else, too)
pre#n	noun	(preschool)
prep		Preposition
pro	noun	Pronoun
pro:dem	noun	Pronoun, demonstrative (this, that)
pro:indef	noun	Pronoun (one,somebody,...)
pro:poss	noun	Pronoun (mine,yours,his)
pro:poss:det		Pronoun (her, your)
pro:refl	noun	Pronoun (-self)
pro:wh	noun	(what)
qn		Quantifier
re#part	verb	(reheated)
re#v	verb	(rewind)
rel		(that, which)
tag	punct	(,,)
test		test word
un#adj		(un-) unhappy,...
un#n	noun	(un-) untie, unscrew, ...
un#part	verb	(un-) unmade, untied, ...
un#v	verb	(un-) untie, unbutton, ...
under#n	noun	(undershirt)
unk		Excluded words (xxx,www,...)
v	verb	Verb
v:cop	verb	Verb, Copula (is,be)
wplay		(caboozle, billy, toebow, ...)

A.9 Chinese

POS Tag	Seed	Explanation
?	punct	
!	punct	
.	punct	
AD		adverb
AS		aspect particle
BA		ba3 ba-construction: he BA you cheat AS <He cheated you>
CC	conj	coordinating conjunction
CD	noun	cardinal number
CS		subordinating conjunction
DEC		de5 as a complementizer or a nominalizer: S/VP DEC NP
DEG		de5 as a gitive marker and an associative marker: NP/PP/JJ/DT DEG NP
DER		Resultative de5: he run DER very fast
DEV		Manner de5: happy DEV speak ¡speak happily¿
DT	noun	Determiner
ETC		etc
FW		Foreign Word
IJ		Interjection
JJ		noun-modifier
LB		bei4 in long bei-construction: he LB I scold AS one M ¡He was scolded by me¿
LC		Localizer
M	noun	Measure Word
MSP		Other Particle: appear before a VP
NN	noun	Other Noun
NR	noun	Proper Noun
NT	noun	Temporal Noun
OD	noun	Ordinal Number
ON	noun	Onomatopoeia
P		Preposition
PN	noun	Pronoun
PU	punct	Punctuation
SB		bei4 in short bei-construction: he SB scold AS one M ¡He was scolded¿
SP		Sentence-final particle
VA	verb	Predicate Adjective (verb)
VC	verb	Copula (verb)
VE	verb	you3 as the main verb
VV	verb	other verb

A.10 Czech (Coarse)

POS Tag	Seed	Alternatives
A	noun	PRON ADJ
C	noun	
D		
I		
J	conj	NUM
N	noun	
P	noun	PRON ADP
R	noun	PRON ADP
T		
V	verb	
X		
Z	punct	

A.11 Czech

POS Tag	Seed	Explanation
*	noun	Word krt (lit. times)
,		Conjunction Subordinate
1	noun	PRON
2		ADJ
4	noun	Relative/interrogative pronoun w/ adjectival declension
5	noun	The pronoun he in forms requested after any preposition
6	noun	Reflexive pronoun se in long forms
7	noun	Reflexive pronouns s, plus contracted
9	noun	Relative pronoun ... after a preposition (n-: lit. who)
8	noun	Possessive reflexive pronoun svj (lit. my/your/her/his when the possessor is the subject of the sentence
:	punct	Punctuation
=	noun	Number written using digits
?	noun	Numeral kolik (lit. how many)
@		Unrecognized word form
A		Adjective, general
B	verb	Verb, present or future form
C		Adjective, nominal
D	noun	Pronoun, demonstrative (ten, onen, ..., lit. this, that, that ... over there, ...)
E	noun	Relative pronoun co (corresponding to English which in subordinate clauses referring to a part of the preceding text)
F		Preposition, part of; never appears isolated, always in a phrase (lit. regardless, because of)
G		Adjective derived from present transgressive form of a verb
H	noun	Personal pronoun, clitical (short) form
I		Interjections
J	noun	Relative pronoun (not after a preposition)
K	noun	Relative/interrogative pronoun
L	noun	Pronoun, indefinite
M		Adjective derived from verbal past transgressive form
N	noun	Noun (general)
O	noun	Pronoun
P	noun	Personal Pronoun
Q	noun	Pronoun relative/interrogative
R		Preposition (general, without vocalization)
S		Pronoun possessive
T		Paricle
U		Adjective possessive
V		Preposition (w/ vocalization)
W	noun	Pronoun negative
X		(Temporary) word form recognized, missing tag
Z	noun	Pronoun indefinite
^	conj	Conjunction (connecting main clauses)
a	noun	Numeral, indefinite
b		Adverb (w/ possibliel to form neg)
c	verb	Conditional
d	noun	Numeral, generic w/ Adjectival declension

A.11 cont'd

POS Tag	UPOS	Explanation
e	verb	Verb, transgressive present
f	verb	Verb, infinitive
g		Adverb, forming negation and comparision
h	noun	Numeral, Generic
i	verb	Verb, imperative
k	noun	Numeral, generic greater ≥ 4 used as adj
l	noun	Numeral, cardinal
m	verb	Verb, past transgressive
n	noun	Numeral, cardinal ≥ 5
o	noun	Numeral, multiplicative indefinite
p	verb	Verb, past participle, active
r	noun	Numeral, ordinal
s	verb	Verb, past participle, passive
t	verb	Verb, present or future tense
u	noun	Numeral, interrogative
v	noun	Numeral, multiplicative, definite
w	noun	Numeral, indefinite, adjectival declension
y	noun	Numeral, fraction ending at -ina, used as noun
}	noun	Numeral, written using Roman numerals

A.12 Danish (Coarse)

POS Tag	Seed	Explanation
AC	noun	cardinal numeral
AN		'adjective' encompasses 'normal' adjectives
AO	noun	ordinal numerals
CC	conj	coordinating conjunctions
CS		subordinating conjunctions
I	noun	Interjections
NC	noun	common nouns
NP	noun	proper nouns
PC	noun	Reciprocal pronouns
PD	noun	demonstrative pronouns
PI	noun	Indefinite pronouns
PO	noun	Possessive pronouns
PP	noun	Personal pronouns
PT	noun	interrogative/relative pronouns
RG		adverbs
SP		prepositions and postpositions
U		most adverbs are marked as 'unmarked for degree'
VA	verb	'main' verb
VE	verb	'medial' verb
XA		abbreviations
XF		foreign words
XP	punct	Punctuation marks
XR		sequences of number and letters are tagged
XS		symbols
XX		text errors

A.13 Danish

POS Tag	Seed	Explanation
AC—U=–	noun	Cardinal Number (Adj)
AC	noun	
AN		
ANA—=-R		Adj
ANA[CN][SP]U=DU		Adj
ANC—=-R		Adj
ANC[CN]PU=[DI]U		Adj
ANC[CN]SU=IU		Adj
ANC[CN][SP]G=[DI]U		Adj
ANC[CN][SP]U=[DI]U		Adj
ANP—=-R		Adj
ANPCSU=IU		Adj
ANPCSU=[DI]U		Adj
ANPNSU=IU		Adj
ANPNSU=[DI]U		Adj
ANP[CN]PG=[DI]U		Adj
ANP[CN]PU=[DI]U		Adj
ANP[CN]SG=DU		Adj
ANP[CN]SU=DU		Adj
ANP[CN]SU=IU		Adj
ANP[CN]SU=[DI]U		Adj
ANP[CN][SP]U=[DI]U		Adj
ANS—=-R		Adj
ANS[CN]PU=DU		Adj
ANS[CN]PU=[DI]U		Adj
ANS[CN]SU=DU		Adj
ANS[CN]SU=IU		Adj
ANS[CN][SP]U=DU		Adj
ANS[CN][SP]U=[DI]U		Adj
AO—U=–	noun	Cardinal Number (Adj)
AO	noun	
CC	conj	Coordinating Conj
CS		Subordinating Conj
I=		Interjection
I		
NC	noun	
NCCPG==D	noun	Noun
NCCPG==I	noun	Noun
NCCPU==D	noun	Noun
NCCPU==I	noun	Noun
NCCPU==[DI]	noun	Noun
NCCSG==D	noun	Noun
NCCSG==I	noun	Noun
NCCSU==D	noun	Noun
NCCSU==I	noun	Noun
NCNPG==D	noun	Noun
NCNPG==I	noun	Noun

A.13 cont'd

POS Tag	UPOS	Explanation
NCNPU==D	noun	Noun
NCNPU==I	noun	Noun
NCNSG==D	noun	Noun
NCNSG==I	noun	Noun
NCNSU==D	noun	Noun
NCNSU==I	noun	Noun
NC[CN]PU==D	noun	Noun
NC[CN]PU==I	noun	Noun
NC[CN]SU==I	noun	Noun
NC[CN][SP]G==[DI]	noun	Noun
NC[CN][SP]U==I	noun	Noun
NC[CN][SP]U==[DI]	noun	Noun
NP	noun	
NP–G==-	noun	Noun
NP–U==-	noun	Noun
PC–PG—	noun	Pronoun
PC–PU—	noun	Pronoun
PC	noun	
PD-CSG–U	noun	Pronoun
PD-CSU–U	noun	Demonstrative Pronoun
PD-NSU–U	noun	Demonstrative Pronoun
PD-[CN]PU–U	noun	Pronoun
PD-[CN][SP]U–U	noun	Pronoun
PD	noun	
PI-CSG–U	noun	Pronoun
PI-CSU–U	noun	Indefinite Pronoun
PI-C[SP]N–U	noun	Pronoun
PI-NSU–U	noun	Indefinite Pronoun
PI-[CN]PG–U	noun	Pronoun
PI-[CN]PU–O	noun	Pronoun
PI-[CN]PU–U	noun	Pronoun
PI	noun	
PO1CSUPNF	noun	Pronoun
PO1CSUSNU	noun	Pronoun
PO1NSUPNF	noun	Pronoun
PO1NSUSNU	noun	Pronoun
PO1[CN]PUPNF	noun	Pronoun
PO1[CN]PUSNU	noun	Pronoun
PO1[CN][SP]UPNU	noun	Pronoun
PO2CSUSNU	noun	Pronoun
PO2NSUSNU	noun	Pronoun
PO2[CN]PUSNU	noun	Pronoun
PO2[CN][SP]UPNU	noun	Pronoun
PO2[CN][SP]U[SP]NP	noun	Pronoun
PO3CSUSYU	noun	Pronoun
PO3NSUSYU	noun	Pronoun
PO3[CN]PUSYU	noun	Pronoun
PO3[CN][SP]UPNU	noun	Pronoun
PO3[CN][SP]USNU	noun	Pronoun

A.13 cont'd

POS Tag	UPOS	Explanation
PO	noun	
PP1CPN-NU	noun	Pronoun
PP1CPU-[YN]U	noun	Pronoun
PP1CSN-NU	noun	Pronoun
PP1CSU-[YN]U	noun	Pronoun
PP2CPN-NU	noun	Pronoun
PP2CPU-[YN]U	noun	Pronoun
PP2CSN-NU	noun	Pronoun
PP2CSU-[YN]U	noun	Pronoun
PP2C[SP]N-NP	noun	Pronoun
PP2C[SP]U-[YN]P	noun	Pronoun
PP3CSN-NU	noun	Pronoun
PP3CSU-NU	noun	Pronoun
PP3NSU-NU	noun	Pronoun
PP3[CN]PN-NU	noun	Pronoun
PP3[CN]PU-NU	noun	Pronoun
PP3[CN][SP]U-YU	noun	Pronoun
PP	noun	
PT-CSU–U	noun	Pronoun
PT-C[SP]U–U	noun	Pronoun
PT-NSU–U	noun	Pronoun
PT-[CN]PU–U	noun	Pronoun
PT-[CN]SU–U	noun	Pronoun
PT-[CN][SP]G–U	noun	Pronoun
PT	noun	
RGA		Adverb
RGC		Adverb
RGP		Adverb
RGS		Adverb
RGU		Adverb
RG		
SP		Preposition
U=		Unique
U		
VADA=—-A-	verb	Indicative (Verb)
VADA=—-P-	verb	Indicative (Verb)
VADR=—-A-	verb	Indicative (Verb)
VADR=—-P-	verb	Indicative (Verb)
VAF-=—-A-	verb	Infinitive (Verb)
VAF-=—-P-	verb	Infinitive (Verb)
VAG-=SCI–U	verb	Gerund (Verb)
VAM-=———	verb	Imperative (Verb)
VAPA=P[CN][DI]A-G	verb	preterite participle (Verb)
VAPA=P[CN][DI]A-U	verb	preterite participle (Verb)
VAPA=SCDA-U	verb	preterite participle (Verb)
VAPA=S[CN]DA-U	verb	preterite participle (Verb)
VAPA=S[CN]IA-U	verb	preterite participle (Verb)
VAPA=S[CN]I[ARU]-U	verb	preterite participle (Verb)
VAPR=—R–	verb	present participle (Verb)

A.13 cont'd

POS Tag	UPOS	Explanation
VAPR=[SP][CN][DI]A-U	verb	present participle (Verb)
VAPR=[SP][CN][DI][ARU]-U	verb	present participle (Verb)
VA	verb	
VEDA=—-A-	verb	Indicative (Verb)
VEDR=—-A-	verb	Indicative (Verb)
VEF-=—-A-	verb	Infinitive (Verb)
VEPA=[SP][CN][DI][ARU]-U	verb	preterite participle (Verb)
VE	verb	
XA		Abbreviation
XF		Foreign Word
XP	punc	Residual Quote
XR		Residual
XS		Symbol
XX		Other

A.14 Dutch (coarse)

POS Tag	Seed	Explanation
Adj		Adjective
Adv		Adverb
Art	noun	Article / Determiner
Conj	conj	Conjunction
Int	noun	Interjection
MWU		Multiword unit
Misc		
N	noun	Noun
Num	noun	Number
Prep		Preposition
Pron	noun	Pronoun
Punc	punct	Punctuation
V	verb	Verb

A.15 Dutch

POS Tag	Seed	Explanation
Adj		Adjective
Adj_Adj		
Adj_Adj_N		
Adj_Adj_N_Adj_N_N		
Adj_Adj_N_N		
Adj_Adv		
Adj_Art		
Adj_Conj_V		
Adj_Int		
Adj_Misc_Misc		
Adj_N		
Adj_N_Conj_N		
Adj_N_N		
Adj_N_N_N		
Adj_N_N_N_N		
Adj_N_N_N_N_N		
Adj_N_Num		
Adj_N_Prep_Art_Adj_N		
Adj_N_Prep_Art_N		
Adj_N_Prep_N		
Adj_N_Prep_N_Conj_N		
Adj_N_Prep_N_N		
Adj_N_Punc		
Adj_Num		
Adj_Prep		
Adj_V		
Adj_V_Conj_V		
Adj_V_N		
Adv		Adverb
Adv_Adj		
Adv_Adj_Conj		
Adv_Adv		
Adv_Adv_Conj_Adv		
Adv_Art		
Adv_Conj		
Adv_Conj_Adv		
Adv_Conj_N		
Adv_N		
Adv_Num		
Adv_Prep		
Adv_Prep_N		
Adv_Prep_Pron		
Adv_Pron		
Adv_V		
Art	noun	Article / Determiner
Art_Adj		
Art_Adj_N		

A.15 cont'd

POS Tag	UPOS	Explanation
Art_Adj_N_Prep_Art_N_Conj_-V_N		
Art_Adv		
Art_Conj_Pron		
Art_N		
Art_N_Conj		
Art_N_Conj_Art_N		
Art_N_Conj_Art_V		
Art_N_Conj_Pron_N		
Art_N_N		
Art_N_Prep_Adj		
Art_N_Prep_Art_N		
Art_N_Prep_N		
Art_N_Prep_Pron_N		
Art_Num		
Art_Num_Art_Adj		
Art_Num_N		
Art_Pron		
Art_Pron_N		
Art_V_N		
Conj	conj	Conjnction
Conj_Adj		
Conj_Adv		
Conj_Adv_Adv		
Conj_Art_N		
Conj_Conj		
Conj_Int		
Conj_N		
Conj_N_Adv		
Conj_N_Prep		
Conj_Pron		
Conj_Pron_Adv		
Conj_Pron_V		
Conj_Punc_Conj	conj	
Conj_V		
Int		Interjection
Int_Adv		
Int_Int		
Int_N_N_Misc_N		
Int_N_Punc_Int_N		
Int_Punc_Int		
Misc		
Misc_Misc		
Misc_Misc_Misc		
Misc_Misc_Misc_Misc		
Misc_Misc_Misc_Misc_Misc_-Misc		
Misc_Misc_Misc_Misc_Misc_-Misc_Misc		

A.15 cont'd

POS Tag	UPOS	Explanation
Misc_Misc_Misc_Misc_Misc_-		
Misc_Misc_Misc_Misc		
Misc_Misc_Misc_Misc_Misc_-		
Misc_Punc_Misc_Misc_Misc		
Misc_Misc_Misc_Misc_Misc_N_-		
Misc_Misc_Misc_Misc_Misc_-		
Misc		
Misc_Misc_Misc_N		
Misc_Misc_N		
Misc_Misc_N_N		
Misc_Misc_Punc_N_N		
Misc_N		
Misc_N_Misc_Misc		
Misc_N_N		
N	noun	Noun
N_Adj	noun	
N_Adj_N	noun	
N_Adj_N_Num	noun	
N_Adv	noun	
N_Adv_Punc_V_Pron_V	noun	
N_Art_Adj_Prep_N	noun	
N_Art_N	noun	
N_Conj	noun	
N_Conj_Adv	noun	
N_Conj_Art_N	noun	
N_Conj_N	noun	
N_Conj_N_N	noun	
N_Int_N	noun	
N_Misc	noun	
N_Misc_Misc	noun	
N_Misc_Misc_Misc_Misc	noun	
N_Misc_Misc_N	noun	
N_Misc_N	noun	
N_Misc_N_N	noun	
N_Misc_N_N_N_N	noun	
N_Misc_Num	noun	
N_N	noun	
N_N_Adj	noun	
N_N_Adj_Art_N_N	noun	
N_N_Adj_N	noun	
N_N_Adv	noun	
N_N_Art_Adv	noun	
N_N_Art_N	noun	
N_N_Conj	noun	
N_N_Conj_N	noun	
N_N_Conj_N_N	noun	
N_N_Conj_N_N_N_N_N	noun	
N_N_Int_N_N	noun	

A.15 cont'd

POS Tag	UPOS	Explanation
N_N_Misc	noun	
N_N_Misc_Misc_Misc	noun	
N_N_N	noun	
N_N_N_Adj_N	noun	
N_N_N_Adv	noun	
N_N_N_Conj_N	noun	
N_N_N_Int	noun	
N_N_N_Misc	noun	
N_N_N_N	noun	
N_N_N_N_Conj_N	noun	
N_N_N_N_Misc	noun	
N_N_N_N_N	noun	
N_N_N_N_N_N	noun	
N_N_N_N_N_N_Int	noun	
N_N_N_N_N_N_N	noun	
N_N_N_N_N_N_Prep_N	noun	
N_N_N_N_N_Prep_N	noun	
N_N_N_N_Prep_N	noun	
N_N_N_N_Punc_N_Punc	noun	
N_N_N_N_V	noun	
N_N_N_Prep_Art_Adj_N	noun	
N_N_N_Prep_N	noun	
N_N_N_Prep_N_N	noun	
N_N_N_Punc	noun	
N_N_N_Punc_N	noun	
N_N_Num	noun	
N_N_Num_N	noun	
N_N_Prep_Art_Adj_N	noun	
N_N_Prep_Art_N	noun	
N_N_Prep_Art_N_Prep_Art_N	noun	
N_N_Prep_N	noun	
N_N_Prep_N_N	noun	
N_N_Prep_N_Prep_Adj_N	noun	
N_N_Punc_N_Punc	noun	
N_Num	noun	
N_Num_N	noun	
N_Num_N_N	noun	
N_Num_N_Num	noun	
N_Num_Num	noun	
N_Prep	noun	
N_Prep_Adj_Adj_N	noun	
N_Prep_Adj_N	noun	
N_Prep_Art_N	noun	
N_Prep_Art_N_Art_N	noun	
N_Prep_Art_N_N	noun	
N_Prep_Art_N_Prep_Art_N	noun	
N_Prep_N	noun	
N_Prep_N_Art_Adj	noun	
N_Prep_N_N	noun	

A.15 cont'd

POS Tag	UPOS	Explanation
N_Prep_N_Prep_Art_N	noun	
N_Prep_N_Prep_N_Conj_N_-Prep_Art_N_N	noun	
N_Prep_N_Punc_N_Conj_N	noun	
N_Prep_Num	noun	
N_Prep_Pron_N	noun	
N_Pron	noun	
N_Punc_Adj_N	noun	
N_Punc_Adj_Pron_Punc	noun	
N_Punc_Adv_V_Pron_N	noun	
N_Punc_Misc_Punc_N	noun	
N_Punc_N	noun	
N_Punc_N_Conj_N	noun	
N_Punc_N_N_N_N	noun	
N_Punc_N_Punc	noun	
N_Punc_N_Punc_N	noun	
N_Punc_Punc_N_N_Punc_-Punc_N	noun	
N_V	noun	
N_V_N	noun	
N_V_N_N	noun	
Num	noun	Number
Num_Adj	noun	
Num_Adj_Adj_N	noun	
Num_Adj_N	noun	
Num_Conj_Adj	noun	
Num_Conj_Art_Adj	noun	
Num_Conj_Num	noun	
Num_Conj_Num_N	noun	
Num_N	noun	
Num_N_N	noun	
Num_N_Num	noun	
Num_N_Num_Num_N	noun	
Num_Num	noun	
Num_Num_N	noun	
Num_Prep_Num	noun	
Num_Punc	noun	
Num_Punc_Num	noun	
Num_Punc_Num_N_N	noun	
Prep		Preposition
Prep_Adj		
Prep_Adj_Conj_Prep_N		
Prep_Adj_N		
Prep_Adv		
Prep_Art		
Prep_Art_Adj		
Prep_Art_Adj_N		
Prep_Art_Misc_Misc		

A.15 cont'd

POS Tag	UPOS	Explanation
Prep_Art_N		
Prep_Art_N_Adv		
Prep_Art_N_Art_N		
Prep_Art_N_Prep		
Prep_Art_N_Prep_Art_N		
Prep_Art_N_V		
Prep_Art_V		
Prep_Conj_Prep		
Prep_Misc		
Prep_N		
Prep_N_Adv		
Prep_N_Conj		
Prep_N_Conj_N		
Prep_N_N		
Prep_N_Prep		
Prep_N_Prep_N		
Prep_N_V		
Prep_Num		
Prep_Num_N		
Prep_Prep		
Prep_Prep_Adj		
Prep_Prep_Adv		
Prep_Prep_Art_N		
Prep_Pron		
Prep_Pron_Adj		
Prep_Pron_N		
Prep_Pron_N_Adv		
Prep_Punc_N_Conj_N		
Prep_V		
Prep_V_N		
Prep_V_Pron_Pron_Adv		
Pron	noun	Pronoun
Pron_Adj	noun	
Pron_Adj_N_Punc_Art_Adj_N_-	noun	
Prep_Art_Adj_N		
Pron_Adv	noun	
Pron_Art	noun	
Pron_Art_N_N	noun	
Pron_N	noun	
Pron_N_Adv	noun	
Pron_N_V_Adv_Num_Punc	noun	
Pron_N_V_Conj_N	noun	
Pron_Prep	noun	
Pron_Prep_Art	noun	
Pron_Prep_N	noun	
Pron_Prep_Pron	noun	
Pron_Pron	noun	
Pron_Pron_V	noun	

A.15 cont'd

POS Tag	UPOS	Explanation
Pron_V	noun	
Pron_V_V	noun	
Punc	punct	Punctuation
Punc_Int_Punc_N_N_N_Punc_-		
Pron_V_Pron_Adj_V_Punc		
Punc_N_Punc_N		
Punc_Num		
Punc_Num_Num		
V	verb	Verb
V_Adj_N		
V_Adv		
V_Adv_Art_N_Prep_Pron_N		
V_Art_N		
V_Art_N_Num_N		
V_Conj_N_N		
V_Conj_Pron		
V_N		
V_N_Conj_Adj_N_Prep_Art_N		
V_N_Misc_Punc		
V_N_N		
V_N_V		
V_Prep		
V_Pron		
V_Pron_Adv		
V_Pron_Adv_Adv_Pron_V		
V_Pron_V		
V_V	verb	

A.16 English (Coarse)

POS Tag	Seed	Explanation
WD		Wh-determiner
FW	noun	Foreign word
WR		Wh-adverb
JJ		Adjective
WP		Wh-pronoun
DT	noun	Determiner
PR	noun	Pronoun
RP		Particle
NN	noun	Noun
TO		To
LS		List
RB		
PO		Possessive Engind?
VB	verb	verb
PD		Predeterminer
CC	conj	Coordinating conjunction
CD	noun	Numeral
EX	noun	Existential there
IN		Preposition or subordinating conjunction
MD	verb	Model
SY		Symbol
UH	noun	Interjection

A.17 English

POS Tag	Seed	Explanation
,	punct conj	Comma
;	punct conj	Semi-Colon
CC	conj	Coordinating conjunction
CD	noun	Cardinal number
DT	noun	Determiner
EX	noun	Existential there
FW	noun	Foreign word
IN		Preposition or subordinating conjunction
JJ		Adjective
JJR		Adjective, comparative
JJS		Adjective, superlative
LS		List item marker
MD	verb	Modal
NN	noun	Noun, singular or mass
NNS	noun	Noun, plural
NNP	noun	Proper noun, singular
NNPS	noun	Proper noun, plural
PDT		Predeterminer
POS		Possessive ending
PRP	noun	Personal pronoun
PRP$	noun	Possessive pronoun
RB		Adverb
RBR		Adverb, comparative
RBS		Adverb, superlative
RP		Particle
SYM		Symbol
TO		to
UH	noun	Interjection
VB	verb	Verb, base form
VBD	verb	Verb, past tense
VBG	verb	Verb, gerund or present participle
VBN	verb	Verb, past participle
VBP	verb	Verb, non-3rd person singular present
VBZ	verb	Verb, 3rd person singular present
WDT		Wh-determiner
WP		Wh-pronoun
WP$		Possessive wh-pronoun
WRB		Wh-adverb

A.18 Japanese

POS Tag	Seed	Explanation
–	verb noun	Not assigned (Nicht zugeordnet) - disfluencies
ADJ		Atributive adjective
ADJ_n		na-adjective
ADJdem		Demonstrative adverb
ADJicnd		i-adjective (conditional)
ADJifin		i-adjective (finite)
ADJiku		i-adjective (-ku ending)
ADJite		i-adjective (-te ending)
ADJsf		Adjective suffix
ADJteki		na-adjective (-teki ending)
ADJwh		Wh Adjective
ADV		ADVerbials in general
ADVdem		Demonstrative adverb
ADVdgr		Degree adverb
ADVtmp		Temporal adverb
ADVwh		Wh adverb
CD	noun	Cardinal number
CDU	noun	Cardinal unit
CDdate	noun	Cardinal date unit
CDtime	noun	Cardinal time unit
CNJ		Conjunction
GR	noun	Greeting (noun?)
ITJ	noun	Interjection (noun?)
NAME	noun	Other proper noun
NAMEloc	noun	Proper noun; location
NAMEorg	noun	Proper noun; organization
NAMEper	noun	Proper noun; person
NF	noun	Formal noun
NN	noun	Common noun
NT	noun	
Ndem	noun	Demonstarative noun
Nsf	noun	Noun sufix
Ntmp	noun	Noune (temporal)
Nwh	noun	Wh noun
P		Postposition
PADJ		Particle adjective
PADV		Particle adverb
PNsf		Personal name suffix
PQ		Quotative postposition
PRON	noun	Pronoun
PSE		Sentence end postposition
PSSa	verb	Subordinate S postposition (and)
PSSb	verb	Subordinate S postposition (but)
PSSq		Subordinate S postposition (question)
PV	verb	Particle verb
PVcnd	verb	Particle verb (conditional)
PVfin	verb	Particle verb (finite)
PVte	verb	Particle verb (-te ending)

A.18 cont'd

POS Tag	UPOS	Explanation
Pacc		Accusative case
Pcnj	conj	Conjunctive particle
Pfoc		Focus
Pgen		Genitive case
Pnom		Nominative case
PreN		Noun prefix
UNIT	noun	Unit
V	verb	Verb (other forms)
VADJ_n	verb	Verb na-adjective
VADJi	verb	Verb i-adjective
VADJicnd	verb	Verb i-adjective (conditional)
VAUX	verb	Auxiliary verb
VAUXbas	verb	Auxiliary verb (base)
VAUXcnd	verb	Auxiliary verb (conditional)
VAUXfin	verb	Auxiliary verb (finite)
VAUXimp	verb	
VAUXte	verb	Auxiliary verb (-te ending)
VN	noun	Verbal noun
VS	verb	Support verb
VSbas	verb	Support verb (base)
VScnd	verb	Support verb (conditional)
VSfin	verb	Support verb (finite)
VSimp	verb	Support verb (imperative)
VSte	verb	Support verb (-te ending)
Vbas	verb	Verb (base)
Vcnd	verb	Verb (conditional)
Vfin	verb	Verb (finite)
Vimp	verb	Verb (imperative)
Vte	verb	Verb (-te/de ending)
xxx		Tokenizing problem

A.19 Portuguese (Coarse)

POS Tag	Seed	Explanation
adv	adverb	
pp	noun	prepositional phrase
art	noun	article
in	noun	interjection
intj	noun	interjection
ec		
n	noun	noun
vp	verb	verb phrase
pron	noun	pronoun
num	noun	numeral
prp		preposition
v	verb	verb
punc	punct	punctuation
conj	conj	conjunction
prop	noun	proper noun
adj		adjective

A.20 Portuguese

POS Tag	Seed	Explanation
art		article
v-inf	verb	infinitive
v-ger	verb	gerund
ec		
vp	verb	verb phrase (missing in train 2)
num	noun	numeral
prp		preposition
in	noun	interjection
intj	noun	interjection (train 2)
adv		adverb
pp		prepositional phrase
pu		(train 2)
prop	noun	proper noun
v-fin	verb	finite verb
adj		adjective
?	punct	(missing in train 2)
conj-c	conj	coordinating conjunction
conj-s		suordination conjunction
pron-pers	noun	personal pronoun
punc	punct	
pron-det	noun	determiner pronoun
n-adj		(train 2)
v-pcp		participle
pron-indp	noun	independent pronoun
n	noun	noun

A.21 Slovene (Coarse)

POS Tag	Seed	Explanation
Adjective Noun	noun	
Pronoun	noun	
Adverb		
Abbreviation		
Residual		
Particle		
Preposition		
Verb	verb	
Numeral	noun	
PUNC	punct	
Conjunction	conj	
Interjection	noun	

A.22 Slovene

POS Tag	Seed	Explanation
Pronoun-general	noun	
Residual		
Numeral-ordinal	noun	
Pronoun-personal	noun	
Adjective-general		
Adverb-general		
PUNC	punct	
Adjective-participle		
Adverb-participle		
Pronoun-reflexive	noun	
Interjection	noun	
Conjunction-coordinating	conj	
Numeral-special	noun	
Numeral-cardinal	noun	
Particle		
Noun-common	noun	
Pronoun-indefinite	noun	
Pronoun-interrogative	noun	
Adjective-possessive		
Adjective-ordinal		
Adjective-qualificative		
Adposition-preposition		
Pronoun-possessive	noun	
Noun-proper	noun	
Pronoun-demonstrative	noun	
Verb-main	verb	
Pronoun-negative	noun	
Conjunction-subordinating		
Residual-foreign		
Pronoun-relative	noun	
Abbreviation		
Preposition		
Verb-auxiliary	verb	
Verb-copula	verb	
Verb-modal	verb	
Numeral-pronominal	noun	
Numeral-multiple	noun	

A.23 Spanish

POS Tag	Seed	Explanation
Fa	punct	Punctuation
Fc	punct	Punctuation
Fd	punct	Punctuation
Fe	punct	Punctuation
Fg	punct	Punctuation
Fh	punct	Punctuation
Fi	punct	Punctuation
Fp	punct	Punctuation
Fs	punct	Punctuation
Fx	punct	Punctuation
Fz	punct	Punctuation
X		
Y		Abbreviation
Zm	noun	number
Zp	noun	number
ao		adjective ordinal
aq		adjective qualifying
cc	conj	Coordinating Conjunction
cs		Subordinating Conjunction
da	noun	Determiner article
dd	noun	Determiner demonstrative numeral
de	noun	Determiner
di	noun	Determiner indefinite
dn	noun	Determiner
dp	noun	Determiner
dt	noun	Determiner
i	noun	Interjection
nc	noun	noun common
np	noun	noun proper
p0	noun	Pronoun
pd	noun	Pronoun demonsrtative
pe	noun	Pronoun
pi	noun	Pronoun indefinite
pn	noun	Pronoun numeral
pp	noun	Pronoun
pr	noun	Pronoun relative interrogative
pt	noun	Pronoun
px	noun	Pronoun
rg		adverb general
rn		adverb negative
sn		Preposition
sp		Preposition
va	verb	verb auxiliary
vm	verb	verb main
vs	verb	verb semiauxiliary
w	noun	Date
z	noun	Number

A.24 Swedish

POS Tag	Seed	Explanation
SP	verb	Present participle – Present participle
FV	verb	The verb "f" (get)
HV	verb	The verb "ha(va)" (have)
YY		Interjection
WV	verb	The verb "vilja" (want)
BV	verb	The verb "bli(va)" (become)
AB		Adverb
PR		Preposition
NN	noun	Other noun
PU	punct	Pause – List item (bullet or number) : Punct
TP	noun	Totality pronoun – Perfect participle
RO	noun	Numeral other than "en", "ett" (one)
PN	noun	Proper name
PO	noun	Pronoun
IR	punct	Parenthesis
GV	verb	The verb "gra" (do, make)
EN		Indefinite article or numeral "en", "ett" (one)
IQ	punct	Colon
IP	punct	Period
IS	punct	Semicolon
AJ		Adjective
IU	punct	Exclamation mark
IT	punct	Dash
VN	noun	Verbal noun
AN	noun	Adjectival noun
IK	punct	Comma
IM		Infinitive marker
VV	verb	Other verb
AV	verb	The verb "vara" (be)
IC	punct	Quotation mark
ID		Part of idiom (multi-word unit)
IG	punct	Other punctuation mark
I?	punct	Question mark
QV	verb	The verb "kunna" (can)
++	conj	Coordination conjunction
XX		Unclassified POS
MN		Adversative – Meta-noun
SV	verb	The verb "skola" (will,shall)
MV	verb	The verb "mste" (must)
KV	verb	The verb locution "komma att" (periphrastic future)
UK		Subordinating conjunction

A.25 German (Tiger)

POS Tag	Seed	Explanation
ADJA		Adjective, attributive
ADJD		adjective, adverbial or predicative
ADV		adverb
APPO		postposition
APPR		preposition; circumposition left
APPRART		preposition with article
APZR		circumposition right
ART	noun	definite or indefinite article
CARD	noun	cardinal number
FM		foreign language material
ITJ		interjection
KOKOM		comparative conjunction
KON	conj	coordinate conjunction
KOUI		subordinate conjunction with zu and infinitive
KOUS		subordinate conjunction with sentence
NE	noun	proper noun
NN	noun	common noun
NNE	noun	
PAV	noun	pronominal adverb
PDAT	noun	attributive demonstrative pronoun
PDS	noun	substituting demonstrative pronoun
PIAT	noun	attributive indefinite pronoun without determiner
PIDAT	noun	attributive indefinite pronoun with determiner
PIS	noun	substituting indefinite pronoun
PPER	noun	non-reflexive personal pronoun
PPOSAT	noun	attributive possessive pronoun
PPOSS	noun	substituting possessive pronoun
PRELAT	noun	attributive relative pronoun
PRELS	noun	substituting relative pronoun
PRF	noun	reflexive personal pronoun
PROAV	noun	
PTKA		particle with adjective or adverb
PTKANT		answer particle
PTKNEG		negative particle
PTKVZ		separable verbal particle
PTKZU		zu before infinitive
PWAT	noun	attributive interrogative pronoun
PWAV	noun	adverbial interrogative or relative pronoun
PWS	noun	substituting interrogative pronoun
SGML		SGML markup
SPELL		letter sequence
TRUNC		word remnant
VAFIN	verb	finite verb, auxiliary
VAIMP	verb	imperative, auxiliary
VAINF	verb	infinitive, auxiliary
VAPP	verb	perfect participle, auxiliary
VMFIN	verb	finite verb, modal
VMINF	verb	infinitive, modal

A.25 cont'd

POS Tag	UPOS	Explanation
VMPP	verb	
VVFIN	verb	finite verb, full
VVIMP	verb	imperative, full
VVINF	verb	infinitive, full
VVIZU	verb	Infinitive with zu, full
VVPP	verb	perfect participle, full
XY		non-word containing non-letter

References

[1] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a Large Annotated Corpus of English: The Penn Treebank," *Computational Linguistics*, vol. 19, pp. 313–330, June 1993.

[2] S. Petrov, D. Das, and R. McDonald, "A Universal Part-of-Speech Tagset," in *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*, Istanbul, Turkey, May 2012, pp. 2089–2096.

[3] B. Santorini, "Part-of-Speech Tagging Guidelines for the Penn Treebank Project (3rd Revision)," University of Pennsylvania, Tech. Rep. 570, 1990.

[4] D. H. Younger, "Recognition and parsing of context-free languages in time n3," *Information and Control*, vol. 10, no. 2, pp. 189–208, Feb. 1967.

[5] T. Kasami, "An Efficient Recognition and Syntaxanalysis Algorithm for Context-Free Languages." Air Force Cambridge Research Laboratory, Bedford, MA, Tech. Rep. AFCRL-65-758, July 1965.

[6] W. N. Francis, "A standard sample of present-day english for use with digital computers," Brown University, Providence, RI, Tech. Rep., 1964.

[7] W. N. Francis and H. Kučera, *Frequency Analysis of English Usage: Lexicon and Grammar*. Houston Mifflin, 1982.

[8] J. Svartvik and R. Quirk, *A Corpus of English Conversation*. Lund: CWK Gleerup, 1980.

[9] R. Garside, G. Leech, and G. Sampson, *The Computational Analysis of English: A Corpus-Based Approach*. London: Longman, 1987.

[10] S. Johansson, G. Leech, and H. Goodluck, "Manual of information to accompany the Lancaster-Oslo/Bergen Corpus of British English, for use with digital computers," Department of English, University of Oslo, Tech. Rep., 1978.

[11] L. Tesnière, *Éléments de Syntaxe Structurale*. Paris, France: Klinsieck, 1959.

[12] I. Mel'cuk, *Dependency syntax: theory and practice*. Albany, NY: State University of New York Press, 1988.

[13] S. Kubler, J. Nivre, and R. McDonald, *Dependency parsing*, synthesis lectures on human language technologies ed. Synthesis Lectures on Human Language Technologies, 2009, vol. 1.

[14] A. M. Zwicky, "Heads," *Journal of Linguistics*, vol. 21, pp. 1 – 29, 1985.

[15] N. Chomsky, "Three models for the description of language," *Information Theory, IRE Transactions on*, vol. 2, no. 3, pp. 113–124, Sep. 1956.

[16] N. Chomsky, "On certain formal properties of grammars," *Information and Control*, vol. 2, no. 2, pp. 137–167, June 1959.

[17] S. Shieber, "Evidence against the context-freeness of natural language," *Linguistics and Philosophy*, vol. 8, no. 3, pp. 333–343, 1985.

[18] A. K. Joshi, "Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions?" in *Natural language parsing*, D. R. Dowty, L. Karttunen, and A. M. Zwicky, Eds. Cambridge University Press, 1985, pp. 206–250, cambridge Books Online.

[19] K. Vijay-Shanker and D. J. Weir, "Polynomial Time Parsing of Combinatory Categorial Grammars," in *Proceedings of the the Annual Meeting of the Association for Computational Linguistics*, 1990, pp. 1–8.

[20] K. Vijay-Shanker and D. J. Weir, "Parsing Some Constrained Grammar Formalisms," *Computational Linguistics*, vol. 19, pp. 592–636, 1994.

[21] K. Vijay-Shanker and D. J. Weir, "The Equivalence Of Four Extensions Of Context-Free Grammars," *Mathematical Systems Theory*, vol. 27, pp. 511–546, 1994.

[22] T. Booth and R. Thompson, "Applying probability measures to abstract languages," *IEEE Transactions on Computers*, pp. 442–450, 1973.

[23] S. Petrov, "Coarse-to-Fine Natural Language Processing," Ph.D. dissertation, University of California, Berkley, 2009.

[24] C. J. Van Rijsbergen, *Information Retrieval, second edition*. London: Butterworths, 1979.

[25] E. Black, S. Abney, D. Flickinger, C. G. R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski, "A procedure for quantitatively comparing the syntactic coverage of english grammars," in *Proceedings of the DARPA Speech and Natural Language Workshop*, San Mateo, CA, 1991, pp. 306 – 311.

[26] G. Carroll and E. Charniak, "Two experiments on learning probabilistic dependency grammars from corpora," *Association for the Advancement of Artificial Intelligence*, p. 1–13, 1992.

[27] C. G. de Marcken, "Unsupervised Language Acquisition," Ph.D. dissertation, Massachusetts Institute of Technology, Sep. 1996.

[28] D. Yuret, "Discovery of Linguistic Relations Using Lexical Attraction," Ph.D. dissertation, Massachusetts Institute of Technology, 1998.

[29] A. Clark, "Unsupervised Language Acquisition: Theory and Practice," Ph.D. dissertation, University of Sussex, 2001.

[30] M. A. Paskin, "Grammatical Bigrams," in *Advances in Neural Information Processing Systems*, 2001, pp. 1–7.

[31] D. Klein, "The Unsupervised Learning of Natural Language Structure," Ph.D. dissertation, Stanford University, 2005.

[32] N. A. Smith, "Novel estimation methods for unsupervised discovery of latent structure in natural language text," Ph.D. dissertation, Johns Hopkins University, Baltimore, MD, Oct. 2006. [Online]. Available: http://cs.jhu.edu/~jason/papers/#smith-2006

[33] S. B. Cohen and N. A. Smith, "Shared Logistic Normal Distributions for Soft Parameter Tying in Unsupervised Grammar Induction," in *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, Boulder, Colorado, June 2009, pp. 74–82.

[34] W. P. Headden III, M. Johnson, and D. McClosky, "Improving Unsupervised Dependency Parsing with Richer Contexts and Smoothing," in *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, Boulder, Colorado, June 2009, pp. 101–109.

[35] T. Berg-Kirkpatrick and D. Klein, "Phylogenetic Grammar Induction," in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, Uppsala, Sweden, July 2010, pp. 1288–1297.

[36] T. Cohn, P. Blunsom, and S. Goldwater, "Inducing Tree-Substitution Grammars," *The Journal of Machine Learning Research*, vol. 11, pp. 3053–3096, Nov. 2010.

[37] T. Naseem, H. Chen, R. Barzilay, and M. Johnson, "Using universal linguistic knowledge to guide grammar induction," in *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, Cambridge, MA, Oct. 2010, pp. 1234–1244.

[38] P. Boonkwan and M. Steedman, "Grammar Induction from Text Using Small Syntactic Prototypes," in *Proceedings of 5th International Joint Conference on Natural Language Processing*, Chiang Mai, Thailand, Nov. 2011, pp. 438–446.

[39] V. I. Spitkovsky, "Grammar Induction and Parsing with Dependency-And-Boundary Models," Ph.D. dissertation, Stanford University, Dec. 2013.

[40] C. Christodoulopoulos, "An Iterated Learning Framework for Unsupervised Part-of-Speech Induction," Ph.D. dissertation, University of Edinburgh, 2013.

[41] D. Klein and C. D. Manning, "Corpus-Based Induction of Syntactic Structure: Models of Dependency and Constituency," in *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, Barcelona, Spain, July 2004, pp. 478–485.

[42] P. Blunsom and T. Cohn, "Unsupervised Induction of Tree Substitution Grammars for Dependency Parsing," *Proceedings of the 2010 Conference on Empirical Methods of Natural Language Processing*, pp. 1204–1213, Oct. 2010.

[43] K. Tu, "Combining the Sparsity and Unambiguity Biases for Grammar Induction," in *NAACL HLT Workshop on Induction of Linguistic Structure*, Montréal, Canada, June 2012, pp. 105–110.

[44] Y. Huang, M. Zhang, and C. L. Tan, "Improved Combinatory Categorial Grammar Induction with Boundary Words and Bayesian Inference," in *Proceedings of the 24rd International Conference on Computational Linguistics (Coling 2012)*, Mumbai, India, Dec. 2012.

[45] V. I. Spitkovsky, H. Alshawi, and D. Jurafsky, "Breaking Out of Local Optima with Count Transforms and Model Recombination: A Study in Grammar Induction," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 2013, pp. 1983–1995.

[46] D. Mareček and M. Straka, "Stop-probability estimates computed on a large corpus improve Unsupervised Dependency Parsing," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2013.

[47] D. Mareček and Z. Žabokrtský, "Exploiting Reducibility in Unsupervised Dependency Parsing," in *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, Jeju Island, Korea, July 2012, pp. 297–307.

[48] K. Ganchev, J. Gillenwater, and B. Taskar, "Dependency grammar induction via bitext projection constraints," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. Suntec, Singapore: ACL, Aug. 2009, pp. 369–377.

[49] V. I. Spitkovsky, H. Alshawi, and D. Jurafsky, "From Baby Steps to Leapfrog: How "Less is More" in Unsupervised Dependency Parsing," in *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, Los Angeles, California, June 2010, pp. 751–759.

[50] V. I. Spitkovsky, H. Alshawi, D. Jurafsky, and C. D. Manning, "Viterbi training improves unsupervised dependency parsing," in *Conference on Computational Natural Language Learning*, July 2010, pp. 9–17.

[51] V. I. Spitkovsky, H. Alshawi, and D. Jurafsky, "Punctuation: making a point in unsupervised dependency parsing," in *Conference on Computational Natural Language Learning*. Association for Computational Linguistics, June 2011, pp. 19–28.

[52] D. Mareček and Z. Žabokrtský, "Dealing with Function Words in Unsupervised Dependency Parsing," *Computational Linguistics and Intelligent Text Processing*, vol. 8403, no. Chapter 20, pp. 250–261, 2014.

[53] M. Steedman, *The Syntactic Process*. The MIT Press, Sep. 2000.

[54] S. Clark, J. Hockenmaier, and M. Steedman, "Building Deep Dependency Structures using a Wide-Coverage CCG Parser," in *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002. [Online]. Available: http://www.aclweb.org/anthology/P02-1042 pp. 327–334.

[55] R. Schwartz, O. Abend, R. Reichart, and A. Rappoport, "Neutralizing Linguistically Problematic Annotations in Unsupervised Dependency Parsing Evaluation," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 663–672.

[56] D. M. Magerman, "Statistical decision-tree models for parsing," in *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*. Cambridge, Massachusetts, USA: Association for Computational Linguistics, June 1995, pp. 276–283.

[57] M. J. Collins, "A new statistical parser based on bigram lexical dependencies," in *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*. Santa Cruz, California, USA: Association for Computational Linguistics, June 1996, pp. 184–191.

[58] H. Yamada and Y. Matsumoto, "Statistical Dependency Analysis with Support Vector Machines," *International Workshop on Parsing Technologies*, 2003.

[59] R. Johansson and P. Nugues, "Extended Constituent-to-Dependency Conversion for English," in *NODALIDA 2007 Proceedings*, J. Nivre, H.-J. Kalep, K. Muischnek, and M. Koit, Eds. University of Tartu, 2007, pp. 105–112.

[60] D. Vadas and J. Curran, "Adding Noun Phrase Structure to the Penn Treebank," in *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*. Prague, Czech Republic: Association for Computational Linguistics, June 2007, pp. 240–247.

[61] J. Nivre, *Inductive Dependency Parsing*. Springer, 2006.

[62] J. Nivre, J. Hall, S. Kübler, R. McDonald, J. Nilsson, S. Riedel, and D. Yuret, "The CoNLL 2007 shared task on dependency parsing," in *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*. Prague, Czech Republic: Association for Computational Linguistics, June 2007. [Online]. Available: http://www.aclweb.org/anthology/D/D07/D07-1096 pp. 915–932.

[63] D. Gelling, T. Cohn, P. Blunsom, and J. V. Graca, "The PASCAL Challenge on Grammar Induction," in *Proceedings of the NAACL HLT Workshop on Induction of Linguistic Structure*, Montréal, Canada, June 2012, pp. 64–80.

[64] M. Collins, "Head-Driven Statistical Models for Natural Language Parsing," *Computational Linguistics*, vol. 29, no. 4, pp. 589–637, Dec. 2003.

[65] K. Ajdukiewicz, "Die syntaktische Konnexität," in *Polish Logic 1920-1939*, S. McCall, Ed. Oxford University Press, 1935, pp. 207–231, translated from *Studia Philosophica*, 1, 1-27.

[66] Y. Bar-Hillel, "A quasi-arithmetical notation for syntactic description," *Language*, vol. 29, pp. 47–58, 1953.

[67] A. E. Ades and M. J. Steedman, "On the order of words," *Linguistics and Philosophy*, vol. 4, no. 4, pp. 517–558, 1982.

[68] M. Steedman, *Surface structure and interpretation*. The MIT Press, Jan. 1996.

[69] H. B. Curry and R. Feys, *Combinatory Logic*. Amsterdam: North-Holland, 1958, vol. I.

[70] J. Hockenmaier and M. Steedman, "CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank," *Computational Linguistics*, vol. 33, pp. 355–396, Sep. 2007.

[71] J. Bos, "Wide-Coverage Semantic Analysis with Boxer," in *Semantics in Text Processing. STEP 2008 Conference Proceedings*, ser. Research in Computational Semantics, J. Bos and R. Delmonte, Eds. College Publications, 2008, vol. 1, pp. 277–286. [Online]. Available: http://www.aclweb.org/anthology/W08-2222

[72] S. Reddy, M. Lapata, and M. Steedman, "Large-scale Semantic Parsing without Question-Answer Pairs," *Transactions of the Association for Computational Linguistics*, pp. 1–16, June 2014.

[73] D. Gildea and J. Hockenmaier, "Identifying Semantic Roles Using Combinatory Categorial Grammar," in *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, 2003, pp. 57–64.

[74] A. Church, "An unsolvable problem of elementary number theory," *American Journal of Mathematics*, vol. 58, no. 2, pp. pp. 345–363, 1936. [Online]. Available: http://www.jstor.org/stable/2371045

[75] S. C. Kleene, "λ -definability and recursiveness," *Duke Math. J.*, vol. 2, no. 2, pp. 340–353, 06 1936.

[76] S. Kleene, "General recursive functions of natural numbers," *Mathematische Annalen*, vol. 112, no. 1, pp. 727–742, 1936.

[77] A. M. Turing, "Computability and -definability," *The Journal of Symbolic Logic*, vol. 2, no. 4, pp. pp. 153–163, 1937. [Online]. Available: http://www.jstor.org/stable/2268280

[78] Y. Artzi and L. S. Zettlemoyer, "Bootstrapping Semantic Parsers from Conversations," in *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, Edinburgh, Scotland, UK., July 2011.

[79] C. Matuszek, N. FitzGerald, L. S. Zettlemoyer, L. Bo, and D. Fox, "A Joint Model of Language and Perception for Grounded Attribute Learning," in *International conference on Machine learning*, June 2012.

[80] J. Krishnamurthy and T. M. Mitchell, "Weakly Supervised Training of Semantic Parsers," in *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Jeju Island, Korea: Carnegie Mellon University, July 2012, pp. 754–765.

[81] Y. Artzi and L. S. Zettlemoyer, "Weakly Supervised Learning of Semantic Parsers for Mapping Instructions to Actions," *Transactions of the Association for Computational Linguistics*, pp. 49–62, 2013.

[82] T. Kwiatkowski, E. Choi, Y. Artzi, and L. S. Zettlemoyer, "Scaling Semantic Parsers with On-the-Fly Ontology Matching," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1545–1556.

[83] Y. Artzi, D. Das, and S. Petrov, "Learning compact lexicons for ccg semantic parsing," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, October 2014, pp. 1273–1283.

[84] J. Hockenmaier, "Data and models for statistical parsing with Combinatory Categorial Grammar," Ph.D. dissertation, PhD Thesis, School of Informatics. Edinburgh, Jan. 2003.

[85] S. Clark and J. Curran, "Wide-Coverage Efficient Statistical Parsing with CCG and Log-Linear Models," *Computational Linguistics*, vol. 33, pp. 493–552, Mar. 2007.

[86] J. Hockenmaier and P. Young, "Non-local scrambling: the equivalence of TAG and CCG revisited," in *Workshop on Tree Adjoining Grammars and Related Formalisms*, Apr. 2008, p. 8.

[87] S. Clark, "Supertagging for Combinatory Categorial Grammar," in *Proceedings of the 6th International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+6)*, Venice, Italy, 2002, pp. 19 – 24.

[88] J. Eisner, "Efficient Normal-Form Parsing for Combinatory Categorial Grammar," in *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, Santa Cruz, California, USA, June 1996, pp. 79–86.

[89] J. Hockenmaier and Y. Bisk, "Normal-form parsing for Combinatory Categorial Grammars with generalized composition and type-raising," in *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, Beijing, China, Aug. 2010, pp. 465–473.

[90] W. Buszkowski and G. Penn, "Categorial Grammars Determined from Linguistic Data by Unification," *Studia Logica: An International Journal for Symbolic Logic*, vol. 49, no. 4, pp. 431–454, Jan. 1990.

[91] M. Osborne and T. Briscoe, "Learning Stochastic Categorial Grammars," in *Conference on Computational Natural Language Learning*, Jan. 1997.

[92] J. Hockenmaier, "Creating a CCGbank and a Wide-Coverage CCG Lexicon for German," in *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*. Sydney, Australia: Association for Computational Linguistics, July 2006, pp. 505–512.

[93] D. Tse, "Chinese CCGBank: Deep Derivations and Dependencies for Chinese CCG Parsing," Ph.D. dissertation, The University of Sydney, 2013.

[94] M. Honnibal, J. Nothman, and J. R. Curran, "Evaluating a Statistical CCG Parser on Wikipedia," in *Proceedings of the 2009 Workshop on The People's Web Meets NLP: Collaboratively Constructed Semantic Resources*, August 2009, pp. 38–41.

[95] S. A. Boxwell and C. Brew, "A Pilot Arabic CCGbank," in *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, may 2010.

[96] S. Uematsu, T. Matsuzaki, H. Hanaoka, Y. Miyao, and H. Mima, "Integrating Multiple Dependency Corpora for Inducing Wide-coverage Japanese CCG Resources," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Sofia, Bulgaria: Association for Computational Linguistics, Aug. 2013, pp. 1042–1051.

[97] B. R. Ambati, T. Deoskar, and M. Steedman, "Using CCG categories to improve Hindi dependency parsing," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, August 2013, pp. 604–609.

[98] J. Hockenmaier and M. Steedman, "Generative Models for Statistical Parsing with Combinatory Categorial Grammar," in *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, Pennsylvania, USA, July 2002, pp. 335–342.

[99] J. Hockenmaier, "Parsing with generative models of predicate-argument structure," in *Association for Computational Linguistics*, 2003, p. 366.

[100] M. Auli and A. Lopez, "A Comparison of Loopy Belief Propagation and Dual Decomposition for Integrated CCG Supertagging and Parsing," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Portland, Oregon, USA, June 2011.

[101] M. Lewis and M. Steedman, "Improved CCG Parsing with Semi-supervised Supertagging," *Transactions of the Association for Computational Linguistics*, vol. 2, no. 10, pp. 327–338, Oct. 2014.

[102] M. Lewis and M. Steedman, "A* ccg parsing with a supertag-factored model," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, October 2014, pp. 990–1000.

[103] D. Garrette, C. Dyer, J. Baldridge, and N. A. Smith, "Weakly-Supervised Grammar-Informed Bayesian CCG Parser Learning," in *Proceedings of the Association for the Advancement of Artificial Intelligence*, 2015.

[104] 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*. Uppsala, Sweden: Association for Computational Linguistics, July 2010, pp. 335–344.

[105] N. Chomsky, "Remarks on nominalization," *Reading in English Transformational Grammar*, pp. 184–221, 1970.

[106] B. Snyder, T. Naseem, and R. Barzilay, "Unsupervised Multilingual Grammar Induction," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, Suntec, Singapore, Aug. 2009, pp. 73–81.

[107] A. Haghighi and D. Klein, "Prototype-Driven Grammar Induction," in *Association for Computational Linguistics*. Morristown, NJ, USA: Association for Computational Linguistics, 2006, pp. 881–888.

[108] R. H. Robins, "Noun and verb in universal grammar," *Language*, vol. 28, no. 3, pp. 289–298, 1952.

[109] T. Givon, *On understanding grammar*. New York: Academic Press, 1979.

[110] P. Schachter, "Parts-of-speech systems," *Language typology and syntactic description. Vol. 1: Clause structure*, pp. 3–61, 1985.

[111] J. A. Hawkins, *Explaining language universals*. Cambridge, MA: Basil Blackwell, Inc, 1988.

[112] Y. Bisk and J. Hockenmaier, "Simple Robust Grammar Induction with Combinatory Categorial Grammars," in *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence (AAAI-12)*, Toronto, Canada, July 2012, pp. 1643–1649.

[113] K. Lari and S. J. Young, "Applications of stochastic context-free grammars using the Inside-Outside algorithm," *Computer speech & language*, vol. 5, no. 3, pp. 237–257, Jan. 1991.

[114] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, Jan. 1977.

[115] 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 *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*. Manchester, England: Coling 2008 Organizing Committee, August 2008, pp. 159–177.

[116] R. Neal and G. Hinton, "A view of the EM algorithm that justifies incremental, sparse, and other variants," *Learning in graphical models*, vol. 89, p. 355–368, 1998.

[117] L. Huang and D. Chiang, "Better k-best parsing," in *International Workshop on Parsing Technology*, 2005, pp. 53–64.

[118] Y. Bisk and J. Hockenmaier, "Induction of Linguistic Structure with Combinatory Categorial Grammars," in *NAACL HLT Workshop on Induction of Linguistic Structure*, Montréal, Canada, June 2012, pp. 90–95.

[119] Y. Bisk and J. Hockenmaier, "An HDP Model for Inducing Combinatory Categorial Grammars," *Transactions of the Association for Computational Linguistics*, pp. 75–88, 2013.

[120] Y.-W. Teh, "Dirichlet Process," in *Encyclopedia of Machine Learning*. Springer, 2010, pp. 280–287.

[121] Y.-W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei, "Hierarchical Dirichlet Processes," *Journal of the American Statistical Association*, vol. 101, no. 476, pp. 1566–1581, 2006.

[122] Y.-W. Teh, "A Hierarchical Bayesian Language Model based on Pitman-Yor Processes," in *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, Sydney, Australia, July 2006, pp. 985–992.

[123] T. S. Ferguson, "A Bayesian Analysis of Some Nonparametric Problems," *The Annals of Statistics*, vol. 1, pp. 209–230, Mar. 1973.

[124] C. E. Antoniak, "Mixtures of Dirichlet Processes with Applications to Bayesian Nonparametric Problems," *The Annals of Statistics*, vol. 2, pp. 1152–1174, Nov. 1974.

[125] D. Aldous, "Exchangeability and related topics," in *cole d't de Probabilits de Saint-Flour XIII 1983*, ser. Lecture Notes in Mathematics, P. Hennequin, Ed. Springer Berlin Heidelberg, 1985, vol. 1117, pp. 1–198.

[126] J. Sethuraman, "A Constructive Definition of Dirichlet Priors," *Statistica Sinica*, vol. 4, pp. 639–650, 1994.

[127] T. Kwiatkowski, S. Goldwater, L. S. Zettlemoyer, and M. Steedman, "A Probabilistic Model of Syntactic and Semantic Acquisition from Child-Directed Utterances and their Meanings," in *European Chapter of the Association for Computational Linguistics*, Apr. 2012, pp. 234–244.

[128] J. Pitman, "Combinatorial stochastic processes," Department of Statistics, University of California at Berkeley, Tech. Rep. 621, 2002.

[129] P. Liang, M. I. Jordan, and D. Klein, "Probabilistic Grammars and Hierarchical Dirichlet Processes," in *The Oxford Handbook of Applied Bayesian Analysis*. Oxford University Press, 2009.

[130] K. Tu and V. Honavar, "On the Utility of Curricula in Unsupervised Learning of Probabilistic Grammars," in *International Joint Conferences on Artificial Intelligence*, 2011.

[131] C. Bishop, *Pattern Recognition and Machine Learning*. Springer-Verlag, Aug. 2006.

[132] D. M. Blei and M. I. Jordan, "Variational Methods for the Dirichlet Process," in *Proceedings of the Twenty-First International Conference on Machine Learning (ICML 2004)*, Banff, Alberta, Canada, July 2004.

[133] Y. Bisk and J. Hockenmaier, "Probing the linguistic strengths and limitations of unsupervised grammar induction," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Beijing,China, July 2015.

[134] G. Carroll and M. Rooth, "Valence induction with a head-lexicalized PCFG," in *Empirical Methods in Natural Language Processing*, 1998, p. 36–45.

[135] S. Buchholz and E. Marsi, "CoNLL-X shared task on multilingual dependency parsing," in *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*. New York City: Association for Computational Linguistics, June 2006, pp. 149–164.

[136] Y. Goldberg, "Automatic Syntactic Processing of Modern Hebrew," Ph.D. dissertation, Ben-Gurion University of the Negev, Nov. 2011.

[137] J. Gillenwater, K. Ganchev, J. V. Graca, F. Pereira, and B. Taskar, "Posterior Sparsity in Unsupervised Dependency Parsing," *The Journal of Machine Learning Research*, vol. 12, pp. 455–490, Feb. 2011.

[138] T. Naseem, R. Barzilay, and A. Globerson, "Selective Sharing for Multilingual Dependency Parsing," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Jeju, Republic of Korea, July 2012, pp. 629–637.

[139] A. Sogaard, "Two baselines for unsupervised dependency parsing," in *NAACL HLT Workshop on Induction of Linguistic Structure*, Montréal, Canada, June 2012, pp. 81–83.

[140] D. Mareček and Z. Žabokrtský, "Unsupervised Dependency Parsing using Reducibility and Fertility features," in *NAACL HLT Workshop on Induction of Linguistic Structure*, Montréal, Canada, June 2012, pp. 84–89.

[141] G. Druck, G. Mann, and A. McCallum, "Semi-supervised Learning of Dependency Parsers using Generalized Expectation Criteria," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. Suntec, Singapore: IJCNLP, Aug. 2009, pp. 360–368.

[142] J. Gillenwater, K. Ganchev, J. V. Graca, F. Pereira, and B. Taskar, "Sparsity in Dependency Grammar Induction," in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, Uppsala, Sweden, July 2010, pp. 194–199.

[143] H. Yamada and Y. Matsumoto, "Statistical Dependency Analysis With Support Vector Machines," in *In proceedings of 8th International Workshop on Parsing Technologies*, 2003, pp. 195–206.

[144] M. Connor, C. Fisher, and D. Roth, *Starting from Scratch in Semantic Role Labeling: Early Indirect Supervision*, ser. Theory and Applications of Natural Language Processing, A. Villavicencio, T. Poibeau, A. Korhonen, and A. Alishahi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[145] Y. Bisk, C. Christodoulopoulos, and J. Hockenmaier, "Labeled grammar induction with minimal supervision," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Beijing,China, July 2015.

[146] C. Christodoulopoulos, S. Goldwater, and M. Steedman, "Turning the pipeline into a loop: Iterated unsupervised dependency parsing and PoS induction," in *NAACL HLT Workshop on Induction of Linguistic Structure*, Montréal, Canada, June 2012, pp. 96–99.

[147] V. I. Spitkovsky, H. Alshawi, A. X. Chang, and D. Jurafsky, "Unsupervised Dependency Parsing without Gold Part-of-Speech Tags," in *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. Edinburgh, Scotland, UK.: Association for Computational Linguistics, July 2011, pp. 1281–1290.

[148] P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. Della Pietra, and J. C. Lai, "Class-Based n-gram Models of Natural Language," *Computational Linguistics*, vol. 18, 1992.

[149] C. Christodoulopoulos, S. Goldwater, and M. Steedman, "A Bayesian Mixture Model for Part-of-Speech Induction Using Multiple Features," in *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, Edinburgh, Scotland, UK., July 2011.

[150] M. Creutz and K. Lagus, "Morfessor in the Morpho challenge," in *Proceedings of the PASCAL Challenge Workshop on Unsupervised Segmentation of Words into Morphemes*, 2006, pp. 12–17.

[151] L. R. Rabiner, "Readings in speech recognition," A. Waibel and K.-F. Lee, Eds. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, ch. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, pp. 267–296.

[152] M. Johnson, "Why doesn't EM find good HMM POS-taggers," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, Jan. 2007.

[153] P. Blunsom and T. Cohn, "A Hierarchical Pitman-Yor Process HMM for Unsupervised Part of Speech Induction," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 865–874.

[154] A. Haghighi and D. Klein, "Prototype-Driven Learning for Sequence Models," in *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*. New York City, USA: Association for Computational Linguistics, June 2006, pp. 320–327.

[155] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. Prague, Czech Republic: Association for Computational Linguistics, June 2007, pp. 410–420.

[156] C. Christodoulopoulos, S. Goldwater, and M. Steedman, "Two Decades of Unsupervised POS induction: How far have we come?" in *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, Cambridge, MA, Oct. 2010.

[157] W. P. Headden III, D. McClosky, and E. Charniak, "Evaluating Unsupervised Part-of-Speech Tagging for Grammar Induction," in *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*. Manchester, UK: Coling 2008 Organizing Committee, Aug. 2008, pp. 329–336.

[158] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM*, vol. 13, no. 6, pp. 377–387, June 1970.

[159] K. Bollacker, P. Tufts, T. Pierce, and R. Cook, "A platform for scalable, collaborative, structured information integration," in *Intl. Workshop on Information Integration on the Web (IIWeb07)*, 2007.

[160] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: A collaboratively created graph database for structuring human knowledge," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '08. New York, NY, USA: ACM, 2008, pp. 1247–1250.

[161] E. Gabrilovich, M. Ringgaard, and A. Subramanya, "FACC1: Freebase annotation of ClueWeb corpora, Version 1 (Release date 2013-06-26, Format version 1, Correction level 0)," June 2013. [Online]. Available: http://lemurproject.org/clueweb12/

[162] T. Parsons, *Events in the Semantics of English.* The MIT Press, 1990.

[163] X. Yao, "Lean question answering over freebase from scratch," in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations.* Denver, Colorado: Association for Computational Linguistics, June 2015, pp. 66–70.

[164] P. Boonkwan, "Scalable semi-supervised grammar induction using cross-linguistically parameterized syntactic prototypes," Ph.D. dissertation, The University of Edinburgh, 2014.

[165] T. K. Landauer, P. W. Foltz, and D. Laham, "An Introduction to Latent Semantic Analysis," *Discourse Processes*, vol. 25, pp. 259–284, 1998.

[166] T. Mikolov, W.-t. Yih, and G. Zweig, "Linguistic Regularities in Continuous Space Word Representations," in *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.* Atlanta, Georgia: Association for Computational Linguistics, June 2013, pp. 746–751.

[167] J. Maillard, S. Clark, and E. Grefenstette, "A Type-Driven Tensor-Based Semantics for CCG," *EACL 2014 Type Theory and Natural Language Semantics Workshop*, 2014.

[168] O. Levy and Y. Goldberg, "Linguistic Regularities in Sparse and Explicit Word Representations," in *Conference on Computational Natural Language Learning*, 2014.

[169] O. Levy, Y. Goldberg, and I. Dagani, "Improving Distributional Similarity with Lessons Learned from Word Embeddings," *Transactions of the Association for Computational Linguistics*, vol. 3, 2015.

[170] M. Lewis and M. Steedman, "Combined Distributional and Logical Semantics," *Transactions of the Association for Computational Linguistics*, vol. 1, pp. 179–192, 2013.

[171] J. Clarke, D. Goldwasser, M.-W. Chang, and D. Roth, "Driving Semantic Parsing from the World's Response," in *Conference on Computational Natural Language Learning*, 2010.

[172] D. Goldwasser and D. Roth, "Learning From Natural Instructions," in *International Joint Conferences on Artificial Intelligence*, 2011.