Linguistically Motivated Combinatory Categorial Grammar Induction

Adrienne X. Wang

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2016

Reading Committee:

Luke Zettlemoyer, Chair

Daniel Weld

Yejin Choi

Program Authorized to Offer Degree:
Computer Science & Engineering

University of Washington

Abstract

Linguistically Motivated Combinatory Categorial Grammar Induction

Adrienne X. Wang

Chair of the Supervisory Committee:
Associate Professor Luke Zettlemoyer
Department of Computer Science & Engineering

Combinatory Categorial Grammar (CCG) is a widely studied grammar formalism that has been used in a variety of NLP applications, e.g., semantic parsing, and machine translation. One key challenge in building effective CCG parsers is a lack of labeled training data, which is expensive to produce manually. Instead, researchers have developed automated approaches for inducing the grammars. These algorithms learn lexical entries that define the syntax and semantics of individual words, and probabilistic models that rank the set of possible parses for each sentence. Various types of universal or language specific prior knowledge and supervising signals can be exploited to prune the grammar search space and constrain parameter estimation.

In this thesis, we introduce new methods for inducing linguistically motivated grammars that generalize well from small amounts of labeled training data. We first present a CCG grammar induction scheme for semantic parsing, where the grammar is restricted by modeling a wide range of linguistic constructions, then introduce a new lexical generalization model that abstracts over systematic morphological, syntactic, and semantic variations in languages. Finally, we describe a weakly supervised approach for inducing broad scale CCG syntactic structures for multiple languages. Such approaches would have the greatest utility for low-resource languages, as well as domains where it is prohibitively expensive to gather sufficient amounts of training data.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Luke Zettlemoyer has been a great advisor to work with. I would like to express my sincere gratitute to Luke for opening the door to NLP and allowing me to pursue topics which I am truly passionate for. Thank you for the guidance and support, and being a rock of stability in my research. I am also grateful to other committe members, Dan Weld, Yejin Choi, and Emily Bender for their advice.

I wish to thank my former advisor, Martin Tompa, for guiding me through my initial years in graduate school and staying positive and encouraging during the time when I was confused about my research direction, and my former committee, Larry Ruzzo, Bill Noble and Anna Karlin. UW has provided an open and supportive environment, which made my transition into another research field an easy process. I wish to offer heartfelt thanks to Lindsay Michimoto, without whom this transition could not have been as smooth.

I am honored to be part of UW CSE and have had the opportunity to work with the brightest researchers. I would like to thank the UW NLP group - Mark Yatskar, Nicholas Fitzgerald, Tom Kwiatkowski, Kenton Lee, Eunsol Choi, Mike Lewis, Yoav Artzi, Gabriel Schubiner, Leila Zilles, Luheng He, Victoria Lin, for valuable research conversations that stimulated insight in my work, my office mates for the wonderful laughter, many other friends and colleagues in the UW research community whom I had the privilege to know and work with, and people outside UW whom I shared many precious moments with over the years. I want to particularly thank Tom Kwiatkowski for being a dedicated mentor and a caring friend. Tom has gone to great lengths to teach me CCG and encouraged me to pull my work together. I would also like to thank Xiao Ling for being a great friend who has never hesitated to offer help, and more importantly, always entertained me with his quirky sense of humor, and Robert Gens, whose positive attitude and passion about research was

contagious.

I feel very grateful for the tremendous amount of trust my parents placed in me, during both good and bad times. They have been great role models for me to emulate and have supported my decisions through every aspect of my life. To my grandparents, my cousins and the rest of my family, I thank you for the ongoing encouragement. Last but not least, I would like to express my deepest appreciation to my dear husband, for the selfless and endless support, strength, wisdom and optimism, for backing me up whenever and wherever possible, and for many inspiring research discussions that eventually lead to completion of this thesis. I have been extremely fortunate to have you in my life - none of these would have been possible without you.

DEDICATION

to M

Chapter 1

INTRODUCTION

Statistical parsing has enabled us to automatically produce accurate analyses of natural language text. State-of-the-art statistical parsers, regardless of the grammar types, unfortunately, are all supervised models trained with abundant data. Creating annotated resources for fully supervised systems is expensive and requires plenty of expertise. To minimize the annotation burdern, the task of grammar induction over unannotated texts has received a great deal of attention. Over the last decade, considerable progress has been achieved on unsupervised approaches [36, 53, 64–66, 83, 93, 102], both boosting performance on longer sentences and extending to more languages, but the models employed still leave a lot to be desired in comparison to their supervised counterpart: unsupervised acquisitions typically produce overly simplistic, less descriptive models, limiting their capability for modelling the more complex linguistic phenomena. Researchers have studied incorporating various linguistic constraints, such as prior knowledge that captures the highly structured processes of languages [12, 13, 16, 36, 48, 83], or training with cheaper to obtain signals, for example natural instructions [51], interactions with the environment [6, 21, 105], and question-answer pairs [8, 22, 32, 70, 77], to effectively define and bias the model towards fewer candidate structures. Their significant successes have provided insight into the amount of latent structures discoverable without access to labor intensive annotated training data. Such approaches would have the greatest utility for low-resource languages, as well as domains where it is prohibitively expensive to gather sufficient amounts of training data.

In this thesis, we introduce frameworks for inducing linguistically motivated grammars that generalize well from less data and less supervision. We focus specifically on inducing Combinatory Categorial Grammar, and the goal is to learn robust parsers that recover the underlying syntactic and semantic structures of sentences through incorporation of language priors as well as widely

available resources.

Learning CCG grammar that generalizes well will benefit downstream tasks that build upon parsing results. CCG's expressive and flexible constituent structure makes it a preferable grammar formalism for many NLP applications, including semantic parsing [5, 71, 72, 110, 111], machine translation [10], wide-coverage syntactic parsing [30, 58, 75, 107], semantic role labeling [20, 49, 73], and language generation [45]. While we focus on developing CCG grammar induction algorithms for specific applications, the modeling techniques use universal knowledge, as such are potentially applicable to other grammar formalisms in more general settings.

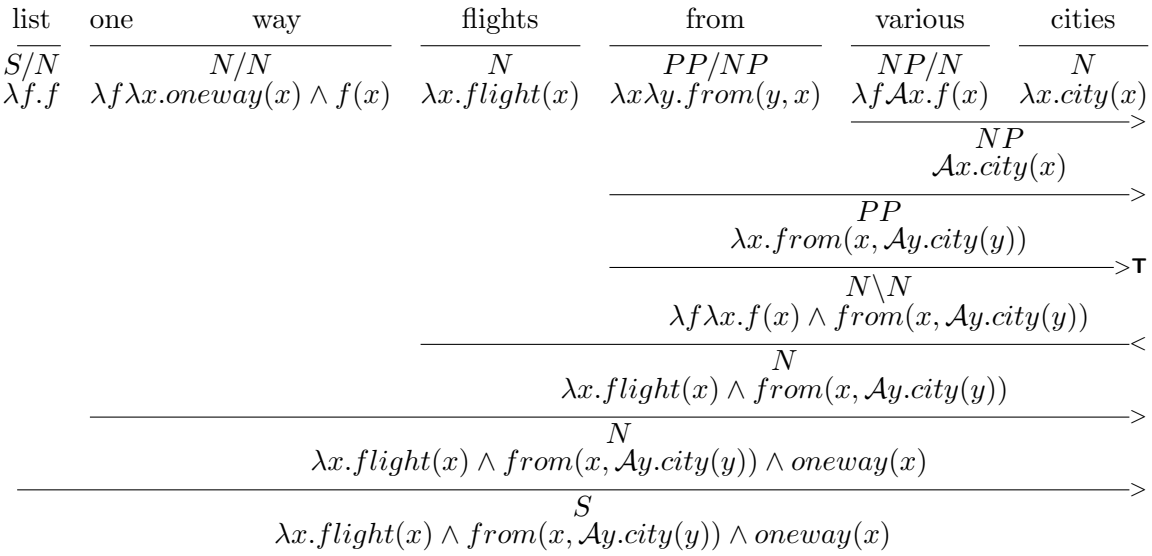


Figure 1.1: An example CCG parse.

1.1 Challenges

Inducing CCG grammars without reliance on the underlying structures is a highly challenging task.

Structure CCG is a strongly lexicalized grammar formalism, which defines the syntactic and semantic aspects of individual words and phrases and a small set of rules on how to combine them to analyze complete sentences. Figure 1.1 shows an example CCG derivation for semantic

(a) Show me arrivals in Seattle at 2pm
$\lambda x.to(x, SEA) \wedge arrival\_time(x, 1400)$

(b) Show me departures from Seattle at 2pm
$\lambda x.from(x, SEA) \wedge departure\_time(x, 1400)$

Figure 1.2: A sample of sentences from the travel-planning domain paired with semantic representation.

analysis. The lexical entries are listed across the top and the output lambda-calculus meaning representation is at the bottom. As illustrated in the example, two types of ambiguity need to be addressed during parsing: which definitions best interpret each individual word, and which derivations best represent the sentence's underlying linguistic structure. There are many choices for each decision, and constructing the desired interpretation relies on the correct inference of all the latent labels. The huge hypothesis space, especially when syntax is coupled with other signals for joint inference, such as logical forms as in the above example, imposes significant challenges on the learning task.

Data In lieu of fully annotated derivations, which dictate the decisions during parsing, the system must learn from weaker forms of supervision. Prior work has induced grammars from a variety of signals, for example, logical forms [62, 69, 82, 106, 110, 111] and question-answer pairs [8, 22, 32, 70, 77]. Challenges arise from each type of annotation through interactions with the system. Specific to semantic grounding, the choice of the annotated meaning representations greatly influences learning. For example, the logical forms in Figure 1.2 require the system to learn two definitions for the word "at": one referring to arrival time ($\lambda y \lambda x.arrival\_time(x, y)$) and another to handle departure time ($\lambda y \lambda x.departure\_time(x, y)$). Such proliferation of lexical associations introduces lexical ambiguity and complicates learning. Using another type of syntactic structure, such as head-dependent relations, to supervise the induction process presents a similar problem. Figure 1.3 exemplifies a mismatch between the structures encoded by the two formalisms. The arcs on the top represent the annotated dependencies, while the CCG derivation

implies semantic dependencies, represented by the arcs below the sentence. Without remediation, such discrepancies have adverse effect on the quality of the learned grammar.

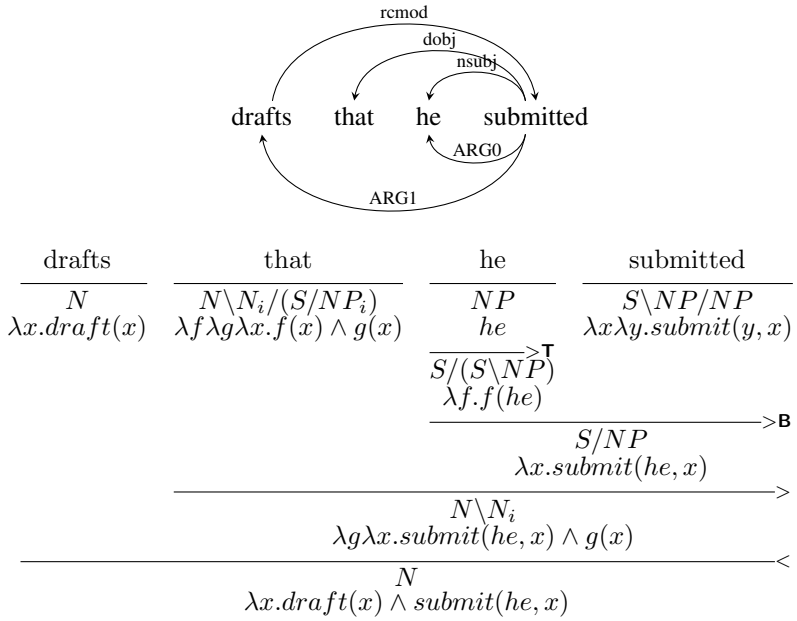


Figure 1.3: Structural discrepancies encoded in two grammar formalisms

1.2 Overview of Approaches

We present CCG grammar induction algorithms for learning a domain-specific semantic parser and generating broad coverage syntactic structures. Here, we briefly overview the approaches before defining the problems and assumptions in more detail throughout the thesis.

1.2.1 Linguistically Motivated Modeling for Semantic Parsing

We first consider the problem of learning a semantic parser that maps sentences to meaning representations, where previous work [70, 72, 110, 111] has adopted semantic representations that introduce lexical ambiguity and induced grammars that over-generate. We design a semantic representation that aligns closely to the underlying syntactic structure, and the grammar is restricted

by modeling a wider range of linguistic constructions than considered in previous work, for example introducing explicit treatments of relational nouns, Davidsonian events, and verb tense. This approach allows us to learn a significantly more compact lexicon that generalizes well in practice.

1.2.2 CCG Lexical Generalization for Semantic Parsing

We then present a linguistically motivated lexical generalization method for semantic parsing that improves parsing accuracies and significantly reduces the sample complexity of each domain dependent learning task. The induction scheme has two major contributions: 1. We import external resources to impose constraints on parses that must be considered 2. We introduce a morpho-syntactic, factored lexical representation that models systematic syntactic, semantic, and morphological variations in English. This includes, for example, the facts that verbs in relative clauses and nominal constructions (e.g., "flights departing NYC" and "departing flights") should be infinitival while verbs in phrases (e.g., "What flights depart at noon?") should have tense. We demonstrate empirically that the proposed generalization model enables effective learning of complete parsers with much less data.

1.2.3 Broad-Coverage CCG Syntactic Grammar Induction

We then present a discriminative model for the weakly supervised learning of CCG syntactic structures. We take advantage of the widely available universal dependency treebanks, where the encoded structures are used to supervise CCG grammar induction. Dependency supervision yields substantial improvements over unsupervised approaches, and the model is proven to be robust cross-linguistically, making it applicable for generating broad scale CCG treebanks for multiple languages.

1.3 Contributions

This thesis includes the following modeling and learning techniques for inducing CCG grammar from natural language sentences.

- Chapter 4 describes a list of linguistically motivated, engineered lexical templates for learning semantic parsers, and reuses an existing learning algorithm with minor modifications.

- Chapter 5 introduces a *morpho-syntactic* factored lexical representation and how to induce such compact CCG lexicons.

- Chapter 6 describes a joint model for predicting CCG syntactic structures and bilexical dependencies and a learning algorithm for inducing grammar and estimating parsing parameters.

1.4 Overview of Thesis

We summarize relevant background in Chapter 2. Chapter 3 describes related work for CCG parsing and grammar engineering. Chapter 4, 5 and 6 present algorithms and experimental results for the the grammar induction approaches outlined above. Finally, Chapter 7 concludes the thesis and proposes future work.

Chapter 2

BACKGROUND

This chapter reviews the relevant background that our approaches are based on.

2.1 Combinatory Categorial Grammar

CCG [96, 97] is an expressive, lexicalized grammar formalism that tightly couples syntax and semantics, and can be used to model a wide range of linguistic phenomena. A traditional CCG grammar includes a lexicon Λ with lexical entries like the following:

$$flights \vdash N : \lambda x.flight(x)$$
$$from \vdash PP/NP : \lambda y.\lambda x.from(x, y)$$
$$cities \vdash N : \lambda x.city(x)$$

where a lexical item $w \vdash X : h$ has words w, syntactic category X, and logical expression h. For syntactic parsing, the lexical entries drop the logical expression, and only comprise the syntatic portion.

Combinatory Rules CCG uses a small set of *combinatory rules* to jointly build syntactic parses and semantic representations. The most common combinatory rules are the *functional application* rules.

The forward and backward application rules:

$$X/Y : f \quad Y : g \quad \Rightarrow \quad X : f(g) \qquad (>)$$
$$Y : g \quad X\backslash Y : f \quad \Rightarrow \quad X : f(g) \qquad (<)$$

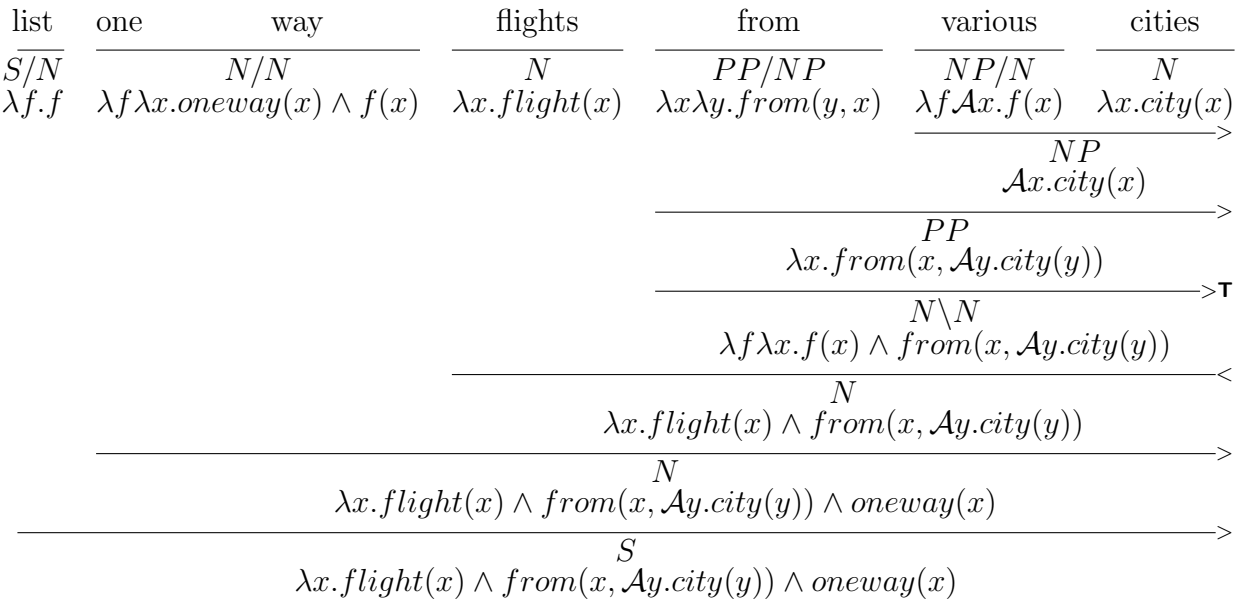


Figure 2.1: An example CCG parse.

Another set of basic combinatory rules are the *functional forward (*$>$ **B***) and backward (*$<$ **B***) composition* rules.

The forward and backward composition rules:

$X/Y : f \quad Y/Z : g \Rightarrow X/Z : \lambda x.f(g(x)) \quad (>$ **B**$)$

$Y\backslash Z : g \quad X\backslash Y : f \Rightarrow X\backslash Z : \lambda x.f(g(x)) \quad (<$ **B**$)$

These rules apply to build syntactic and semantic derivations concurrently. CCG typically includes a set of unary type shifting rules to parse specific syntactic constructions.

The type-raising rules:

$X : g \Rightarrow T/(T\backslash X) : \lambda f.f(g) \quad (>$ **T**$)$

$X : g \Rightarrow T\backslash(T/X) : \lambda f.f(g) \quad (<$ **T**$)$

These general type-raising combinators are restricted to categories ($X \in \{N, NP\}$, and $T \in \{S, S\backslash N, S/N\}$). The special categories have reasonable semantic interpretations in English.

For coordination, many CCG parsers use a special ternary rule

The coordination rules:

$X \quad X[conj] \Rightarrow X \qquad (\&\mathbf{1})$

$conj \quad X \quad \Rightarrow X[conj] \quad (\&\mathbf{2})$

which allows us to extract the argument of the conjuncts. Throughout this thesis, we represent coordination as a lexical category of the form $(X\backslash X)/X$ instead of including the coordination rules in the parser. This treatment requires specific categories to merge the conjuncts. For example, $S/N\backslash(S/N)/(S/N)$ is required for transitive verbs, and $N\backslash N/N$ is required for nouns. Since representing the coordination functions in the lexicon introduces complex categories not generalizable to other word classes, their usages are carefully constrained to the coordinating conjunctions.

Finally, we implement a set of additional type-shifting rules to deal with special linguistic phenomena in English. This additional set includes the plural existential type-shifting rule

The plural existential type-shifting rule:

$N : f \Rightarrow NP : \mathcal{A}x.f(x) \quad (\mathbf{T_p})$

which is used to shift the bare form nouns to a single entity, and the that-less type-shifting rule

The that-less relative type-shifting rule:

$N : f \quad S\backslash N : g \Rightarrow N : \lambda x.f(x) \wedge g(x) \quad (> \mathbf{T_t})$

$N : f \quad S/N : g \Rightarrow N : \lambda x.f(x) \wedge g(x) \quad (< \mathbf{T_t})$

which fills in the missing 'that' phrases for the that-less relative constructions.

The sentential modifier type-shifting rules are created to allow verb phrases to attach to the main verb as an infinitival or participial modifier:

The sentential modifier type-shifting rules:

$S/N : f \quad S/N : g \Rightarrow S/N : \lambda x.f(x) \wedge g(x) \quad (> \mathbf{T_s})$

$S\backslash N : f \quad S\backslash N : g \Rightarrow S\backslash N : \lambda x.f(x) \wedge g(x) \quad (< \mathbf{T_s})$

We also implement type-raising rules for compact representation of PP (prepositional phrase) and AP (adverbial phrase).

$$PP : g \Rightarrow N\backslash N : \lambda f \lambda x.f(x) \wedge g(x) \quad (\mathbf{T})$$
$$AP : g \Rightarrow S\backslash S : \lambda f \lambda e.f(e) \wedge g(e) \quad (\mathbf{T})$$
$$AP : g \Rightarrow S/S : \lambda f \lambda e.f(e) \wedge g(e) \quad (\mathbf{T})$$

2.2 Semantics

CCG is known for its transparent interface to jointly represent syntax and semantics, which makes it a preferable grammar formalism for recovering semantic analyses. In particular, a syntactic category is coupled with a logical representation in the lexicon.

2.2.1 Lambda Calculus

We represent the meanings of sentences, words and phrases with lambda calculus logical expressions. We use a version of the typed lambda calculus [24], in which the basic primitive types include e for entities, ev for events, t for truth values and i for numerals. The system allows complex function types that are combinations of the primitive types, for example, $\langle e, t\rangle$ is a function type that maps from e-typed entities to t-typed truth values. The expression $\lambda x.flight(x)$ with type $\langle e, t\rangle$ takes an entity and returns a truth value, and represents a set of entities that the $flight$ function evaluates to true.

Our lambda-calculus expressions contain constants, quantifiers and logical connectors. Logical constants are used to represent entities, often database IDs like *texas*, or functions, such as $flight$ in the expression $\lambda x.flight(x)$. We also make use of logical connectors, including conjunction $\wedge$, disjunction $\vee$, negation $\neg$, and implication $\Rightarrow$. The expressions include universal quantification $\forall$ and existential quantification $\exists$. For example, $\exists x.river(x) \wedge loc(x, texas)$ is true if and only if there is at least one river that is located in Texas. In Chapter 4, we will introduce another quantifier $\mathcal{A}$, which implements a $\langle\langle e, t\rangle, e\rangle$ function as suggested by [97]. $\mathcal{A}x.river(x)$ represents a single entity drawn from the set of rivers.

2.3 Factored Lexicon

Our work extends the factored CCG lexicon introduced by [72], in which lexical entries are represented by lexemes and templates separately. Unlike traditional CCG lexicons that associate categories with each word and phrase, the factored lexicon models systematic variations in a given word class by factoring entries into a lexeme $(w, \vec{c})$, which pairs a word sequence w with logical constants $\vec{c} = [c_1 \dots c_k]$, and a template

$$\lambda(\xi, \vec{v}).[\xi \vdash X : h_{\vec{v}}]$$

which abstracts over the constants and represents the syntactic and semantic information. To instantiate a template, ξ is filled with the original word w and the constants in $\vec{c}$ replace the variables $\vec{v}$.

Consider a traditional lexicon that contains the following entries:

$flights \vdash N : \lambda x.flight(x)$

$from \vdash PP/NP : \lambda y.\lambda x.from(x, y)$

$cities \vdash N : \lambda x.city(x)$

Its factored representation is constructed by the lexemes and the templates

$(flights, [flight])$

$(from, [from])$

$(cities, [city])$

$\lambda(\xi, \vec{v}).[\xi \vdash N : \lambda x.v_1(x)]$

$\lambda(\xi, \vec{v}).[\xi \vdash PP/NP : \lambda y.\lambda x.v_1(x, y)]$

Both flights and cities are instantiated by the standard N category $\lambda(\xi, \vec{v}).[\xi \vdash N : \lambda x.v_1(x)]$. This factorization allows related entries to share templates, and therefore, generalizes better during parsing, especially for words unseen in the training data. In Chapter 5, we will present a factored lexicon that further generalizes lexemes and efficiently models common syntactic and morphological variations.

2.4 Weighted CCGs

There are potentially exponentially many parses for each given sentence due to ambiguity in the CCG lexicon and the set of combinatory rules applied to construct the derivations. To rank the parses allowed under Λ, we define a weighted CCG grammar as $G = (\Lambda, \Theta)$, where Λ is a CCG lexicon and $\Theta \in \mathbb{R}^d$ is a d-dimensional parameter vector.

For a sentence x, G produces a set of candidate parse trees $Y = \mathsf{Y}(x; G)$. The joint probability of a parse tree y and the predicted label z [1] for sentence x is defined as:

$$P(y, z|x; \theta, \Lambda) = \frac{e^{\theta \cdot \phi(x,y,z)}}{\sum_{z'} \sum_{y' \in \mathsf{Y}(x,z';G)} e^{\theta \cdot \phi(x,y',z')}} \tag{2.1}$$

The inference problem is to predict the most likely z given the current model $G = (\Lambda, \Theta)$ by marginalizing all parses y that generate z:

$$\hat{z} = \arg\max_{z} \sum_{y \in \mathsf{Y}(x,z;G)} P(y, z|x; \theta, \Lambda) \tag{2.2}$$

The features Φ used in the model will be described in more detail in the later chapters.

2.5 Learning with GENLEX

We will also make use of an existing learning algorithm GENLEX from ZC07 [111]. In Chapter 4 and 5, we reuse the overall learning framework, as described in Figure 2.2, but modify the lexical learning to factor the entries into lexemes, base form templates and morphological transformation templates, as well as the feature space to represent the factored components in the model. We review the learning algorithm here, and will describe our modifications in Section 5.5.

Given a set of training examples $D = \{(x_i, z_i) : i = 1...n\}$, x_i being the ith sentence and z_i being its annotated logical form, the algorithm learns a set of parameters Θ for the grammar, while also inducing the lexicon L.

---

[1] z is a logical form in Chapter 4 and 5 and a dependency structure in Chapter 6

Inputs:
Training set $\{(x_i, z_i) : i = 1 \dots n\}$ where x_i is a sentence and z_i is a logical form. Initial lexicon L_0. Empty lexeme set L. Predefined template set T. Number of iterations J.

Definitions: Let $\mathsf{Y}(x; \Lambda)$ be the set of all possible parses for the sentence x, given the lexicon Λ. $\mathsf{Y}(x, z; \Lambda)$ be the set of all possible parses for the sentence x, which return the logical form z. $LEX(y)$ is the set of lexical entries used in the parse tree y. Let T be the set of predefined templates. $GENLEX(x, z; T)$ takes as input a sentence x, a logical form z and a set of templates T, and returns a set of lexical entries, as defined in Section 2.5.

Algorithm:
Initialize θ, $\Lambda \leftarrow \Lambda_0$
For $i = 1 \dots n$:

Step 1: (Lexical expansion)

a. $L_G \leftarrow GENLEX(x_i, z_i; T) \cup L$
b. $\tilde{y} = \arg\max_{y \in \mathsf{Y}(x_i; z_i, (L_G, T))} P(y|x, z; \Theta, (L_G, T))$
c. $L \leftarrow L \cup LEX(\tilde{y})$

Step 2: (Update parameters)

a. $\hat{y}, \hat{z} = \arg\max_{y, z \in \mathsf{Y}(x_i; (L, T))} P(y, z|x; \Theta, (L, T))$
b. $\theta \leftarrow \theta + \phi(x_i, \tilde{y}, z_i) - \phi(x_i, \hat{y}, \hat{z})$

Output: Parameters θ and lexicon Λ

Figure 2.2: The ZC07 learning algorithm.

The learning algorithm uses a function $\mathsf{GENLEX}(x, z)$ to define a set of lexical entries that could be used to parse the sentence x and construct the logical form z. For each training example (x, z), $\mathsf{GENLEX}(x, z)$ maps all substrings x to a set of potential lexical entries, generated by exhaustively pairing the logical constants in z using a set of hand-engineered templates. The example is then parsed with this much bigger lexicon and lexical entries from the highest scoring parses are added to Λ. The parameters Θ used to score parses are updated using a perceptron learning algorithm.

2.6 Supertagging Model

Low-dimensional, dense word embeddings capture statistical syntactic and semantic similarities between words, and therefore, can reduce sparsity resulted from the traditional lexicalized indicator features and enable the model to generalize better. Embedding features have been widely used in modeling approaches for a variety of NLP tasks, including tagging [37, 76], machine transla-

tion [42], dependency parsing [27], NER and chunking [104] and constituency parsing [75, 92]. We follow Lewis and Steedman [76]'s window approach network (Figure 6.11) to predict lexical categories for sentences. Each word in x is represented as a d-dimensional vector ${e_i}^w \in \Re^d$, and the embedding matrix is $E^w \in \Re^{dV_w}$, where V_w is the vocabulary size. As in [76], we also compute 2 non-embedding feature sets to capture each word's 2-character suffix and whether it is capitalized, and map each feature onto a 5-dimensional vector. The input layer computes features for each individual word, and the second layer concatenates features from each word in a context window of size 7. The second layer is mapped to a hidden layer through a *hard-tanh* activation function, and another layer through a *softmax* activation function to predict lexical categories.

2.7 CCG Dependency Structures

CCG and other expressive grammar formalisms like LFG, HPSG and LTAG attempt to recover deep semantic analyses, instead of surface structures. CCG categories encode predicate-argument relations that are ideal for recovering long-range dependencies arising from complex linguistic construtions like control and binding. Since each word can be associated with multiple parents, the CCG parses return non-projective dependency graphs. This is in direct contrast to the bilexical universal dependencies we use in Chapter 6. We briefly review the CCG dependencies in this section.

The predicate-argument dependency structure is a set of relations encoded in terms of the lexical heads of functor categories and their arguments. Since Clark et al. [31]'s introduction of the CCG predicate-argument dependencies, it has become the standard to evaluate CCG parsers [59]. Clark and Curran [30] define the dependency relations as 4-tuples: $\langle h_f, f, s, h_a \rangle$, where h_f is the token of the lexical category encoding the relation; f is the lexical category; s is the position of the argument slot; and h_a is the head token of the argument. For example, in the following derivation,

$$\dfrac{\dfrac{\overline{\text{he}}}{NP} \quad \dfrac{\overline{\text{eats}}}{S\backslash NP/NP} \quad \dfrac{\overline{\text{noodle}}}{NP}}{\dfrac{\dfrac{}{S\backslash NP}>}{S}<}$$

the transitive verb category $S\backslash NP/NP$ encodes the object relation $\langle eats, S\backslash NP_1/NP_2, 2, noodle\rangle$ and the subject relation $\langle eats, S\backslash NP_1/NP_2, 1, he\rangle$. Both dependency relations are realized when the NP argument slot is filled through the application rules.

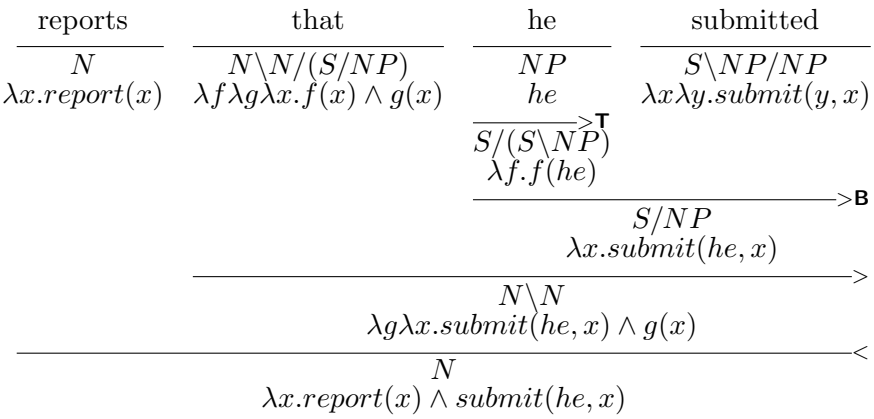


Figure 2.3: CCG derivation illustrating the long range dependency propagation

The predicate-argument structure is capable of generating dependency relations beyond the bilexical surface dependencies. Its underlying semantic relations facilitates recovery of long-range dependencies, making it a powerful feature compared to other expressive grammar formalisms. Figure 2.3 illustrates the process of propagating the arguments for long-range dependencies. The derivation generates the following dependency relations:

$\langle that,\ (N\backslash N)/(S/NP),\ 1,\ reports\rangle$

$\langle that,\ (N\backslash N)/(S/NP),\ 2,\ submitted\rangle$

$\langle submitted,\ (S\backslash NP_1)/NP_2,\ 1,\ he\rangle$

$\langle submitted,\ (S\backslash NP_1)/NP_2,\ 2,\ reports\rangle$

The long-range dependency between submitted and reports $\langle submitted, (S\backslash NP_1)/NP_2, 2,\ reports\rangle$ is recovered through unifying the object argument x (NP_2) in the transitive verb category $submitted \vdash S\backslash NP/NP : \lambda x\lambda y.submit(y, x)$ and the variable x in $reports \vdash N : \lambda x.report(x)$. The relative

pronoun *that* requires the category $N\backslash N_i/(S/NP_i) : \lambda f \lambda g \lambda x.f(x) \wedge g(x)$, which takes arguments S/NP and N, and provides the mechanism for coindexation of the NP and N. Its semantics clearly shows the variable x is shared by the functions f and g.

Chapter 3

RELATED WORK

3.1 Syntactic Grammar Induction

Unsupervised grammar induction has gained general attention for several decades - many grammar representations have been explored for this task, among which, the two most common ones are constituency grammar and dependency grammar [80]. The search space for both grammars is highly ambiguous: for dependency grammar, the space is defined by a set of spanning trees consisting unlabeled edges between words; for constituency grammar, such as context-free grammar (CFG), the problem is further complicated by the necessity to invent a set of categories to label the non-terminals. While inducing constituency grammar remains a challenging problem [25, 26, 29, 65], the attempts with dependency grammar have been more fruitful, given its set of labels directly observable in the data. Most recent approaches extend from Klein and Manning's dependency model with valence (DMV) [66], proposing novel techniques ranging from additional probability models to address data sparsity [33, 34, 50, 53, 90], more sophisticated learning strategies [93, 95], incorporation of linguistic priors [9, 35, 43, 83] to leveraging data from the web to differentiate word classes [78].

Grammar Induction with Linguistic Constraints Our work falls in the broader category of weakly supervised induction that employs linguistic priors to improve learning of syntactic structure. Several approaches have explored this direction, with varying level of linguistic knowledge incorported, and prove to yield substantial improvements [9, 16, 43, 83] over purely unsupervised baselines.

Snyder et al. [91] took advantage of existing bilingual parallel corpora and induced grammar for both languages assuming access to word-level alignments. Similarly, Ganchev et al. [47] exploited bitext projections and an English parser as constraints. Not requiring translated text and alignments, Berg-Kirkpatrick et al. [9] used phylogenetic priors of multiple languages to share model parameters. The limiting factor for approaches like these is the scarce availability of such parallel resources.

There are also approaches that used language specific priors. Haghighi and Klein [52] extracted bracketing rules from WSJ10 to improve CFG induction for English and Chinese. Druck et al. [43] used expert engineered dependency rules to improve the accuracy of DMV. Naseem et al. [83] diverged from the other work by using universal knowledge, represented as 13 soft universal head-dependent pairs. The rules were imposed to bias the model towards prefering the predefined dependency relations. However, this method still required language-specific heuristics to boost accuracy at test time.

There has also been a collection of work based on CCG. Bisk and Hockenmaier have produced a number of CCG induction techniques, only providing POS-tagged text and heuristics on major word classes, like nouns and verbs. They paired their induction algorithm with a basic MLE model [12], a more sophisticated nonparametric bayesian HDP model [13], and with additional modeling of lexicalization and punctuation [14]. Boonkwan and Steedman [16] constructed a CCG lexicon from a simple questionnaire for linguists to assign syntactic prototypes. The result of the survey defined the grammar space. Garrette et al. [48] proposed a CCG induction scheme with no direct supervision on syntactic structures, but assumed access to a partial gold tag dictionary and expert knowledge for pruning, which also circumvented the need for inventing most of the categories, simplifying the problem to learning the structure only.

CCG Category Invention Our strategy to propose new lexical items is generalized from Thomforde and Steedman's chart inference algorithm [100], which invents CCG categories for an unseen word from a partial chart. Our induction relies on stronger dependency signals to define the search space, but grants more flexibility and constructs the entire lexicon from scratch. Ambati et al. [1]

induced a CCG lexicon from a Hindi dependency treebank, but applied a set of very simplistic heuristics and combinators, making it difficult to recover all linguistic constructions.

Constructing CCG Treebanks Hockenmaier and Steedman [59] introduced the CCGBank, a treebank consisting a set of CCG derivations automatically converted from the Penn Treebank. Since the introduction, quite a few accurate supervised CCG syntactic parsing models [30, 48, 58, 75, 76] have been developed. Other than English, a small number of languages have also converted their phrase-structure treebanks to CCG, including Chinese [101], Arabic [19], and German [56], providing abundant data to train supervised parsers. Like our approach, there have been some efforts at converting from dependency treebanks. Bos et al. [18] constructed a CCGbank from an Italian dependency treebank by translating dependency trees into phrase-structure trees and then CCG derivations. Ambati et al. [1] first extracted a CCG lexicon from Hindi dependencies, and then produced CCG trees using a CKY parser.

3.2 Learning Semantic Parsers

Grammar induction methods for CCG semantic parsers have either used hand-engineered lexical templates [4, 110, 111], or developed algorithms to learn such templates directly from data [71, 72]. We extend the first approach, and show that better lexical generalization provides significant performance gains.

Grammar Formalisms CCG provides a transparent mapping between each word's syntactic function and semantic interpretations. Although being a common choice for semantic parsers, many other formalisms have been studied, including DCS trees [77], integer linear programs [32], and synchronous grammars [3, 61, 106]. All of these approaches build complete meaning representations for individual sentences, but the data we use has also been studied in related work on cross-sentence reasoning [81, 112] and modeling semantic interpretation as a tagging problem [54, 103]. Although we focus on full analysis with CCG, the general idea of using linguistic constraints to improve learning is broadly applicable.

Supervision Semantic parsers are also commonly learned from a variety of different types of supervision, including logical forms [62, 69, 82, 106], question-answer pairs [32, 77], conversational logs [4], distant supervision [23, 67], sentences paired with system behavior [7, 28, 51], and even from database constraints with no explicit semantic supervision [86]. We learn from logical forms, but CCG learning algorithms have been developed for each case above, making our techniques applicable.

Scalability There has also been recent progress on semantic parsing against large, open domain databases such as Freebase [8, 22, 70]. The key challenge for these datasets is learning to map words to the correct logical constants, and all of the advances complement our focus on more syntactically rich models that support compositional reasoning. Unfortuantely, existing Freebase datasets are not a good fit to test our approach. In order to collect the open domain datasets, the questions were filtered and do not require complex, compositional meaning representations like GeoQuery and ATIS. Instead, the sentences they include have relatively simple structure and can be interepreted accurately using only factoid lookups with no database joins [108].

Linguistically Motivated Grammar Design There has been significant related work that influenced the design of our morpho-syntactic grammars. Our work brings together several strands of research, including linguistics studies of relational nouns [41, 85], Davidsonian events [40], parsing as abduction [55], and other more general theories for lexicons [87] and CCG [58, 98]. It also includes work on using morphology in CCG syntactic parsing [60] and more broad-coverage semantics in CCG [17, 74]. However, our work is unique in studying the use of related ideas for semantic parsing.

Chapter 4

LINGUISTICALLY MOTIVATED GRAMMAR MODELING FOR SEMANTIC PARSING

4.1 Introduction

Semantic parsers map sentences to formal representations of their meaning [77, 109, 110]. One common approach is to induce a probabilistic CCG grammar, which defines the meanings of individual words and phrases and how to best combine them to analyze complete sentences. There has been recent work developing learning algorithms for CCG semantic parsers [4, 71] and using them for applications ranging from question answering [23, 70] to robot control [68, 79], but most work has focused primarily on developing new learning algorithms, showing how to induce models from labeled meaning representations [62, 110] or weaker forms of supervision [4, 77]. Relatively little attention has been paid to determining what semantic representations should be constructed, and how these decisions influence learning and parsing.

Consider the meaning representations marked with A in Figure 4.1. Although commonly used [70, 72, 110, 111], such representations present challenges. Given sentences such as (a) and (b) in Figure 4.1, the induced model will need to pair the word "at" with two separate definitions: one referring to arrival time ($\lambda y \lambda x.\ arrival\_time(x, y)$) and another to handle departure time ($\lambda y \lambda x.\ depart\_time(x, y)$). Similarly, the word "of" in Figure 4.1(c) will introduce the loc constant, which will conflict with the sentence in Figure 4.1(d). Such proliferation of lexical associations creates significant ambiguity at the lexical level, providing a challenge for learning and inference, given the large number of conflicting analyses they will allow for each sentence, especially when the complexity of the domain and grammar increases.

Careful choice of representation can avoid these problems. Consider the alternative logical forms (marked with B in Figure 4.1). Examples (a) and (b) show how adopting Davidsonian

(a) Show me arrivals in Seattle at 2pm
A: $\lambda x.to(x, SEA) \wedge arrival\_time(x, 1400)$
B: $\lambda x.arrive(\mathcal{A}e.to(e, SEA) \wedge time(e, 2PM),\ x)$

(b) Show me departures from Seattle at 2pm
A: $\lambda x.from(x, SEA) \wedge arrival\_time(x, 1400)$
B: $\lambda x.depart(\mathcal{A}e.from(e, SEA) \wedge time(e, 2PM),\ x)$

(c) What is the capital of Texas?
A: $\lambda x.capital(x) \wedge loc(x, TX)$
B: $\lambda x.capital(TX, x)$

(d) What is the population of Texas?
A: $\lambda x.equals(x, population(TX))$
B: $\lambda x.population(TX, x)$

Figure 4.1: A sample of sentences from the travel-planning and geography domains paired with two forms of semantic representation: (A) the representation used in previous work [70, 72, 110, 111], and (B) our proposed modeling approach.

event semantics [40] allows us to model "at" as an unambiguous temporal modifier. Similarly, if we adopt a formal treatment of relational nouns [41, 85], we can model "of" as a syntactic modifier for relational content provided by the associated nouns, removing the ambiguity from the original analyses of (c) and (d). In Section 5.3 we describe several such modeling choices, carefully motivating them with respect to the grammars we aim to learn.

To build such representations, we learn a factored CCG grammar [70, 99], including both the lexical entries that pair individual words and phrases with their respective meanings and the parsing model parameters. We propose a set of linguistically motivated modeling decisions that align with the underlying syntactic structures. The engineered grammar templates introduce detailed syntactic modeling of a wider range of constructions than considered in previous work, for example explicit treatments of relational nouns, Davidsonian events, and quantification, as well as treatments for elliptical sentences, as is commonly seen in natural language interface applications. Significant gains can be achieved by using such linguistically motivated grammar induction scheme. Our experimental evaluation in two benchmark domains demonstrate state-of-the-art accuracies, including up to 1.7% error reduction. The learned grammars are significantly more compact than those produced

by recent CCG learning algorithms, and generalize better in practice.

4.2 Linguistically Motivated Grammar Modeling

We make three major modifications to the representations used by [110, 111] that have direct impact on the compactness, efficiency and accuracy of learned CCG grammars.

4.2.1 Quantification

Standard treatments of quantification in first order predicate logic often require complicated scoping control, and yield analyses that can not be built from syntactically correct parses. For example, the traditional analysis followed by [110, 111] uses an existential quantifier scoped higher than the verb to define the meaning of many determiners, forcing the lexical entries to take the preceding verbs as arguments, as shown below:

$$\dfrac{\dfrac{\text{have}}{\begin{array}{c}S\backslash NP/NP\\ \lambda x\lambda y.loc(y,x)\end{array}} \quad \dfrac{\dfrac{\text{a}}{\begin{array}{c}S\backslash NP\backslash(S\backslash NP/NP)/N\\ \lambda f\lambda g\lambda x\exists y.f(y)\wedge g(y,x)\end{array}} \quad \dfrac{\text{river}}{\begin{array}{c}N\\ \lambda x.river(x)\end{array}}}{\begin{array}{c}S\backslash NP\backslash(S\backslash NP/NP)\\ \lambda g\lambda x\exists y.river(y)\wedge g(y,x)\end{array}}>}{\begin{array}{c}S\backslash NP\\ \lambda x\exists y.river(x)\wedge loc(y,x)\end{array}}<$$

and produces the overly specific lexical item $a \vdash S\backslash NP\backslash(S\backslash NP/NP)/N : \lambda f\lambda g\lambda x\exists y.f(y) \wedge g(y,x)$.

We overcome the syntactic problems associated with maintenance of existential scope by adopting generalized Skolem terms [97]. In Figure 4.2, the determiner "a" is paired with the Skolem term $\mathcal{A}$. This term implements a $\langle\langle e,t\rangle,e\rangle$ function and the entire phrase "a river" is then represented as a single entity drawn from the set of rivers. This matches the argument type of the transitive verb "have" and allows a syntactically correct parse that generalizes to other syntactic contexts.

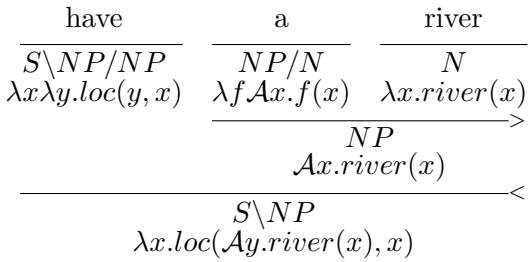


Figure 4.2: An example of a determiner introducing a skolem function

4.2.2 Davidsonian Event Semantics

In domains such as the travel-planning interface ATIS, a large portion of the sentences refer to events that may have modifiers defining properties such as time, rate and duration. In order to model this information accurately while maintaining a compact and general representation, we make use of Davidsonian event semantics [40], which allows verbs to introduce events and adverbial modifiers to be represented by predicates and linked by the shared events, so strings of adverbial modifiers are treated as conjunctive predicates.

More specifically, the Davidsonian approach assigns an *event variable* ev to each verb. The verb may then be modified with adverbial predications over this event. For example, the sentence in Figure 4.3 associates each flight with an arrival event $arrive(ev, x)$, allowing the temporal and destination information to be attached to the flight without introducing overly specific analyses such as $arrival\_time$ for the second preposition "at" - as is done by [111].

4.2.3 Relational Nouns

We model nouns as denoting a set of entities that satisfy a given property. Regular nouns are represented using unary predicates. Relational nouns syntactically function as regular nouns but semantically describe sets of entities that have some relationship with a complement [85]. For example, the relational noun *fare* describes a binary relationship between flights and their price information, as we see in Figure 4.4a.

This analysis differs from previous approaches [111], shown in Figure 4.4b, where relational

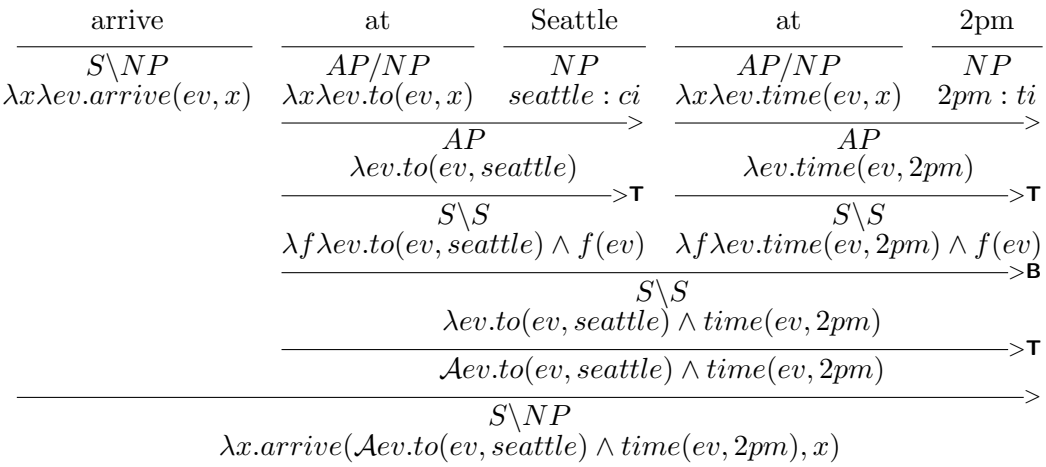
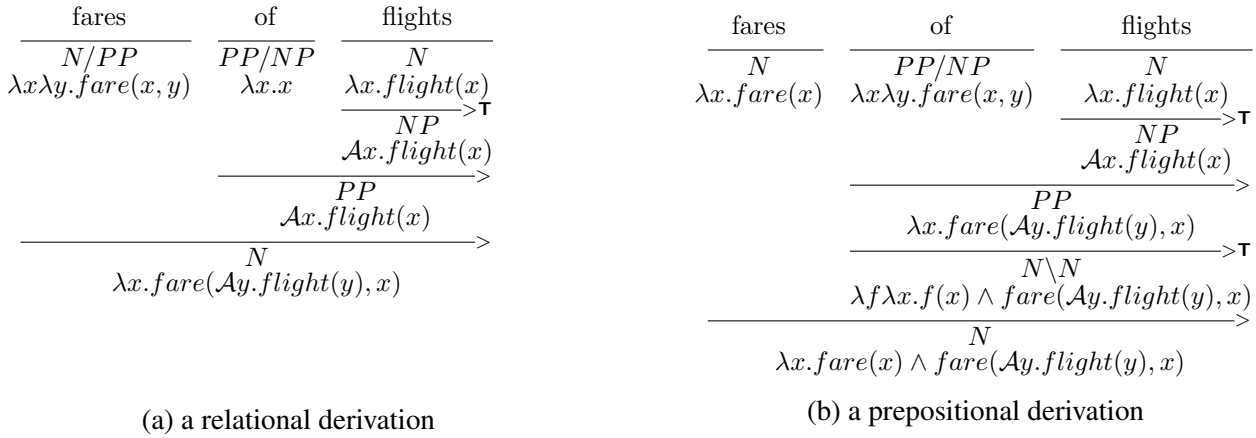


Figure 4.3: An example verb that introduces a Davidsonian event



(a) a relational derivation

(b) a prepositional derivation

Figure 4.4: Two example derivations for relational nouns

nouns were treated as regular nouns and prepositions introduced the binary relationship. The relational noun model reduces lexical ambiguity for the prepositions, which are otherwise highly polysemous.

4.3 Learning

We learn a factored CCG lexicon [72] (Section 2.3), in which the lexical entries are represented by a set of lexemes (L) and a set of templates (T) separately. A lexeme $(w, \vec{c})$ pairs a word w with

an ordered list of logical constants $\vec{c} = [c_1 \dots c_m]$. We use a lexical template to construct a lexical entry. Each template has the form

$$\lambda(\xi, \vec{v}).[\xi \vdash X : h_{\vec{v}}]$$

where ξ and $\vec{v}$ are variables that abstract over the words and logical constants that will be used to define a lexical entry with syntax X and templated logical form $h_{\vec{v}}$. To build a lexical entry, ξ is filled with the original word w and the constants in $\vec{c}$ replace the variables $\vec{v}$.

The complete lexical learning algorithm is adapted from an existing algorithm ZC07 [111], as reviewed in Section 2.5. The algorithm requires a training set $D = \{(x_i, z_i) : i = 1 \dots n\}$ where each sentence x_i is paired with a logical form z_i. In addition, we compile an initial lexicon Λ_0, which consists of a list of domain independent lexical items for function words, such as interrogative words and conjunctions, as well as lexemes with a list of NP entities complied from the database, e.g., $(Boston, [bos])$. The output is a parsing model, which includes the induced lexicon Λ and a set of parsing parameters θ. The ZC07 learning algorithm uses a function $\mathsf{GENLEX}(x, z)$ to define a set of lexical entries that could be used to parse the sentence x to construct the logical form z. For each training example (x, z), $\mathsf{GENLEX}(x, z)$ first maps each substrings in the sentence x into a set of potential lexical entries, generated by exhaustively pairing the logical constants in z using a set of hand-engineered templates. The example is then parsed with this much bigger lexicon and lexical entries from the highest scoring parses are added to Λ. The parameters Θ used to score parses are updated using perceptron. Finally, the lexemes that participate in the top scoring correct parse of x are added to the permanent lexicon Λ.

The set of templates T in our lexicon are predefined, and model the syntactic and semantic aspects of lexical entries that are shared within each word class. We diverge from previous approaches that also used hand-engineered lexical templates, as reviewed in Section 2.5, by modeling a wider range of linguistic constructions. The following two subsections describe the templates used during learning, first presenting those designed to model grammatical sentences and then a small second set designed for more elliptical spoken utterances.

4.3.1 Templates for Modeling Grammatical Languages

Table 4.1 lists the primary template set, where each row shows an example with a sentence illustrating its use. Templates are also grouped by the word classes, including adjectives, adverbs, prepositions, and several types of nouns and verbs. We detail the motivation for the design of each template below.

Verbs Verbs have the general syntactic category $S\backslash NP/\mathcal{T}$, where the function takes a subject and various type and number of arguments $\mathcal{T}$ following the verb to generate a sentential category S. We subcategorize verbs by their grammatical structures, distinguished by the valency and relationship between the arguments/complements and the verb itself, into transitive (V_{trans}), intransitive ($V_{intrans}$), impersonal ($V_{imperson}$), auxiliary (V_{aux}), modal auxiliary (V_{m_aux}) and copula (V_{copula}). A primitive event variable e is associated with verbal predicates, following the Davidsonian event-based semantics [40].

Valency is a grammatical property that classify action verbs by the number of arguments they take to complete the meaning of a predicate. The set of templates we create for action verbs have systematic correspondence to their arguments.

We first discuss transitivity, which only considers objects:

(a) What states **border** tennessee?
$border \vdash S\backslash NP/NP : \lambda x \lambda y \lambda e.border(e, y, x)$

(b) They **give** him a book.
$give \vdash S\backslash NP/NP/NP : \lambda x \lambda y \lambda z \lambda e.give(e, z, y, x)$

(c) The mississipi **runs** through Iowa.
$runs \vdash S\backslash NP : \lambda x \lambda e.run(e, x)$

Most transitive verbs take one direct object, such as "*border*" in (a), while a small set of ditransitive verbs like "*give*" in (b) may take an additional indirect object. In contrast, intransitive verbs, such as "*run*" in (c), take a subject and no object receiving the action.

The template for $V_{imperson}$ is generated to model impersonal verbs as in sentences like:

It **costs** 500 dollars to fly from boston to seattle.

$costs \vdash S\backslash NP/NP/NP : \lambda x \lambda y \lambda z \lambda e.cost(e, y, x)$

where the expletive pronoun "*it*" is used as the syntactic subject, with no semantic contribution to the predicate. The object "*to fly from boston to seattle*", which appears in apposition to the pronoun, is modeled as the semantic subject of the verb "*cost*".

Auxiliary verbs accompany main verbs to express voice, tense, aspect or mood. Unlike action verbs, auxiliary verbs are in general used to provide grammatical properties. We exemplify the use of auxiliary verbs in the following sentences:

(a) What flights **are** provided by delta airlines?

$are \vdash S\backslash NP/(S\backslash NP) : \lambda f.f$

(b) What aircraft united airline **is** using on the flights from denver?

$is \vdash S\backslash NP/(S\backslash NP) : \lambda f.f$

(c) The flights **have** arrived at boston.

$have \vdash S\backslash NP/(S\backslash NP) : \lambda f.f$

(d) He **can** catch the flights.

$can \vdash S\backslash NP/(S\backslash NP) : \lambda f.f$

(e) **Does** delta airlines provide flights from seattle?

$does \vdash S/(S\backslash NP)/N : \lambda f \lambda g.g(f)$

In example (a), (b), and (c), the auxiliary verbs help express the passive voice, the present aspect, the perfect aspect, respectively. The modal auxiliary *can* in (d) is modeled without any potential semantic content, like other auxiliaries, as intensionality is beyond the scope of this work. In (e), *does* is used to accompany the main verb *provide* to form a question, and requires the special template for the inverted auxiliary.

A copula links the subject in a sentence to a predicative expression that refer back to the subject. The expression that complements the subject may be a noun or noun phrase, or a nominal modifier, as in the following examples:

(a) The flights **are** from boston.

$are \vdash S\backslash NP/PP : \lambda f \lambda x.f(x)$

(b) which flight **is** cheap?

$is \vdash S\backslash NP/(N/N) : \lambda f \lambda x.f(\lambda y.true, x)$

(c) Alaska **is** the state with the most rivers.

$is \vdash S\backslash NP/NP : \lambda x \lambda y.equals(y, x)$

In (a) and (b), the copulas are assigned the templates to link the complement, which are prepositional phrase and adjective respectively, to the subject through intersection. In (c), the subject "*alaska*" and the complement "*the state with the most rivers*" are of an identical concept, and this identity is expressed with the predicate $equals$.

Nouns and Noun phrases We model noun phrases as e-type logical constants, which represent pre-defined entities. For example, the proper noun Boston is paired with the lexical item $Boston \vdash NP : bos$. Regular nouns define a set of entities that satisfy a property, and are mapped to $\langle e, t\rangle$-type functions, e.g., $\lambda x.flight(x)$ picks out a set of entities that applying the $flight$ function yields true. As motivated in Section 4.2, relational nouns syntactically function as regular nouns, but take a complement and describe a particular relationship. The relational nouns are characterized as $\langle e, \langle e, t\rangle\rangle$-type binary functions with an argument for the object in the relation. The complement of a relational noun may also be introduced as a possessive or prenominal modifier, as in "*delta's fares*" and "*flight schedule*", where the template takes an adjectival argument preceding the relational noun. Function nouns are modeled as a $\langle e, i\rangle$-type function.

Adjectives Adjectives are nominal modifiers that take a noun or a noun phrase and add properties in addition to the property of the nominal through conjunction. The running example in Fig 2.1 shows a derivation of the adjective "*one way*".

Prepositions Prepositions precede nominal objects and function as adjectival modifiers for nouns or adverbial modifiers for verbs.

Adverbs Adverbs are verb modifiers defining aspects like time, rate and duration. The adoption of event semantics [40] allows verbs to introduce events and adverbial modifiers to be represented by predicates and linked by the shared events, so strings of adverbial modifiers are treated as conjunctive predicates. The templates have natural account of the similarity to prepositions, except that they are functions of event type variables.

Determiners Determiners precede nouns or noun phrases and distinguish a reference of the noun. Following the generalized Skolem terms, we model determiners, including indefinite and definite articles, as a $\langle\langle e,t\rangle,e\rangle$ function that quantifies over a $\langle e,t\rangle$ set and selects a unique individual from the set.

word class	example usage	base template
Noun phrase	Boston	$\xi \vdash NP : v$
Noun (regular)	What **flight** is provided by delta?	$\xi \vdash N : \lambda x.v(x)$
Noun (relation)	I need **fares** of flights	$\xi \vdash N/PP : \lambda x\lambda y.v(x,y)$
	delta **schedule**	$\xi \vdash N\backslash(N/N) : \lambda f\lambda x.v(\mathcal{A}y.f(\lambda z.true,y),x)$
Noun (function)	**size** of California	$\xi \vdash NP/NP : \lambda x.v(x)$
$V_{intrans}$	What flights **depart** from New York?	$\xi \vdash S\backslash NP : \lambda x\lambda e.v(e,x)$
V_{trans}	Which airlines **serve** Seattle (active verb)	$\xi \vdash S\backslash NP/NP : \lambda x\lambda y\lambda e.v(e,y,x)$
	What airlines **have** flights (passive verb)	$\xi \vdash S\backslash NP/NP : \lambda x\lambda y\lambda e.v(e,x,y)$
$V_{ditrans}$	They **give** him a book	$\xi \vdash S\backslash NP/NP/NP : \lambda x\lambda y\lambda z\lambda e.v(e,z,y,x)$
$V_{imperson}$	It **costs** $500 to fly to Boston	$\xi \vdash S\backslash NP/NP/NP : \lambda x\lambda y\lambda z\lambda e.v(e,y,x)$
V_{aux}	The flights **have** arrived at Boston	$\xi \vdash S\backslash NP/(S\backslash NP) : \lambda f.f$
		$\xi \vdash S/NP/(S/NP) : \lambda f.f$
	Does delta provide flights from Seattle?	$\xi \vdash S/S : \lambda f.f$
V_{copula}	The flights **are** from Boston	$\xi \vdash S\backslash NP/PP : \lambda f\lambda x.f(x)$
	What flight **is** cheap?	$\xi \vdash S\backslash NP/(N/N) : \lambda f\lambda x.f(\lambda y.true,x)$
	Alaska **is** the state with the most rivers	$\xi \vdash S\backslash NP/NP : \lambda x\lambda y.equals(y,x)$
Adjective	I need a **one way** flight	$\xi \vdash N/N : \lambda f\lambda x.f(x) \wedge v(x)$
	Boston flights **round trip**	$\xi \vdash PP : \lambda x.v(x)$
	How **long** is mississippi?	$\xi \vdash DEG : \lambda x.v(x)$
Preposition	List flights **from** Boston	$\xi \vdash PP/NP : \lambda x\lambda y.v(y,x)$
	List flights that go **to** Dallas	$\xi \vdash AP/NP : \lambda x\lambda e.v(e,x)$
	List flights **between** Dallas and Boston	$\xi \vdash PP/NP/NP : \lambda x\lambda y\lambda z.v_1(z,x) \wedge v_2(z,y)$
	What flights leave **between** 8am and 9am?	$\xi \vdash AP/NP/NP : \lambda x\lambda y\lambda e.v_1(e,x) \wedge v_2(e,y)$
Adverb	Which flight departs **daily**?	$\xi \vdash AP : \lambda e.v(e)$
	How **early** does the flight arrive?	$\xi \vdash DEG : \lambda x.v(x)$
Determiner	Which airline has **a** flight from Boston?	$\xi \vdash NP/N : \lambda f\mathcal{A}x.f(x)$

Table 4.1: Base templates that define different syntactic roles.

4.3.2 Relaxed Grammar for Parsing Elliptical Languages

The templates presented so far model grammatically correct input. However, in dialogue domains such as ATIS, speakers often omit words. For example, speakers can drop the preposition "*from*" in "*flights from Newark to Cleveland*" to create the elliptical utterance "*flights Newark to Cleveland*". We address this issue with the templates $t_{elliptical}$ illustrated in Table 4.2. Each of these adds a binary relation P to a lexeme with a single entity typed constant. For our example, the word "*Newark*" could be assigned the lexical item $Newark \vdash PP : \lambda x.from(x, newark)$ by selecting the first template and $P = from$.

Another common problem is the use of metonymy. In the utterance "*What airlines depart from New York?*", the word "*airlines*" is used to reference flight services operated by a specific airline. This is problematic because the word "*depart*" needs to modify an event of type *flight*. We solve this with the $t_{metonymy}$ templates in Table 4.2. These introduce a binary predicate P that would, in the case of our example, map airlines on to the flights that they operate.

The templates in Table 4.2 handle the major cases of missing words seen in our data and are more efficient than the approach taken by [111] who introduced complex type shifting rules and relaxed the grammar to allow every word order.

type	example usage	base template
$t_{elliptical}$	flights **Newark** to Cleveland	$\xi \vdash PP : \lambda x.P(x, v)$
	flights arriving **2pm**	$\xi \vdash AP : \lambda e.P(e, v)$
	american airline from Denver	$\xi \vdash N : \lambda x.P(x, v)$
$t_{metonymy}$	List **airlines** from Seattle	$\xi \vdash N/PP : \lambda f \lambda x.v(x) \wedge P(\mathcal{A}y.f(y), x))$
	What **airlines** depart from Seattle?	$\xi \vdash N/(S\backslash NP) : \lambda f \lambda x.v(x) \wedge P(\mathcal{A}y.f(y), x)$
	fares from miami to New York	$\xi \vdash N/PP : \lambda f \lambda x.v(\mathcal{A}y.f(y), x)$

Table 4.2: Base templates for ungrammatical linguistic phenomena

4.3.3 Features

We use standard features for discriminating parses. Following previous work [72], we fire indicator features for the lexeme, the base template, and the complete lexical item. We also compute the standard *logical expression* features [111] on the root semantics to track the pair-wise predicate-argument relations and the co-occuring predicate-predicate relations in conjunctions and disjunctions.

4.4 Experimental Setup

4.4.1 Data and Metrics

We evaluate performance on two benchmark semantic parsing datasets, Geo880 and ATIS. We use the standard data splits, including 600/280 train/test for Geo880 and 4460/480/450 train/develop/test for ATIS. To support the new representations in Section 4.2, we systematically convert annotations with existential quantifiers, temporal events and relational nouns to new logical forms with equivalent meanings. All systems are evaluated with exact match accuracy, the percentage of fully correct logical forms.

4.4.2 Initialization

We assign positive initial weights to the indicator features for entries in the initial lexicon Λ_0, as defined in Section 4.3, to encourage their use. The elliptical template and metonymy template features are initialized with negative weights to initially discourage word skipping.

4.4.3 Comparison Systems

We compare performance with all recent CCG grammar induction algorithms that work with our datasets. This includes methods that used a limited set of hand-engineered templates for inducing the lexicon, ZC05 [110] and ZC07 [111], and those that learned grammar structure by automatically splitting the labeled logical forms, UBL [71] and FUBL [72]. We also compare the state-of-the-art for Geo880 (DCS [77] and DCS+ which includes an engineered seed lexicon) and ATIS

(which is ZC07). Finally, we include results for GUSP [86], a recent unsupervised approach for ATIS.

4.5 Results

Full Models Tables 4.3 and 4.4 present the main learning results on the ATIS and GeoQuery domains. Our approach achieves state-of-the-art accuracies on both datasets, demonstrating that the new semantic representation enables learning of better models.

System	Test
ZC05	79.3
ZC07	86.1
UBL	87.9
FUBL	88.6
DCS	87.9
FULL	89.0
DCS$^+$	91.1

Table 4.3: Exact-match Geo880 test accuracy.

System	Dev	Test
ZC07	74.4	84.6
UBL	65.6	71.4
FUBL	81.9	82.8
GUSP	-	83.5
Baseline	78.2	83.1
+Quantification	81.3	84.9
+Davidsonian	82.9	85.0
+Relational Noun	80.7	84.4
Full (+Q+D+R)	85.5	87.2

Table 4.4: Exact-match accuracy on the ATIS development and test sets.

On Geo880 the full method achieves performance comparable to the previous best systems without domain specific initialization on the test set. The highest accuracy is achieved by DCS. It

is important to note that although DCS requires less supervision, it uses extensive external signals and pairs words with semantic contents.

On the ATIS dev set, our full model outperforms the prior best system FUBL by 4.4%. We improve the highest accuracy on the test set by 3.1%. Additionally, we conduct an ablation test to study the contribution of each new semantic representation. The baseline model uses the previously annotated semantic labels for training, and constructs lexical entries using templates that do not model skolem terms, davidsonian events, and relational nouns. Each modeling decision demonstrates improvements on the learned grammar, and together creates a big performance margin over the other approaches.

4.6 Summary

Building accurate semantic parsers without prohibitive engineering costs is a long standing open research problem. While recent work offered new approaches to learning and inference, the problem of semantic modeling was often solved using opportunistic solutions customized to the domain at hand. In this work we design a linguistically motivated modeling approach that generalizes across domains. We show that not only this approach allows for learning compact models, it also improves grammar induction performance to two well-studied domains, achieving new state-of-the-art results.

Chapter 5

LEXICAL GENERALIZATION FOR CCG SEMANTIC PARSING

5.1 Introduction

Having introduced semantic representations that eliminate ambiguity from the induced lexicon, and a linguistically motivated approach for modeling the lexical templates to be used in learning a semantic parser, we can induce CCG grammar that interpretes the underlying syntactic structures of English. In this chapter, we introduce a lexical generalization model that further abstracts over systematic morphological, syntactic, and semantic alternations within word classes. This includes, for example, the facts that verbs in relative clauses and nominal constructions (e.g., "flights departing NYC" and "departing flights") should be infinitival while verbs in phrases (e.g., "What flights depart at noon?") should have tense. These grammar modeling techniques use universal, domain-independent facts about the English language to restrict word usage to appropriate syntactic contexts, and as such are potentially applicable to any semantic parsing application.

More specifically, we introduce a new *morpho-syntactic*, factored CCG lexicon that imposes our grammar restrictions during learning. Each lexical entry has (1) a word stem, automatically constructed from Wiktionary, with part-of-speech and morphological attributes, (2) a lexeme that is learned and pairs the stem with semantic content that is invariant to syntactic usage, and (3) a lexical template that specifies the remaining syntactic and semantic content. The full set of templates is defined in terms of a small set of base templates and template transformations that model morphological variants such as passivization and nominalization of verbs. This approach allows us to efficiently encode a general grammar for semantic parsing while also eliminating large classes of incorrect analyses considered by previous work.

We perform experiments in two benchmark semantic parsing datasets: GeoQuery [109] and ATIS [39]. In both cases, our approach achieves state-of-the-art performance, including a nearly

Word	Lexeme : Base Template	Trans	Lexical entry
depart	$(depart, [depart])$:	I	$depart \vdash S\backslash NP : \lambda x \lambda e.depart(e, x)$
departing		I	$departing \vdash S\backslash NP : \lambda x \lambda e.depart(e, x)$
departing	$\xi \vdash S\backslash NP : \lambda x \lambda e.v_1(e, x)$	f_{pres}	$departing \vdash PP : \lambda x \lambda e.depart(e, x)$
departure		f_{nom}	$departure \vdash N : \lambda x \lambda e.depart(e, x)$
use	$(use, [airline])$:	I	$use \vdash S\backslash NP/NP : \lambda x \lambda y \lambda e.airline(e, y, x)$
using		I	$using \vdash S\backslash NP/NP : \lambda x \lambda y \lambda e.airline(e, y, x)$
using	$\xi \vdash S\backslash NP/NP : \lambda x \lambda y \lambda e.v_1(e, y, x)$	f_{pres}	$using \vdash PP/NP : \lambda x \lambda e.airline(e, y, x)$
use		f_{nom}	$use \vdash N/NP : \lambda x \lambda y \lambda e.airline(e, y, x)$

Trans	Template Transformation
f_{pres}	$\xi \vdash S\backslash NP/\mathcal{T} : \lambda x_1..x_n \lambda e.v(e, x_n..x_1) \rightarrow \xi \vdash PP/\mathcal{T} : \lambda x_1..x_n \lambda e.v(e, x_n..x_1)$
f_{nom}	$\xi \vdash S\backslash NP/\mathcal{T} : \lambda x_1..x_n \lambda e.v(e, x_n..x_1) \rightarrow \xi \vdash N/\mathcal{T} : \lambda x_1..x_n \lambda e.v(e, x_n..x_1)$

Table 5.1: Lexical entries constructed by combining a *lexeme*, *base template*, and *transformation* for the intransitive verb 'depart' and the transitive verb 'use'.

45% relative error reduction on the ATIS test set. We also show that the gains increase with less data, including matching previous model's performance with less than 25% of the training data. Such gains are particularly practical for semantic parsers; they can greatly reduce the amount of data that is needed for each new application domain.

5.2 Morpho-Syntactic Lexicon Factorization

This section defines our morpho-syntactic lexical formalism. We present the factored representation of the lexicon, techniques for learning potential new lexemes and templates, and finally the overall learning algorithm. The lexicon is factored into 3 components, each representing a different aspect of the systematic variations we discussed in the previous section. Table 5.1 shows examples of how lexemes, templates, and morphological transformations are used to build lexical entries for example verbs. In this section, we formally define each of these components and show how they are used to specify the space of possible lexical entries that can be built for each input word. In the following two sections, we will provide more discussion of the complete sets of templates (Section 5.3) and transformations (Section 5.4).

We build on the factored CCG lexicon introduced by [72] but (a) further generalize lexemes to represent word stems, (b) constrain the use of templates with widely available syntactic informa-

Verb, Noun, Preposition, Pronoun, Adjective, Adverb, Conjunction, Numeral, Symbol, Proper Noun, Interjection, Expression

Table 5.2: Part-of-Speech types

POS	**A**ttribute	**V**alues
Noun	Number	singular, plural
Verb	Person	first, second, third
Verb	Voice	active, passive
Verb	Tense	present, past
Verb	Aspect	simple, progressive, perfect
Verb	Participle	present participle, past participle
Adj, Adv, Det	Degree of comparison	comparative, superlative

Table 5.3: Morphological attributes and values.

tion, and (c) efficiently model common morphological variations between related words.

The first step, given an input word w, is to do morphological and part-of-speech analysis with the **morpho-syntactic function** F. F maps a word to a set of possible morpho-syntactic representations, each containing a triple (s, p, m) of word stem s, part-of-speech p and morphological category m. For example, F maps the word $flies$ to two possible representations:

$$F(flies) = \{(fly, \text{Noun}, (plural)),$$
$$(fly, \text{Verb}, (third, singular, simple, present))\}$$

for the plural noun and present-tense verb senses of the word. F is defined based on the stems, part-of-speech types, and morphological attributes marked for each definition in Wiktionary. [1] The full sets of possible part-of-speech and morphological types required for our domains are shown in

[1] www.wiktionary.com

Table 5.2 and Table 5.3.

Each morpho-syntactic analysis $a \in F(w)$ is then paired with lexemes based on stem match. A **lexeme** $(s, \vec{c})$ pairs a word stem s with a list of logical constants $\vec{c} = [c_1 \dots c_k]$. Table 5.1 shows the words 'depart', 'departing', 'departure', which are all assigned the lexeme $(depart, [depart])$. In general, there can be many different lexemes for each stem, that vary in the selection of which logical constants are included.

Given analysis (s, p, m) and lexeme $(s, \vec{c})$, we can use a **lexical template** to construct a **lexical entry**. Each template has the form:

$$\lambda(\xi, \vec{v}).[\xi \vdash X : h_{\vec{v}}]$$

where ξ and $\vec{v}$ are variables that abstract over the words and logical constants that will be used to define a lexical entry with syntax X and templated logical form $h_{\vec{v}}$.

To instantiate a template, ξ is filled with the original word w and the constants in $\vec{c}$ replace the variables $\vec{v}$. For example, the template $\lambda(\xi, \vec{v}).[\xi \vdash S\backslash NP : \lambda x \lambda e.v_1(e, x)]$ could be used with the word 'departing' and the lexeme $(depart, [depart])$ to produce the lexical entry $departing \vdash S\backslash NP : \lambda x \lambda e.depart(e, x)$. When clear from context, we will omit the function signature $\lambda_p(\xi, \vec{v})$. for all templates, as seen in Table 5.1.

In general, there can be many applicable templates, which we organize as follows. Each final template is defined by applying a **morphological transformation** to one of a small set of possible **base templates**. The pairing is found based on the morphological analysis (s, p, m), where each base template is associated with part-of-speech p and each transformation is indexed by the morphology m. A transformation f_m is a function:

$$\mathsf{f}_m(\lambda_p(\xi, \vec{v}).[\xi \vdash X : h_{\vec{v}}]) = \lambda_p(\xi, \vec{v}).[\xi \vdash X' : h'_{\vec{v}}]$$

that takes the base template as input and produces a new template to model the inflected form specified by m.

For example, both base templates in Table 5.1 are for verbs. The template $\xi \vdash S\backslash NP : \lambda x \lambda e.v_1(e, x)$ can be translated into three other templates based on the transformations I, f_{pres}, and f_{nom}, depending on the analysis of the original words. These transformations generalize across word type; they can be used for the transitive verb 'use' as well as the intransitive 'depart.' Each resulting template, potentially including the original input if the identity transformation I is available, can then be used to make an output lexical entry, as we described above.

5.3 Base Templates

The templates in our lexicon, as introduced in Section 5.2, model the syntactic and semantic aspects of lexical entries that are shared within each word class. Previous approaches have also used hand-engineered lexical templates, as reviewed in Section 2.5, but we differ by (1) using more templates allowing for more fine grained analysis and (2) using word class information to restrict template use, for example ensuring that words which cannot be verbs are never paired with templates designed for verbs. Section 4.3.1 describes the templates used during learning, including those designed to model grammatical sentences and a small second set designed for more elliptical spoken utterances.

5.4 Morphological Transformations

Finally, the morpho-syntactic lexicon introduces morphological transformations, which are functions from base lexical templates to lexical templates that model the syntactic and semantic variation as the word is inflected. These transformations allow us to compactly model, for example, the facts that argument order is reversed when moving from active to passive forms of the same verb, and that the subject can be omitted. To the best of our knowledge, we are the first to study such transformations for semantic parsing.

Table 5.4 shows the transformations. Each row groups a set of transformations by linguistic category, including singular vs. plural number, active vs. passive voice, and so on, and also includes example sentences where the output templates could be used. We do not detail the motivation for every class, but it is worth looking at some of the alternations for verbs and nouns as our

Template transformations f_m	Example usage
Plural Number (f_{plural})	
I	**flight** $\rightarrow$ early **flights**
$\xi \vdash N : \lambda x.v(x) \rightarrow \xi \vdash NP : \mathcal{A}x.v(x)$	**city** $\rightarrow$ flights to **cities**
Singular Number ($\mathsf{f}_{singular}$)	
I	**flight** $\rightarrow$ **flight**
Possessive ($\mathsf{f}_{possess}$)	
$\xi \vdash NP : v \rightarrow \xi \vdash N/N : \lambda f \lambda x.f(x) \wedge P(x, v)$	**delta** $\rightarrow$ **delta's** flights
$\xi \vdash N : \lambda x.v(x) \rightarrow \xi \vdash N/N : \lambda f \lambda x.f(x) \wedge P(x, \mathcal{A}y.v(y))$	**airline** $\rightarrow$ **airline's** flights
Passive Voice ($\mathsf{f}_{passive}$)	
$\xi \vdash \mathcal{Y}/NP : \lambda x_1..x_n \lambda e.v(e, x_1..x_n) \rightarrow \xi \vdash \mathcal{Y}/PP : \lambda x_1..x_n \lambda e.v(e, x_n..x_1)$	**serves** $\rightarrow$is **served** by
$\xi \vdash \mathcal{Y}/NP : \lambda x_1..x_n \lambda e.v(e, x_1, .., x_n) \rightarrow \xi \vdash \mathcal{Y} : \lambda x_1..x_{n-1} \lambda e.v(e, x_{n-1}..x_1)$	**name** $\rightarrow$city **named** Austin
Present Participle ($\mathsf{f}_{present}$)	
$\xi \vdash S\backslash NP/\mathcal{T} : \lambda x_1..x_n \lambda e.v(e, x_n..x_1) \rightarrow \xi \vdash PP/\mathcal{T} : \lambda x_1..x_n \lambda e.v(e, x_n..x_1)$	**use** $\rightarrow$flights **using** twa
$\xi \vdash S\backslash NP : \lambda x \lambda e.v(e, x) \rightarrow \xi \vdash N/N : \lambda f \lambda x \lambda e.f(x) \wedge v(e, x)$	**arrive** $\rightarrow$**arriving** flights
$\xi \vdash S\backslash NP/\mathcal{T} : \lambda x_1..x_n \lambda e.v(e, x_n..x_1) \rightarrow \xi \vdash N/\mathcal{T} : \lambda x_1..x_n \lambda e.v(e, x_n..x_1)$	**land** $\rightarrow$ **landings** at jfk
Past Participle (f_{past})	
$\xi \vdash S\backslash NP/NP : \lambda x_1..x_n \lambda e.v(e, x_n..x_1) \rightarrow \xi \vdash PP/PP : \lambda x_1..x_n \lambda e.v(e, x_1..x_n)$	**use** $\rightarrow$ plane **used** by
Nominalization ($\mathsf{f}_{nominal}$)	
$\xi \vdash S\backslash NP/\mathcal{T} : \lambda x_1..x_n \lambda e.v(e, x_n..x_1) \rightarrow \xi \vdash N/\mathcal{T} : \lambda x_1..x_n \lambda e.v(e, x_n..x_1)$	**depart** $\rightarrow$ **departure**
Comparative (f_{comp})	
$\xi \vdash DEG : \lambda x.v(x) \rightarrow \xi \vdash PP/PP : \lambda x \lambda y.v(y) < v(x)$	**short** $\rightarrow$ **shorter**
$\xi \vdash DEG : \lambda x.v(x) \rightarrow \xi \vdash PP/PP : \lambda x \lambda y.v(y) > v(x)$	**long** $\rightarrow$ **longer**
Superlative (f_{super})	
$\xi \vdash DEG : \lambda x.v(x) \rightarrow \xi \vdash NP/N : \lambda f.argmin(\lambda x.f(x), \lambda x.v(x))$	**short** $\rightarrow$ **shortest**
$\xi \vdash DEG : \lambda x.v(x) \rightarrow \xi \vdash NP/N : \lambda f.argmax(\lambda x.f(x), \lambda x.v(x))$	**long** $\rightarrow$ **longest**

Table 5.4: Morphological transformations with examples. $\mathcal{T} = [\epsilon, NP, NP/NP]$ and $\mathcal{Y} = [S\backslash NP, S\backslash NP/NP]$ allow a single transformation to generalize across word type.

prototypical example.

Some verbs can act as noun modifiers. For example, the present participle "using" modifies "flights" in "flights *using* twa". To model this variation, we use the transformation $\mathsf{f}_{present\_part}$, a mapping that changes the root of the verb signature $S\backslash NP$ to PP:

$$\mathsf{f}_{present\_part} : \xi \vdash S\backslash NP/\mathcal{T} : \lambda x_1..x_n \lambda e.v(e, x_n..x_1) \rightarrow \xi \vdash PP/\mathcal{T} : \lambda x_1..x_n \lambda e.v(e, x_n..x_1)$$

where $\mathcal{T} = [\epsilon, NP, NP/NP]$ instantiates this rule for verbs that take different sets of arguments,

effectively allowing any verb that is in its finite or -ing form to behave syntactically like a prepositional phrase.

Intransitive present participles can also act as prenominal adjectival modifiers as in "the *departing* flight". We add a second mapping that maps the intransitive category $S\backslash NP$ to the noun modifier N/N.

$$\begin{aligned} \mathsf{f}_{present\_part}: \quad & \xi \vdash S\backslash NP : \lambda x \lambda e.v(e,x) \\ \rightarrow \quad & \xi \vdash N/N : \lambda f \lambda x \lambda e.f(x) \wedge v(e,x) \end{aligned}$$

Finally, verbal nouns have meanings derived from actions typically described by verbs but syntactically function as nouns. For example, *landing* in the phrase "landing from jfk" is the gerundive use of the verb *land*. We add the following mapping to $\mathsf{f}_{present\_part}$ and $\mathsf{f}_{nominal}$:

$$\begin{aligned} \xi \vdash S\backslash NP/\mathcal{T} : \lambda x_1..x_n \lambda e.v(e,x_n..x_1) \rightarrow \\ \xi \vdash N/\mathcal{T} : \lambda x_1..x_n \lambda e.v(e,x_n..x_1) \end{aligned}$$

with $\mathcal{T}$ from above. This allows for reuse of the same meaning across quite different syntactic constructs, including for example "flights that *depart* from Boston" and "*departure* from Boston."

Nouns can be inflected by number to denote singular and plural forms or by adding an apostrophe to mark a possessive case. The transformation function $\mathsf{f}_{\mathsf{singular}}$ is an identity transformation. Plurals may have different interpretations: one is the generic $\langle e,t \rangle$ set representation, which requires no transformation on the base, or plurals can occur without overt determiners (bare plurals), but semantically imply quantification. We create a plural to singular type shifting rule which implements the $\langle \langle e,t \rangle, e \rangle$ skolem function to select a unique individual from the set. The possessive transformation $\mathsf{f}_{\mathsf{possess}}$ transfers the base template to a noun modifier, and adds a binary predicate P that encodes the relation, as is shown in the following derivation:

$$\dfrac{\dfrac{\text{delta}'\text{s}}{\dfrac{N/N}{\lambda f \lambda x.airline(x, delta:al) \wedge f(x)}} \quad \dfrac{\text{flights}}{\dfrac{N}{\lambda x.flight(x)}}}{\dfrac{}{N}\,(>)} \\ \lambda x.airline(x, delta:al) \wedge flight(x)$$

There are also a number of instances of the identity transformation function I, which does not change the base template. Because the semantics we are constructing was designed to answer questions against a static database, it does not need to represent certain phenomena to return the correct answer. This includes more advanced variants of person, tense, aspect, and potentially many others. Ideally, these morphological attributes should add semantic modifiers to the base meaning, for example, tense can constrain the time at which an event occurs, characterized as adding predicates for the time restrictions relative to a reference time t_{ref} to the base template. For example, to model the past tense, a time index can be associated with the event, providing the relative temporal frame for interpreting time expressions.

$$\begin{aligned} \mathsf{f}_{past\_tense}: \quad & \xi \vdash S\backslash NP/\mathcal{T} : \lambda x_1..x_n \lambda e.v(e, x_n..x_1) \\ \rightarrow \quad & \xi \vdash S\backslash NP/\mathcal{T} : \lambda x_1..x_n \lambda e \lambda t.v(e, x_n..x_1, t) \wedge (t < t_{ref}) \end{aligned}$$

However, none of our domains support such reasoning, so we assign the identity transformation, and leave the exploration of these issues to future work.

5.5 Learning Factored Lexicon

Our morpho-syntactic, factored lexicon can be easily learned with small modifications to an existing algorithm [111]. We only need to modify the GENLEX procedure that defines the space of possible lexical entries. We first briefly review the ZC07 algorithm.

Given a set of training examples $D = \{(x_i, z_i) : i = 1...n\}$, x_i being the ith sentence and z_i being its annotated logical form, the algorithm learns a set of parameters Θ for the grammar, while also inducing the lexicon Λ.

The ZC07 learning algorithm uses a function $\mathsf{GENLEX}(x, z)$ to define a set of lexical entries

that could be used to parse the sentence x to construct the logical form z. For each training example (x, z), $\mathsf{GENLEX}(x, z)$ maps all substrings x to a set of potential lexical entries, generated by exhaustively pairing the logical constants in z using a set of hand-engineered templates. The example is then parsed with this much bigger lexicon and lexical entries from the highest scoring parses are added to Λ. The parameters Θ used to score parses are updated using a perceptron learning algorithm. To avoid over generation caused by exhaustive pairings, we aggressively prune all partially built logical expressions not in accordance with the label z.

For each training example (x, z), $\mathsf{GENLEX}(x, z, F)$ first maps each substring in the sentence x into the morphological representation (s, p, c) using F introduced in Section 5.2. A candidate lexeme set L' is then generated by exhaustively pairing the word stems with all subsets of the logical constants from z. Lexical templates are applied to the lexemes in L' to generate candidate lexical entries for x. Finally, the lexemes that participate in the top scoring correct parse of x are added to the permanent lexicon.

5.5.1 Initialization

Following standard practice, we compile an initial lexicon Λ_0, which consists of a list of domain independent lexical items for function words, such as interrogative words and conjunctions. These lexical items are mostly semantically vacuous and serve particular syntactic functions that are not generalizable to other word classes. We also initialize the lexemes with a list of NP entities complied from the database, e.g., $(Boston, [bos])$.

5.5.2 Features

We use two types of features in the model for discriminating parses. Four *lexical* features are fired on each lexical item: $\phi_{(s,\vec{c})}$ for the lexeme, ϕ_{t_p} for the base template, ϕ_{t_m} for the morphologically modified template, and ϕ_l for the complete lexical item. We also compute the standard *logical expression* features [111] on the root semantics to track the pair-wise predicate-argument relations and the co-occuring predicate-predicate relations in conjunctions and disjunctions. We assign a

scaling factor of 0.1 to ϕ_{t_p}, ϕ_{t_m}, and all *logical expression* features.

5.6 Experimental Setup

5.6.1 Data and Metrics

We evaluate performance on two benchmark semantic parsing datasets, Geo880 and ATIS. We use the standard data splits, including 600/280 train/test for Geo880 and 4460/480/450 train/develop/test for ATIS. To support the new representations in Section 5.3, we systematically convert annotations with existential quantifiers, temporal events and relational nouns to new logical forms with equivalent meanings. All systems are evaluated with exact match accuracy, the percentage of fully correct logical forms.

5.6.2 Initialization

We assign positive initial weights to the indicator features for entries in the initial lexicon, to encourage their use. The elliptical template and metonymy template features are initialized with negative weights to initially discourage word skipping.

5.6.3 Comparison Systems

We compare performance with all recent CCG grammar induction algorithms that work with our datasets. This includes methods that used a limited set of hand-engineered templates for inducing the lexicon, ZC05 [110] and ZC07 [111], and those that learned grammar structure by automatically splitting the labeled logical forms, UBL [71] and FUBL [72]. We also compare the state-of-the-art for Geo880 (DCS [77] and DCS+ which includes an engineered seed lexicon) and ATIS (which is ZC07). Finally, we include results for GUSP [86], a recent unsupervised approach for ATIS.

5.6.4 System Variants

We report results for a complete approach (Full), and variants which use different aspects of the morpho-syntactic lexicon. The TEMP-ONLY variant learned with the templates from Section 5.3 but, like ZC07, does not use any word class information to restrict their use. The TEMP-POS removes morphology from the lexemes, but includes the word class information from Wiktionary. Finally, we also include DCS$^+$, which initialize a set of words with POS tag JJ, NN, and NNS.

5.6.5 Learning curve

We plot learning curves for Geo880 and ATIS with variable number of labeled training examples and a fixed size test set to analyze different algorithms' relative performance on generalization. For Geo880, the training size is gradually increased from 60 to 600, with training data randomly sampled into each set without replacement. Each trained model is evaluated on the Geo880 test set. For ATIS, we use the same setup as FUBL [] for direct comparison. The experiments are run on the first 100, 500, 1000, 2000, and all 4460 training examples from the ATIS training set, and models are evaluated on the ATIS development set. All evaluations report the accuracy using the exact match criteria.

5.7 Results

Full Models Tables 5.5 and 5.6 report the main learning results. Our approach achieves state-of-the-art accuracies on both datasets, demonstrating that our new grammar induction scheme provides a type of linguistically motivated regularization; restricting the algorithm to consider a much smaller hypothesis space allows to learn better models.

On Geo880 the full method edges out the best systems by 2% absolute on the test set, as compared to other systems with no domain-specific lexical initialization. Although DCS requires less supervision, it also uses external signals including a POS tagger.

We see similarly strong results for ATIS, outperforming FUBL on the ATIS development set by 6.8%, and improving the accuracy on the test set by 7.9% over the previous best system ZC07.

System	Test
ZC05	79.3
ZC07	86.1
UBL	87.9
FUBL	88.6
DCS	87.9
FULL	90.4
DCS$^+$	91.1

Table 5.5: Exact-match Geo880 test accuracy.

System	Dev	Test
ZC07	74.4	84.6
UBL	65.6	71.4
FUBL	81.9	82.8
GUSP	-	83.5
TEMP-ONLY	85.5	87.2
FULL	87.5	91.3

Table 5.6: Exact-match accuracy on the ATIS development and test sets.

Unlike FUBL, which excels at the development set but trails ZC07's templated grammar by almost 2 points on the test set, our approach demonstrates consistent improvements on both. Additionally, although the unsupervised model (GUSP) rivals previous approaches, we are able to show that more careful use of supervision open a much wider performance gap.

Learning Curve with Ablations Figure 5.2 presents a learning curve for the ATIS domain, demonstrating that the learning improvements become even more dramatic for smaller training set sizes. Our model outperforms FUBL by wide margins, matching its final accuracy with only 22% of the total training examples. Our full model also consistently beats the variants with fewer word class restrictions, although by smaller margins. Again, these results further highlight the benefit of importing external syntactic resources and enforcing linguistically motivated constraints during learning.

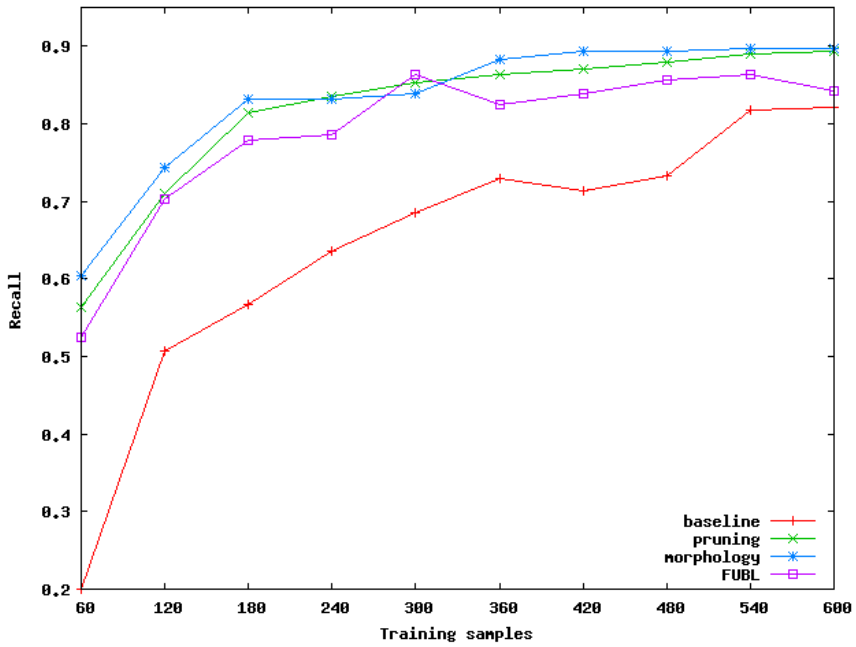


Figure 5.1: Geo880 Learning Curve

Figure 5.1 presents a learning curve for the Geo880 domain. Although the performance gap is not as dramatic as the ATIS learning curve, it still shows similar trends that learning regularization improves generalization on various training set sizes.

Learned Lexicon The learned lexicon is also more compact. Table 5.7 summarizes statistics on unique lexical entries required to parse the ATIS development set. The morpho-syntactic model uses 80.3% of the lexical entries and 63.7% of the lexemes that FUBL needs, while increase performance by nearly 7 points. Upon inspection, our model achieves better lexical decomposition by learning shorter lexical units, for example, the adoption of Davidsonian events allows us to learn unambiguous adverbial modifiers, and the formal modeling of nominalized nouns and relational nouns treats prepositions as syntactic modifiers, instead of being encoded in the semantics. Such restrictions generalize to a much wider variety of syntactic contexts.

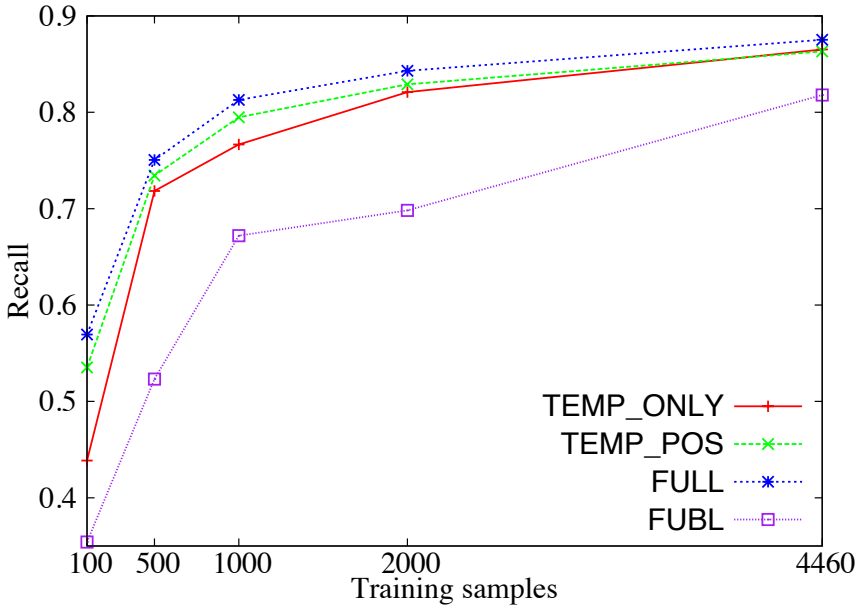


Figure 5.2: ATIS Learning Curve

System	Lexical Entries	Lexemes
FUBL	1019	721
Our Approach	818	495

Table 5.7: Lexicon size comparison on the ATIS dev set (460 unique tokens).

5.8 Summary and Future Work

We demonstrated that significant performance gains can be achieved in CCG semantic parsing by introducing a more constrained, linguistically motivated grammar induction scheme. We introduced a morpho-syntactic factored lexicon that uses domain-independent facts about the English language to restrict the number of incorrect parses that must be considered and demonstrated empirically that it enables effective learning of complete parsers, achieving state-of-the-art performance.

Because our methods are domain independent they should also benefit other semantic parsing applications and other learning algorithms that use different types of supervision, as we hope to verify in future work. We would also like to study how to generalize these gains to languages other

than English, by inducing more of the syntactic structure.

Chapter 6

WEAKLY SUPERVISED CCG GRAMMAR INDUCTION

6.1 Introduction

Statistical parsers rely heavily on the accurate syntactic analyses in treebanks to extract grammar and estimate parsing models. However, obtaining supervised training data is expensive due to the complexity of the syntactic derivations. Grammar induction becomes increasingly desirable for training statistical models, especially for low resource languages that do not have manually annotated treebanks. Considerable research effort has been made on unsupervised approaches that automatically induce grammars from part-of-speech tagged text or other types of universal or language specific knowledge [11, 13, 14, 36, 53, 64–66, 83, 93, 102], but the models employed are still far from supervised approaches, especially for recovering more complex linguistic constructions. We argue that there are strong motivations for learning parsers from gold standard dependency structures: 1. Large dependency treebanks are widely available on multiple languages due to the less costly annotation efforts. 2. Dependencies capture the underlying syntactic structure of a language. Incorporating such signals will likely improve grammar induction performance over unsupervised approaches [14, 16, 83].

In this chapter, we focus on inducing CCG syntactic structures, assuming only supervision of the universal treebank dependencies [88]. CCG is particularly difficult for this task since its categories are complex functions that specify the number and order of the arguments, which is a much larger inventory than what is required by other types of grammar formalisms, such as CFGs and dependency grammars (See Figure 6.1 and Figure 6.2 for a dependency parse and a CCG parse for the same sentence). CCG grammars encode stronger linguistic constraints and are capable of recovering more linguistic information than the overly simplistic dependency models, especially long-range dependencies arising from complex linguistic pheonomenon.

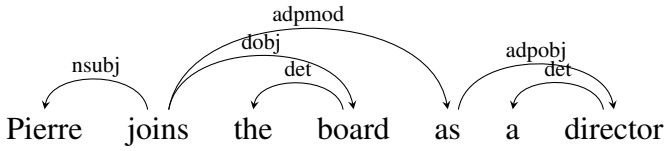


Figure 6.1: An example dependency structure.

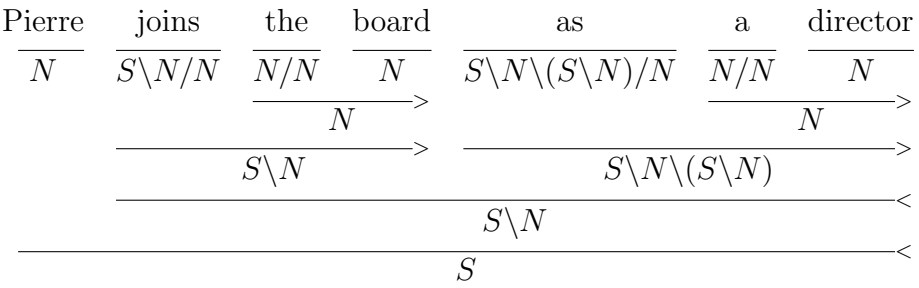


Figure 6.2: An example CCG derivation.

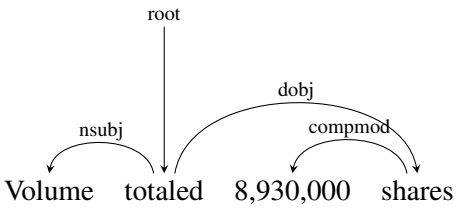
The universal treebank [88] annotates the bilexical surface dependency structures for multiple languages. It provides a consistent annotation conventions and unify the dependency types across languages makes it a homogenous resource for multilingual dependency analyses. We choose the universal dependencies as the training signals for its consistent cross-lingual labels, which allows us to easily transfer the induction scheme to other languages.

We describe a simple procedure to extract local dependencies from CCG derivations to be evaluated against the labeled dependencies in Section 6.3. In Section 6.4, we describe a log-linear model with word embedding and dependency features, and demonstrate an efficient EM-based training algorithm in Section 6.5. We present results in Section 6.6. Experiments show that dependency supervision yields substantial improvements on grammar induction over the unsupervised approaches, especially for longer sentences. Our parser achieves 81.6% correct unlabelled dependency attachments on English, and demonstrates cross-linguistically that broad syntactic structure of a language can be recovered quite successfully from widely available dependency treebanks. We provide an in-depth analysis on the constructions where the local dependency supervision breaches the underlying semantics and propose a solution to remedy this misalignment without introducing

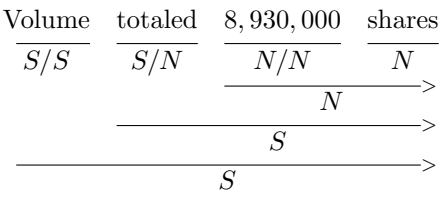
additional supervision or language specific constraints. Our induction system incorporates universal knowledge about languages, making it suitable as the first stage to generate broad scale multilingual CCG treebanks.

6.2 Lexical Induction

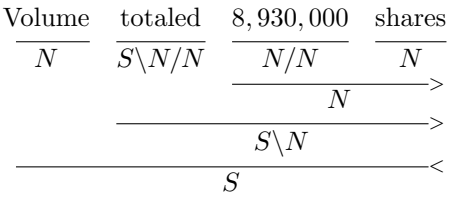
Our grammar induction algorithm is a EM-style procedure. Given the dependency parse in Figure 6.3a, we aim to learn a set of lexical entries and a parsing model that construct a CCG derivation for the sentence, as illustrated in Figure 6.3c. The algorithm visits each (sentence, dependency) pair sequentially, induces lexical categories, and then estimates model parameters using the derivations under the current lexicon.



(a) The input dependency structure



(b) CCG derivation #1



(c) CCG derivation #2

Figure 6.3: Two example CCG derivations that both produce the correct unlabeled dependency structures

6.2.1 Oracle Derivation Forest

In this section, we introduce an algorithm for mapping the gold standard dependency tree d to a set of oracle CCG derivations H, which will be used to supervise grammar induction. This con-

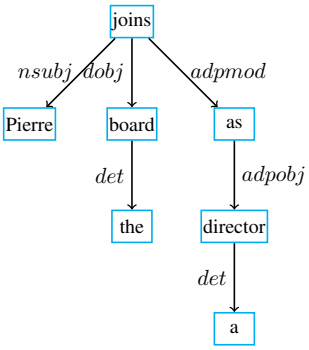
version will over-generate, producing lots of spurious parses that match the correct dependencies. For example, Figure 6.3 illustrates two oracle parses, where parse #1 fails to capture the correct linguistic constructions by treating the subject *Volume* as an adverb and letting it attach to the verb *totaled* as a modifier. Parameter estimation will later regularize the lexicon and find a small subset of good lexical entries.

Structure Representations The dependency structure captures the binary relations between words in a sentence. A standard dependency tree d of a sentence x is a spanning tree with nodes $\{x_i | i \in [0, n]\}$, where $\{x_1..x_n\}$ are words in the sentence x and x_0 is a special root node, and labeled directed dependency edges $\{(i, j, l_k) | i \neq j \wedge i \in [0, n] \wedge j \in [1, n]\}$. Each dependency edge (i, j, l_k) points from the head x_i to the dependent x_j with label l_k. $L = \{l_1...l_{|L|}\}$ is a set of predefined permissible dependency labels. A CCG parse contrasts with the dependency structure by including non-terminal phrasal nodes to encode linguistically coherent constituents that span contiguous words. Theoretically, each dependency structure may map to exponentially many CCG syntactic derivations given the multiple choices for creating the non-terminal nodes.

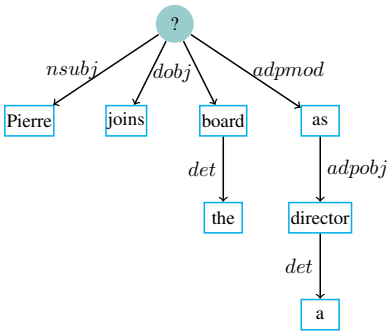
We first introduce a weighted directed hypergraph [46] to abstract the CCG syntactic structures and incorporate the notion of dependency by identifying the head of each constituent.

Definition 1. *An ordered hypergraph H is a pair $\langle V, E \rangle$, where V is a finite set of vertices, and E is a finite set of hyperarcs. Each vertex $v \in V$ is a 4-tuple $\langle c, i_h, i_b, i_e \rangle$, where c is the CCG category for the span $[i_b, i_e]$ with the span's head word at position i_h. Each hyperarc $e \in E$ is a pair $\langle T, H \rangle$, where $H \subseteq V$ is a set of head nodes and $T \subseteq V$ is a set of tail nodes.*

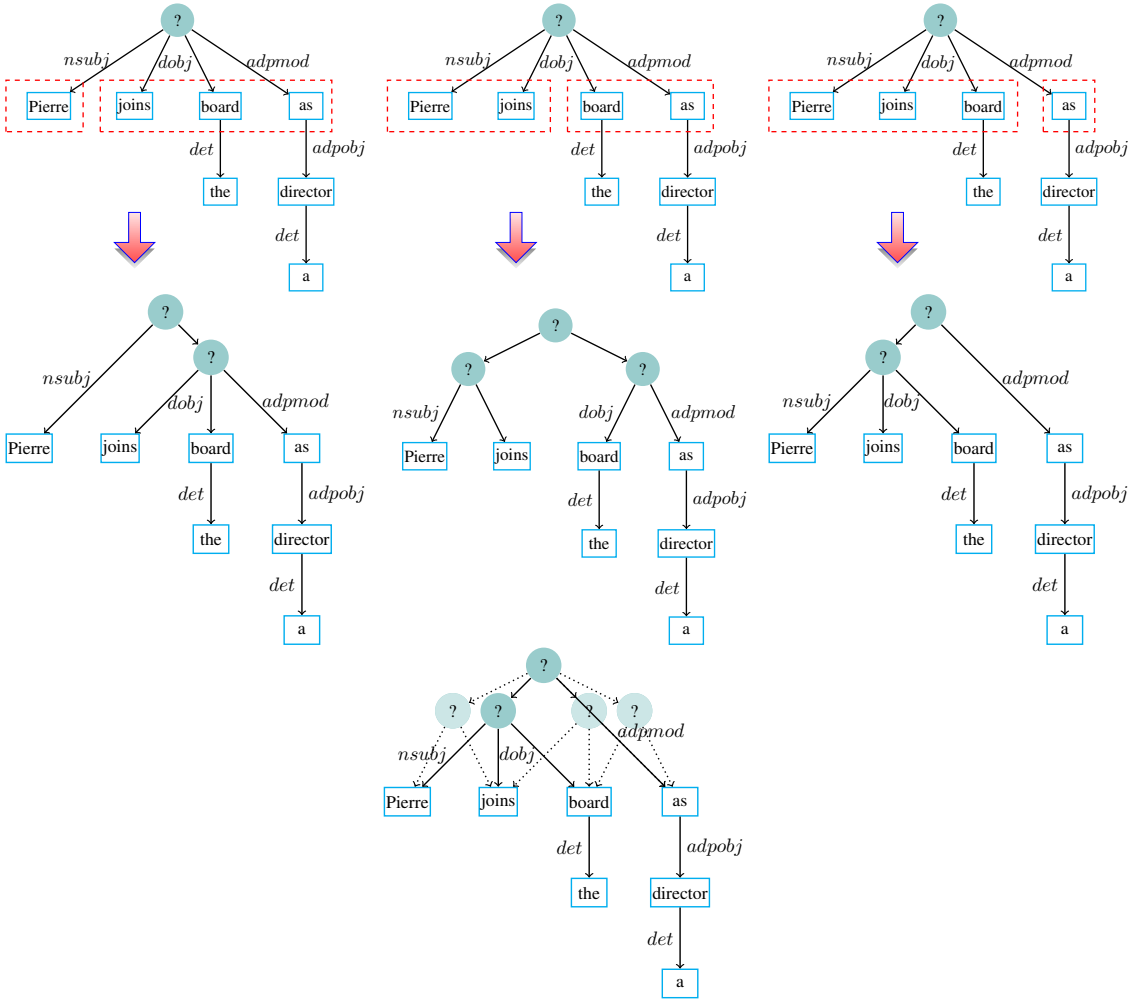
For CKY chart parsing, H contains one head node, and T contains one tail node for unary rules and two tail nodes for binary rules. $x = [x_1 \cdots x_n]$ denotes the list of tokens in a given sentence, so $i_b, i_e, i_h \in [1, n]$.



(a) The input dependency tree



(b) Create a constituent node spanning the entire sentence.



(c) Binarize the tree.

Figure 6.4: Binarizing the tree structure by splitting the tree. The set forms an *Oracle Derivation Forest*

Given this hypergraph definition, we can now define the space of all possible CCG parses that produce a set of universal dependencies. Algorithm 1 takes the dependency tree d as input, and produces the set of oracle CCG derivations that will recover d when fully induced. It recursively traverses d from the root. For every node i in d, the algorithm generates a vertex v in H that represents a constituent spanning i and all its children if any, with i being the head word. An empty vertex $(\emptyset, i_h, i_b, i_e)$ accounts for a valid constituent for the span $[i_b, i_e]$ with unknown category. Figure 6.4b illustrates the process of creating such a constituent node for the entire sentence. We then binarize the constituency structure by considering all $|J| - 1$ binary splitting points in the sorted list J. For example, splitting a list $\{j_1, j_2, ..., j_{|J|}\}$ into two sub-structures $\{j_1, ..., j_s\}$ and $\{j_{s+1}, ..., j_{|J|}\}$ at the position $1 \leq s < |J|$ creates a binary parsing step (Figure 6.4c). One type of splitting is excluded from the final candidates: any splitted sublist consisting of more than one nodes, but none being the head of the constituent. This particular binarization can never reproduce the gold head-dependent relation. The algorithm recursively creates vertices for each splitted pair, and connects them to the parent.

The resulting hypergraph H is a compact representation of all possible CCG derivations under the gold standard dependency d. Each oracle derivation outlines a sequence of valid parsing steps that will recover d when fully induced.

Algorithm 1 Initialize

Input: d, L
Output: H
1: $H \leftarrow (\emptyset, \emptyset)$
2: Init$(\{j|(0, j) \in d\}, d, L, H)$ ▷ traverse d from the root
3: **r**eturn H

6.2.2 Seed Lexicon

The oracle derivation forest H represents a set of skeleton trees, in which the empty nodes need to be labeled using CCG categories. We start by compiling a seed lexicon Λ_0 that pairs dependency labels with atomic categories. We pair all nominal labels with the category N, the root label with

Algorithm 2 Init

Input: I, h_I, d, L, H
Output: V

if $|I| = 1$ **then**
  $J \leftarrow \{j | \forall j, (i, j, k) \in d\} \bigcup I$ ▷ I and I's children
  $d \leftarrow d \setminus \{(i, j, k) | \forall j, (i, j, k) \in d\}$ ▷ remove original edges
  $i_h \leftarrow i$ ▷ mark the head
  $l \leftarrow l_k \in L$
else
  $J \leftarrow I, i_h \leftarrow h_I, l \leftarrow \emptyset$
end if
if $|J| = 1$ **then**
  $v \leftarrow \mathsf{vertex}(l, I, i_h)$ ▷ terminal node, create a vertex
else
  $v \leftarrow \mathsf{vertex}(l, J, i_h)$ ▷ create the parent vertex
  $\mathsf{sort}(J)$ ▷ sort the list according to word order
  for $(J_1, J_2) \in \mathsf{split}(J, i_h)$ **do** ▷ find the binary splittings of J
    $v_1 \leftarrow \mathsf{Init}(J_1, i_h, d, L, H)$
    $v_2 \leftarrow \mathsf{Init}(J_2, i_h, d, L, H)$ ▷ recursively create children vertices
    $E \leftarrow E \bigcup \{(\{v_1, v_2\}, \{v\})\}$ ▷ create an edge
  end for
  $V \leftarrow V \bigcup \{v\}$
end if
return V

the sentential category S, and the modifier labels with the restrictive category patterns $X\backslash X$ and X/X. $X\backslash X$ and X/X are not real categories for parsing, instead, they are restrictions on the induced categories being in such generic forms. Example modifier categories include N/N, $S\backslash S$ and $S\backslash NP/(S\backslash NP)$. The initial category assignments are summarized in Table 6.1. The seed lexicon allows us to fill partial chart, and provides a starting point for further inducing complex categories. All spans, including the ones that are assigned initial categories, are able to acquire additional categories during the induction process.

Algorithm 3 split

Input: J, i_h
Output: R

$s \leftarrow |J|$
if $s = 1$ **then**
    $R \leftarrow \emptyset$ ▷ no division for singleton set
else
    if $J_1 = i_h$ **then**
        $R \leftarrow \{(\{J_1, \cdots, J_{s-1}\}, \{J_s\})\}$
    else if $J_s = i_h$ **then**
        $R \leftarrow \{(\{J_1\}, \{J_2, \cdots, J_s\})\}$
    else
        $R \leftarrow \{(\{J_1\}, \{J_2, \cdots, J_s\}), (\{J_1, \cdots, J_{s-1}\}, \{J_s\})\}$
    end if
end if
return R

Algorithm 4 vertex

Input: $l, children, i_h$
Output: v

$c \leftarrow \mathsf{seed}(l)$ ▷ Look up categories in the seed lexicon 6.1
$i_b \leftarrow \mathsf{min_b}(children)$ ▷ begin index of the span
$i_e \leftarrow \mathsf{max_e}(children)$ ▷ end index of the span
return (c, i_h, i_b, i_e) ▷ create a vertex

Category	Dependency Labels
N	nsubj, dobj, iobj, csubj, csubjpass, nsubjpass, expl, num, det
$X\backslash X$, X/X	det, poss, adpmod, advcl, advmod, amod, neg, aux, compmod, infmod, nmod, num, partmod, rcmod, ccomp
S	ccomp, root
N	root (fragmented sentences with a non-verb root)
$EMPTY$	p

Table 6.1: Seed Lexicon

Punctuation Some punctuation marks do not imply any syntactic functions: they act simply as implicit partial bracketing constraints [94]. We initialize all punctuations with the $EMPTY$ category in the seed lexicon, which allows them to freely combine with adjacent categories with-

Algorithm 5 Induce

Input: $(x, d), \Lambda_0, \Lambda, L$
Output: (Λ, H)

$H \leftarrow \mathsf{Initialize}(d, L)$ ▷ oracle derivation forest
$H \leftarrow \mathsf{parse}(x, \Lambda_0 \bigcup \Lambda, H)$ ▷ parsing given the oracle
$\mathsf{enqueue}(H.root)$, $\Lambda \leftarrow \emptyset$
while $queue \neq \emptyset$ **do**
  $v \leftarrow \mathsf{dequeue}()$
  if $!\mathsf{explored}(v) \wedge v.c \neq \emptyset$ **then**
    if $v.i_b = v.i_e$ **then** ▷ terminal, add to lexicon
      $\Lambda \leftarrow \Lambda \bigcup \{\mathsf{x_{i_b}} \vdash \mathsf{v.c}\}$
    end if
    for $(\{v_1, v_2\}, \{v\}) \in E$ **do** ▷ loop through all valid parsing steps
      $\mathsf{enqueue}(\{v_1, v_2\})$
      $C_1 \leftarrow \mathsf{induce\_left}(v.c, v_2.c)$ ▷ induce $v_1.c$ given $v.c$ and $v_2.c$
      for $c \in C_1$ **do**
        $u \leftarrow \mathsf{vertex}(c, v_1.i_h, v_1.i_b, v_1.i_e)$
        $V \leftarrow V \bigcup \{u\}$, $E \leftarrow E \bigcup \{(\{u, v_2\}, \{v\})\}$ ▷ update H
        $\mathsf{enqueue}(u)$
      end for
      $C_2 \leftarrow \mathsf{induce\_right}(v.c, v_1.c)$ ▷ induce $v_2.c$ given $v.c$ and $v_1.c$
      for $c \in C_2$ **do**
        $u \leftarrow \mathsf{vertex}(c, v_2.i_h, v_2.i_b, v_2.i_e)$
        $V \leftarrow V \bigcup \{u\}$, $E \leftarrow E \bigcup \{(\{v_1, u\}, \{v\})\}$ ▷ update H
        $\mathsf{enqueue}(u)$
      end for
    end for
  end if
  $\mathsf{explored}(v) \leftarrow true$ ▷ mark v as explored
end while
return (Λ, H)

out altering them. Commas, colons and semicolons, however, can act as conjunctions in English when used in the context of appositives (eg. *Howard Mosher **, chief and executive officer**, said...*) and parataxis sentences (multiple sentences are conjoined by colons or semicolons, instead of us-

ing subordinating conjunctions). Allowing the punctuation marks to learn conjunction categories eliminates the need to induce a different set of grammar for the appositive phrases and subordinating sentences. In the induction process, we treat all punctuation marks as ordinary lexical entries for which CCG categories can be induced.

6.2.3 Inverse Combinatory Rules

Before we introduce the induction algorithm, we first describe a set of inverse combinatory rules for learning new categories. Each standard CCG binary combinatory rule prompts two *inverse combinators* for inducing the categories in a top-down fashion, one for the left and one for the right. For the standard forward ($>$) and backward ($<$) *application* rule:

$$X/Y \quad Y \quad \Rightarrow \quad X \qquad (>)$$

$$Y \quad X\backslash Y \quad \Rightarrow \quad X \qquad (<)$$

We create four inverse application combinatory rules:

$$LEFT \quad Y \quad \Rightarrow \quad X \qquad (>) \quad LEFT = X/Y$$

$$X/Y \quad RIGHT \quad \Rightarrow \quad X \qquad (>) \quad RIGHT = Y$$

$$LEFT \quad X\backslash Y \quad \Rightarrow \quad X \qquad (<) \quad LEFT = Y$$

$$Y \quad RIGHT \quad \Rightarrow \quad X \qquad (<) \quad RIGHT = X\backslash Y$$

CCG includes combinatory rules of forward ($>$ **B**) and backward ($<$ **B**) *composition*:

$$X/Y \quad Y/Z \Rightarrow X/Z \quad (> \mathbf{B})$$

$$Y\backslash Z \quad X\backslash Y \Rightarrow X\backslash Z \quad (< \mathbf{B})$$

for which we create four inverse composition combinatory rules:

$$LEFT \quad Y/Z \quad \Rightarrow X/Z \qquad (> \mathbf{B}) \quad LEFT = X/Y$$

$$X/Y \quad RIGHT \Rightarrow X/Z \qquad (> \mathbf{B}) \quad RIGHT = Y/Z$$

$$LEFT \quad X\backslash Y \quad \Rightarrow X\backslash Z \qquad (< \mathbf{B}) \quad LEFT = Y\backslash Z$$

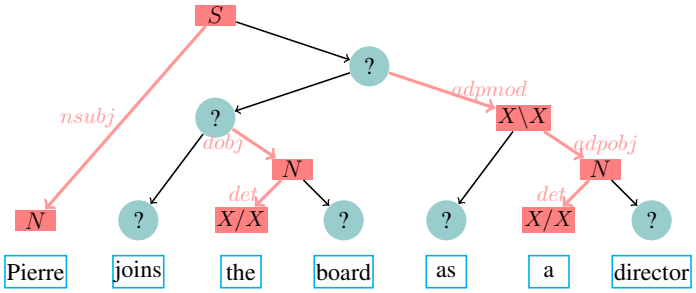
$$Y\backslash Z \quad RIGHT \Rightarrow X\backslash Z \qquad (< \mathbf{B}) \quad RIGHT = X\backslash Y$$

The inverse combinators will allow us to create new lexical categories during learning, but will over generate, as we will see. The application of these rules will be carefully constrained in the experiments to keep the induced categories tractable. In addition, although some binary rules that involve unary type shifting (eg. forward type-raising composition, defined in Section 2.1)

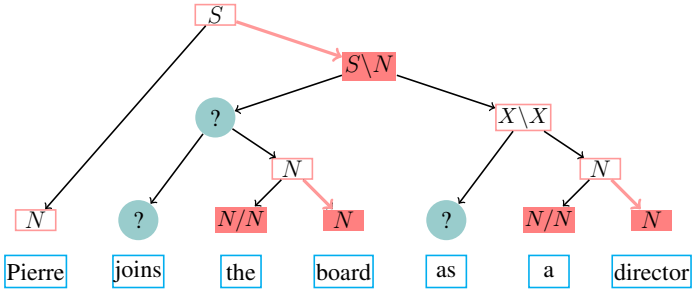
are essential to parse special linguistic constructions, their inverse combinators are disallowed to reduce the induction complexity, with the hope that the correct categories can be induced using basic combinators in different syntactic contexts.

6.2.4 Induction Algorithm

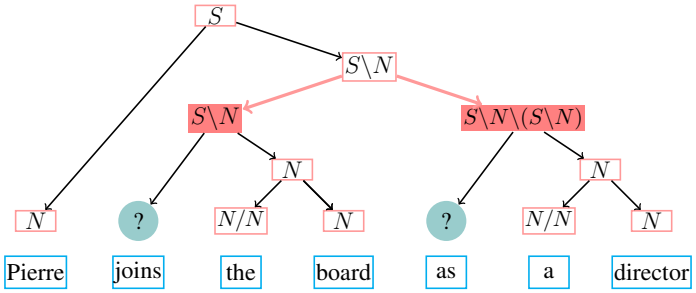
Our induction algorithm is a curriculum learning process. We follow the oracle derivation forest H, which defines a much smaller search space for the syntactic structure, to propose new lexical entries for Λ. Algorithm 5 visits every training example (x, d) sequentially. It initializes the hypergraph skeleton H first, then populates the categories for x from the current lexicon Λ and Λ_0, and partially fills in the non-terminal categories using a CKY-style parsing procedure. We illustrate a partially filled chart for the running example in Figure 6.5a. Any parse that does not conform with the oracle H is filtered. The algorithm then induces Λ by traversing vertices in the oracle H starting from the root v with the sentential category S (or N for fragmented sentences with non-verb roots). For each hyperarc $(\{v_1, v_2\}, \{v\}) \in E$ rooted at v, it proposes candidate CCG categories for v_1's span, with the categories of v and the sibling v_2 fixed, using all inverse combinatory rules from Section 6.2.3. Categories for v_2 are induced similarly by fixing v and v_1. Figure 6.5b shows the category $S\backslash N$ can be induced using the inverse combinator for backward application. The hypergraph H is updated with new vertices and edges encoding the induced categories. We repeat the induction process until the entire hypergraph H has been traversed. The resulting H defines a set of CCG derivations that align with the gold standard dependencies.



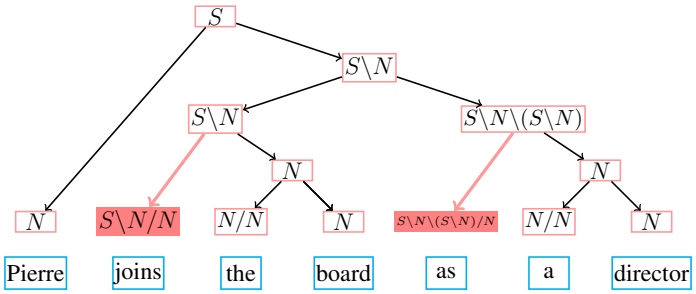
(a) parially filled derivation from Λ and Λ_0



(b) Inducing categories from parent and sibling



(c) Further inductions



(d) The fully induced derivation

Figure 6.5: Induction

6.3 Extracting Dependency Structures from CCG Derivations

Our parsing model jointly infers dependency and CCG syntactic structure. Each CCG parse is forced to deterministically produce a set of dependencies that match the labeled dependency trees. The following steps are followed to extract local dependencies from CCG parses.

- For every unary derivation step $(\{v_1\}, \{v\}) \in E$
  - propagate itself as the head of the constituent
- For every binary derivation step $(\{v_1, v_2\}, \{v\}) \in E$,
  - If composing two modifier categories ($v_1.c, v_2.c = X/XorX\backslash X$)
    - defer the dependency decision, propagate both children as candidates for the head of the constituent
  - else if both children have assigned head ($v_1.i_h \neq \emptyset \wedge v_2.i_h \neq \emptyset$)
    - If the binary combinator is forward application or forward composition rule
      - $v.i_h = v_1.i_h$ the left child is propagated as the head
    - If the binary combinator is backward application or backward composition rule
      - $v.i_h = v_2.i_h$ the right child is propagated as the head
    - If the combinator is forward type-raising composition
      - $v.i_h = v_2.i_h$ the right child is propagated as the head
    - If the head is modifier category
      - The direction of the dependency is reversed: the modifier is treated as the dependent syntactically, although it is the functor category.
  - else if one of the children has assigned head
    - create a directed dependency between the assigned head and every unassigned heads using the rules described above.

- else if both children are unassigned
    - concatenate the list
    - propagate the new list as candidates for the constituent

The extraction algorithm works for most syntactic constructions. However, unlike dependency grammar, which pays little attention to the interface between syntax and semantics, CCG aims to extract the underlying semantic structures, and provide mechanisms for recovering non-local dependencies through coindexation. Section 2.7 gives detailed description on how coindexation works in CCG predicate-argument dependencies. The semantic structure produces dependency graphs, where words can have multiple heads, which is in direct contrast to the local dependencies we use to supervise induction. As we will see, this fundamental difference essentially breaks our extraction rules and will require remediation to bridge the gap.

Function Words for wh-extraction Figure 6.6 illustrates a standard analysis for a relative clause "drafts that he submitted". The relative pronoun's category $N\backslash N/(S\backslash NP)$ consumes the clause "he submitted" and attaches to "drafts" as a modifier, and produce the long-range dependency between "submit" and "drafts" by identifying N_i and NP_i as the same entity in $N\backslash N_i/(S\backslash NP_i)$. We add semantic representation to better demonstrate how the variable x is shared between the predicate f from "submitted" and g from "drafts". Our induction algorithm is incapable of learning such a category since the universal treebanks only annotate local dependencies.

Copulas and Control Verbs vs. Auxiliary Verbs Similarly, control verbs like "attempt" in Figure 6.7 require coindexation $S\backslash N_i/(S\backslash N_i)$ to generate the long-range dependency between "submit" and "I". "attempt" takes both "to submit" and "I" as argument, and unifies the subject N_i of submit and its own subject "I". Special treatment is necessary to distinguish $S\backslash N_i/(S\backslash N_i)$ for control verbs from the auxiliary verb category $S\backslash NP/(S\backslash NP)$, which propagates "submit" as the head and produces the right local dependency.

The same error occurs when to-infinitive follows a copula, as in

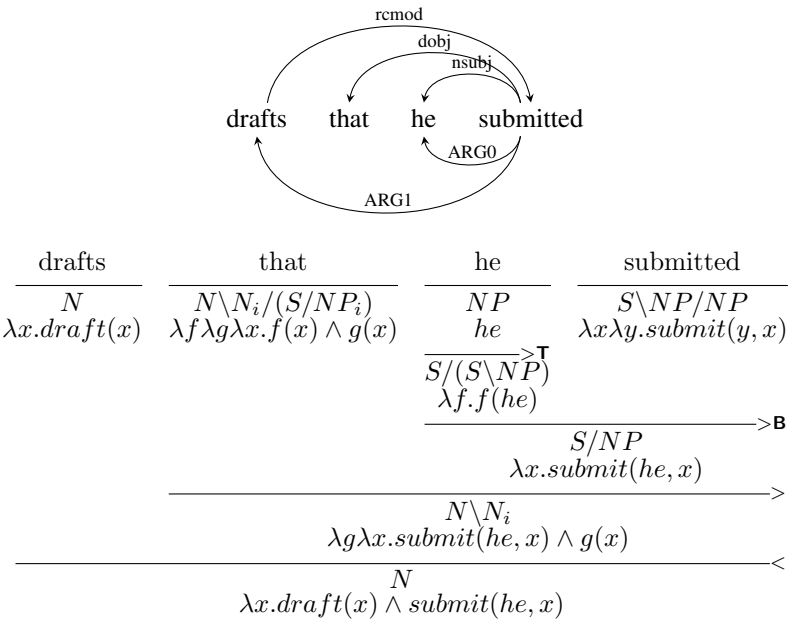
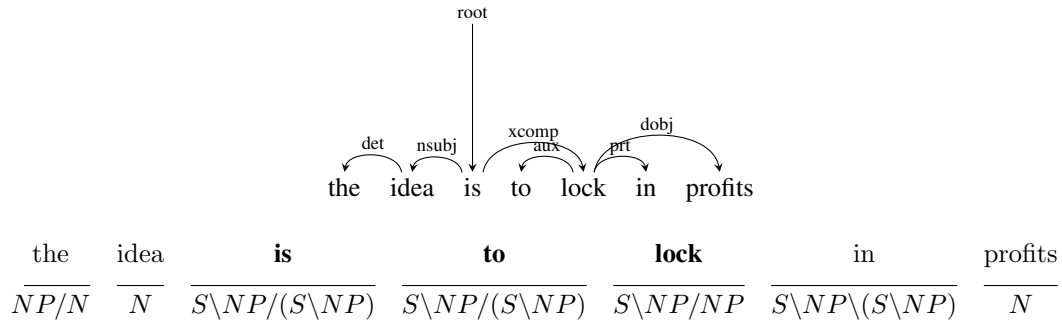
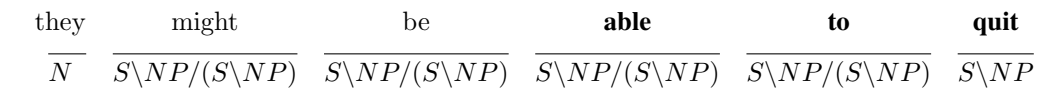


Figure 6.6: CCG derivation illustrating long-range dependency propagation through coindexation



or an adjectival complement, as in



the universal treebank annotates the copula and the adjective as the head, respectively.

These discrepancies could potentially be corrected by incorporating semantic supervision, such as the predicate-argument relations from the PropBank [84]. Identifying the long-range dependencies and the set of local dependencies they span could very much hint a category that requires coindexation. Alternatively, one can adopt a semi-supervised approach with a set of annotated gold

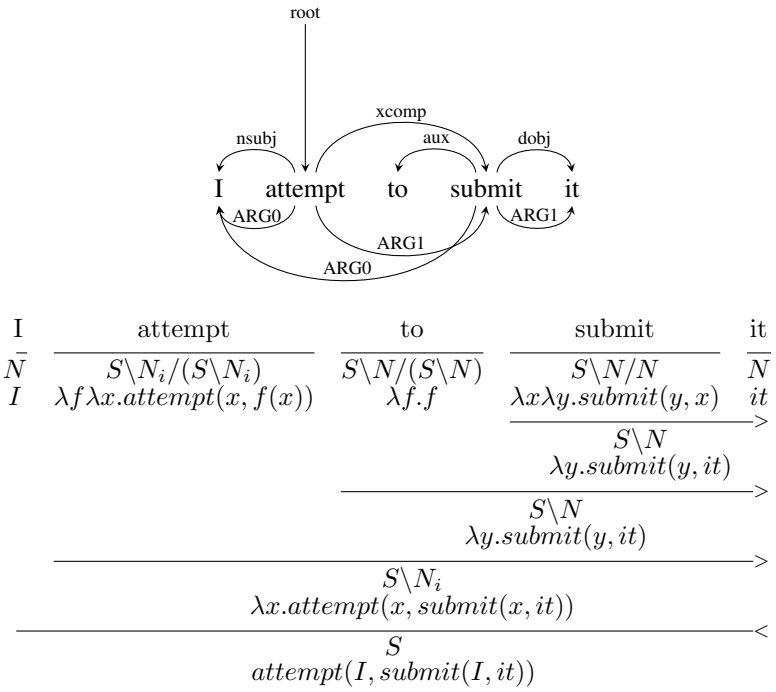


Figure 6.7: Example CCG derivation for control verb

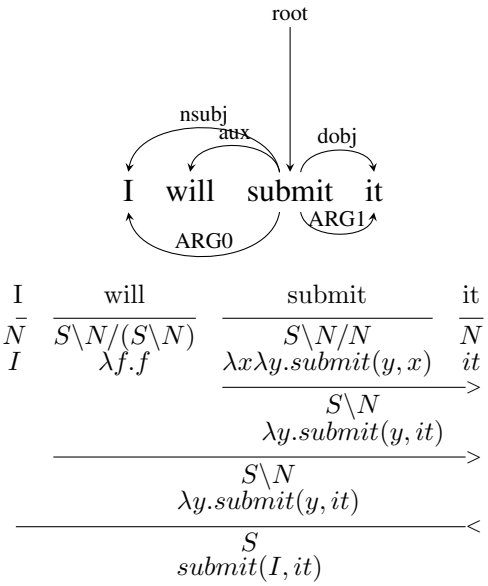


Figure 6.8: Example CCG derivation for auxiliary verb

coindexation categories for function words, like CCGBank. This includes a small set of words, but will have great impact on the overall grammar induced. Since we aim to produce CCG grammars for multiple languages, where supervising signal on semantics is not always available, we propose a simpler approach that applies more limited syntactic supervision to address the misalignments. We allow the function words to induce special categories, as illustrated in Figure 6.9, with corrected local dependencies. This broadens the grammar search space, and imposes more ambiguity during learning, but regularizing the contextual constructions will hopefully bias the model toward the semantically appropriate analyses.

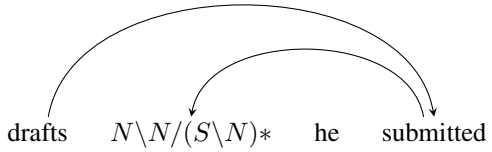


Figure 6.9: The special category $N\backslash N/(S\backslash N)*$ flips the dependency direction

6.3.1 Annotation Conventions

We also observe discrepancies resulted from the universal treebank's annotation conventions on analyzing specific constructions. These can be easily corrected by making systematic modifications.

Possessive Markers The possessive markers (Figure 6.10) contribute to another systematic discrepancies, as in CCG analysis it either becomes the head of "The SEC" if it learns the category $N/N\backslash N$ or the head of "Mr. Lane" if $N\backslash N/N$. Neither analyses align with the universal dependency. We treat possessive markers as special lexical entries that can induce flexible categories.

Coordination Coordination has always been a problematic construction in terms of dependency. The coordinating conjunction has the CCG category $X\backslash X/X$. Following our CCG dependency,

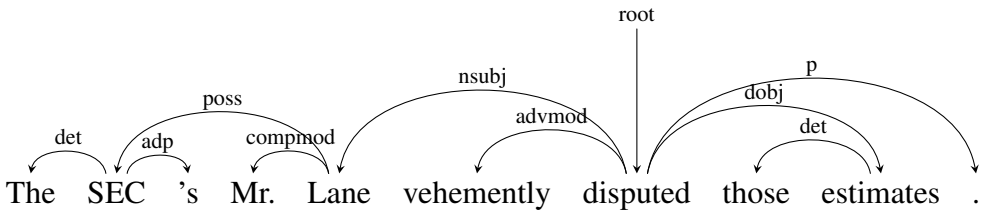


Figure 6.10: An example dependency structure for possessive markers

the first conjunct is the head of the conjunction, and the second conjunct is a dependent of the conjunction. We make explicit rules for coordination, and translate the universal treebank's format to match with the final output.

6.4 Model

We formalize the parser as a discriminative model that jointly infers CCG derivations and dependencies. The sentences are parsed using a CKY-style parsing algorithm that is extended to propagate head-dependent relations. The probability of a parse y that returns dependency d, given a sentence x is defined as:

$$P(y, d|x; \Theta, \Lambda) = \frac{e^{f(x,y,d;\Theta)}}{\sum_{d'} \sum_{y' \in \mathbf{Y}(x,d';\Lambda)} e^{f(x,y',d';\Theta)}} \quad (6.1)$$

$$f(x, y, d; \Theta) = \theta_{parse} \cdot \phi_{parse}(x, y, d) + \phi_{lex}(x, y, d; \theta_{lex}) \quad (6.2)$$

where Θ is the parameter vector, and the scoring function $f(x, y, d; \Theta)$ is decomposed into a linear model with parsing features ϕ_{parse} and the output score $\phi_{lex}(x, y, d; \theta_{lex})$ of a neural network based supertagging model adapted from [76] (Figure 6.11). ϕ_{parse} and ϕ_{lex} are described in Section 6.4.1. The overall architecture is illustrated in Figure 6.12.

We predict the most likely dependency structure $\hat{d}$ given the current model $G = (\Lambda, \Theta)$ by finding the viterbi parse $\hat{y}$:

$$\hat{y} = \arg \max_{y \in \mathbf{Y}(x;\Lambda)} P(y, d|x; \Theta, \Lambda) \quad (6.3)$$

$\hat{d}$ is the dependency structure returned by $\hat{y}$.

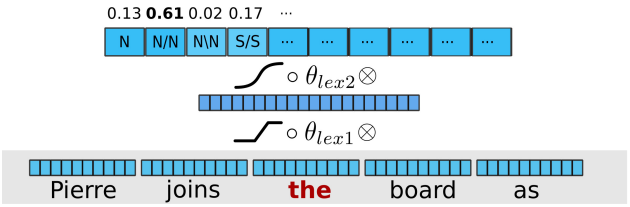


Figure 6.11: A single hidden layer supertagger using context window of size 5

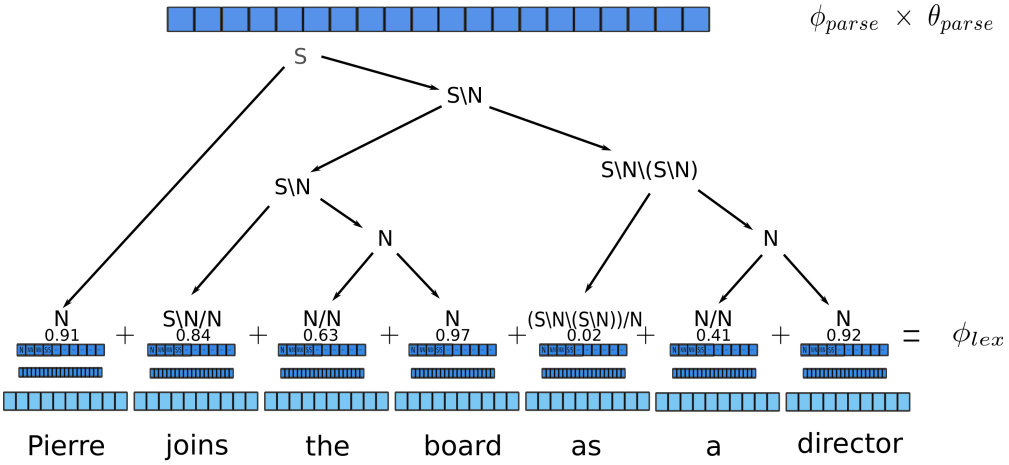


Figure 6.12: The overall parsing architecture

6.4.1 Features

We use two types of features ϕ_{lex} and ϕ_{parse} in the model to rank parses. Lexical features ϕ_{lex} are the output score from the supertagging model. We introduce three types of parsing features ϕ_{parse} to rank CCG parses and dependencies: word attachment features, POS attachment features, and CCG combination features. All features decompose over local parsing and rule instantiations for efficient calculation and dynamic programming.

Word attachment features ϕ_{word} fire on the head words of two conjoined spans. For each (w_1, w_2) pair in a parsing step, w_1 being the head word of the left constituent and w_2 of the right

constituent, we compute the dependency direction $w_1 \leftarrow w_2$ or $w_1 \rightarrow w_2$ for this parsing step using the rules in 6.3, and trigger an indicator feature $\phi_{word(w_1 \leftarrow w_2)}$ or $\phi_{word(w_1 \rightarrow w_2)}$.

POS attachment features ϕ_{pos} are similar to ϕ_{word}, but instead fire on the head words' POS tags (p_1, p_2). We have an indicator feature $\phi_{pos(p_1 \leftarrow p_2)}$ or $\phi_{pos(p_1 \rightarrow p_2)}$ following their dependency direction. We also use an indicator feature $\phi_{pos(p_1 \leftarrow p_2, r)}$ or $\phi_{pos(p_1 \rightarrow p_2, r)}$ to encode the combination rule r used in the parsing step.

CCG combination features ϕ_{ccg} are computed on the two conjoined lexical categories (c_1, c_2), their dependency direction, and the combination rule r. At each binary parsing step, we trigger $\phi_{ccg(c_1 \leftarrow c_2)}$ (or $\phi_{ccg(c_1 \rightarrow c_2)}$) and $\phi_{ccg(c_1 \leftarrow c_2, r)}$ (or $\phi_{ccg(c_1 \rightarrow c_2, r)}$). ϕ_{ccg} help regularize CCG combination rules that involve unary type shifting.

ϕ_{word} and ϕ_{pos} are scaled down with factor 0.1 and ϕ_{ccg} are scaled down with factor 0.01 to prevent them from undermining the lexical features.

6.4.2 Normal Form CCG Parsing

While CCG's expressive and flexible constituent structure makes it a preferable grammar formalism for many NLP applications, its flexibility also causes an explosion of spurious analyses, where multiple syntactic derivations map to the same semantic interpretation. To rule out certain spurious derivations and induce a more compact lexicon, our parser implements the Eisner Normal Form rules [44] to constrain basic combinations, and Hockenmaier and Bisk's [57] constraints for type shifting.

6.5 Learning

The model is trained using an EM-like algorithm. For a set of training data $D = \{(x, d)\}$, where a sentence x is paired with a dependency tree d, we induce Λ in the E-step, and optimize the parsing

model parameters θ_{parse} and the supertagging model parameters θ_{lex} alternately in the M-step.

$$\Lambda^{n+1} = \mathsf{Induce}(\{x,d\}, \Lambda^n, \Theta^n) \tag{6.4}$$

$$\theta_{lex}^{n+1} = \arg\max_{\theta_{lex}} \sum_{\{x,d\}} \log \sum_{y} P(y,d|x,\Lambda^{n+1},\theta_{parse}^{n},\theta_{lex}) \tag{6.5}$$

$$\theta_{parse}^{n+1} = \arg\max_{\theta_{parse}} \sum_{\{x,d\}} \max_{y} P(y,d|x,\Lambda^{n+1},\theta_{parse},\theta_{lex}^{n+1}) \tag{6.6}$$

The lexicon Λ is expanded using Algorithm 5 in Eq 6.4. More specifically, the parser constructs a set of full and partial derivations $\mathsf{H}(x;\Lambda,d)$ for the training example (x,d) under the current model $G=(\Lambda,\Theta)$. New lexical entries are induced based on the existing structures and added to Λ.

Eq 6.5 optimizes θ_{lex}. We compare two training regimes for the supertagging model:

1. hard EM which produces training examples using the viterbi parses $\tilde{y}$ under the current model.

$$\tilde{y} = \arg\max_{y \in \mathsf{H}(x;\Lambda,d)} P(y|x,d;\Theta,\Lambda) \tag{6.7}$$

2. n-best soft EM which weighs training examples by the interpolated inside-outside scores of the n-best lexical categories for each word.

The supertagging model is trained using weighted stochastic gradient descent with a learning rate of 0.01, optimizing for cross-entropy. The joint parsing model parameters θ_{parse} is then updated using perceptron based on the updated θ_{lex} in eq 6.6.

6.5.1 Initialization

To initialize the EM algorithm, we set the initial embedding parameters for the supertagging model (see Section 2.6 for a detailed description) by the Turian 50-dimensional word embeddings [104], and initialize the suffix and capitalization feature parameters with Gaussian noise. All parsing features ϕ_{word}, ϕ_{pos} and ϕ_{ccg} are initialized to be 0. The seed lexicon Λ_0 is compiled using dependency labels from the training (sentence, dependency) pairs $D=\{(x,d)\}$.

6.5.2 Lexicon Regularization

As illustrated in Figure 6.3, the lexical induction algorithm produces a set of oracle parses, but may not always guarantee linguistically correct analyses. To identify incorrect analyses and regularize the induced lexicon, we prune examples with infrequent lexical categories from the n-best lists. We expect that correct lexical categories will generalize better to a variety of syntactic contexts, and appear more often in the top scoring parses. We first count the number of times ($\#_{c,p}$) each lexical category c is induced for all words with POS type p and the number of times ($\#_{c,w,p}$) each lexical category c is induced for the word w with POS type p. All training examples with category c that satisfy the following condition are pruned.

$$\Big(\#_{c,w,p} < \alpha \max_{c}(\#_{c,w,p})\Big) \ \wedge \ \Big(\#_{c,p} < \beta \max_{c}(\#_{c,p})\Big) \tag{6.8}$$

6.8 prunes lexical categories occuring less than α fraction of that of the most frequent category for word w with POS p and less than β fraction of that of the most frequent category for p. We perform a grid search with cutoff threshold $\alpha = \{0.1, 0.05, 0.01\}$ and $\beta = \{0.1, 0.05, 0.01\}$.

We filter lexical categories with probability less than γ fraction of that of the best category for each word under the supertagging model. $\gamma = \{10^{-3}, 10^{-4}, 10^{-5}\}$. The max number of categories allowed for each word is restricted by the parser beam size $\mu = \{30, 50\}$.

The inverse combinators allow us to create new lexical categories during induction, but will over generate. To keep the induced categories tractable, we ensure that: 1. each new category is limited to maximal arity of 7. 2. shorter categories are prefered by assigning a negative initial score inversely proportional to its arity. 3. the results for each span are pruned with a constant beam size $\mu = \{30, 50\}$.

6.6 Experiments

6.6.1 Data and Metrics

The accuracy of the parser is evaluated on the universal treebanks [88]. For all experiments, we use the standard training/dev/test splits. The English treebank is an automatic conversion from the WSJ section 2-21/22/23 of the Penn Treebank. For better efficiency, we induce CCG categories using sentences of length 20 or less obtained from the training set. Each test sentence is parsed with the induced grammar, and the viterbi parse's dependency structure is evaluated against the label. The performance is measured as the percentage of correct unlabeled directed dependency edges (UAS). As a common practice, we do not evaluate attachments involving punctuations in the experiments.

6.6.2 Dependency Evaluation

Comparison Systems We evaluate the parser against the universal treebanks using the unlabeled attachment score, and compare the performance of our model on English against Bisk and Hockenmaier's unsupervised CCG inductions: BH12 with a basic MLE model [12], BH13 [13] which pairs the same induction algorithm with a more sophisticated nonparametric bayesian HDP model, and BH15 [14] with additional lexicalization and punctuations modeling, as well as Blunsom and Cohn's tree-substitution grammar (TSG) [15]. All systems above are unsupervised phrase-structure grammar induction approaches. We also compare with Boonkwan and Steedman's approach (BS) [16] which constructs a CCG lexicon from a simple questionnaire for linguists to assign syntactic prototypes. They only test on WSJ10 ($|x| \leq 10$), making it hard to investigate how the grammar generalizes when syntactic complexity increases. Garrette et al. [48] proposed a CCG induction scheme with no supervision on syntactic structures, but assumed access to a partial gold tag dictionary and expert knowledge for pruning, which alleviates the ambiguity at the tagging stage. Their results are not directly comparable since the model is evaluated against dependencies from CCGBank's derivations.

System	$\|x\| \leq 10$	$\|x\| \leq 20$	$\|x\| \leq \infty$
BC	68.6	-	55.7
BH12	68.2	59.6	-
BH13	70.7	63.1	58.4
BH15	71.3	65.9	62.3
BS	74.77	-	-
Our approach	86.0	81.5	77.6
+ special cats	87.6	84.0	81.6

Table 6.2: System comparison with unsupervised approaches

Results Table 6.2[1] summarizes the unlabeled dependency attachment scores for all systems tested on sentences $|x| \leq 10, 20, \infty$. The results indicate that our induced grammar consistently achieves better dependency attachments, and is significantly more robust on longer sentences than the unsupervised approaches. A key reason that the model is capable of recovering structure more accurately than previous work is that dependencies help minimize the amount of hidden structure that must be induced.

Allowing the model to acquire special categories that match the underlying semantic structures also proves to generalize better. We plot the UAS for sentences of a specific length, comparing the basic model to the special category model. Figure 6.13 depicts the average UAS for various sentence lengths, and clearly shows the performance gap between the basic model and the +special model widens when the sentences get longer, and presumably with more complex syntactic structures. The grammar induced for other languages exhibits similar trend in Figure 6.15.

[1]This is not an 100% direct comparison as the universal English treebank follows a slightly different annotation scheme. Variations between treebank annotations might have contributions on the final performance.

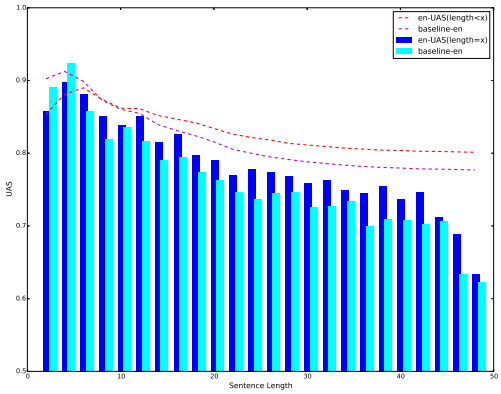


Figure 6.13: Performance on English with the basic model and the +special model

In Figure 6.14, the viterbi EM reports 84.4% oracle UAS during testing, 11% lower than the 20-best EM, and trails the 20-best EM on the predicted UAS by almost 2%. The viterbi EM tends to induce a smaller set of categories due to pruning (Table 6.3), which takes away some ambiguity for supertagging, but fails to recover certain types of linguistic constructions.

	All	Simplified	special cats	UAS for $\|x\| < \infty$
CCGBank	1640	458	-	-
viterbi EM	-	105	20	79.9
20-best EM	-	400	151	81.6

Table 6.3: Unique category types induced by viterbi EM and 20-best EM

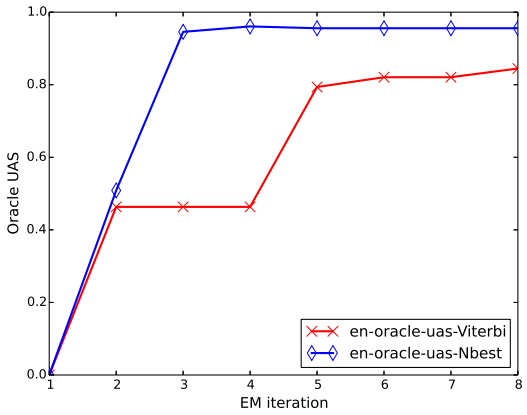


Figure 6.14: The oracle UAS during testing using Viterbi EM vs. 20-best EM

6.6.3 Multilingual Induction

We evaluate our induction algorithm on 3 additional languages in the universal treebank, French, Indonesian, and Portuguese. In each case, we use the standard split, and report the UAS of the basic model and the model with special categories for sentences of specific length. Following the same setting as English, we train only on sentences up to length 20 from the training set. Figure 6.15 depicts the generalization performance for sentences up to a certain length. Since the corpora have been standardized and relabeled for many languages in the universal treebank, numbers are not directly comparable to previous work. Similar to English, the models with special categories exhibit better generalization performance.

6.6.4 Induced Lexicon

CCG is a lexicalized formalism that encodes linguistic information like word order and number of arguments directly in the categories, which can be used to interpret the syntactic functions of the words associated with them.

As shown in Fig 6.3, dependency evaluations occassionally fail to identify linguistically incorrect analyses. For metrics that better reflect the quality of the induced grammar, we sample 100 sentences that cover a wide range of linguistic phenomena from the CCGBank [59] and compare

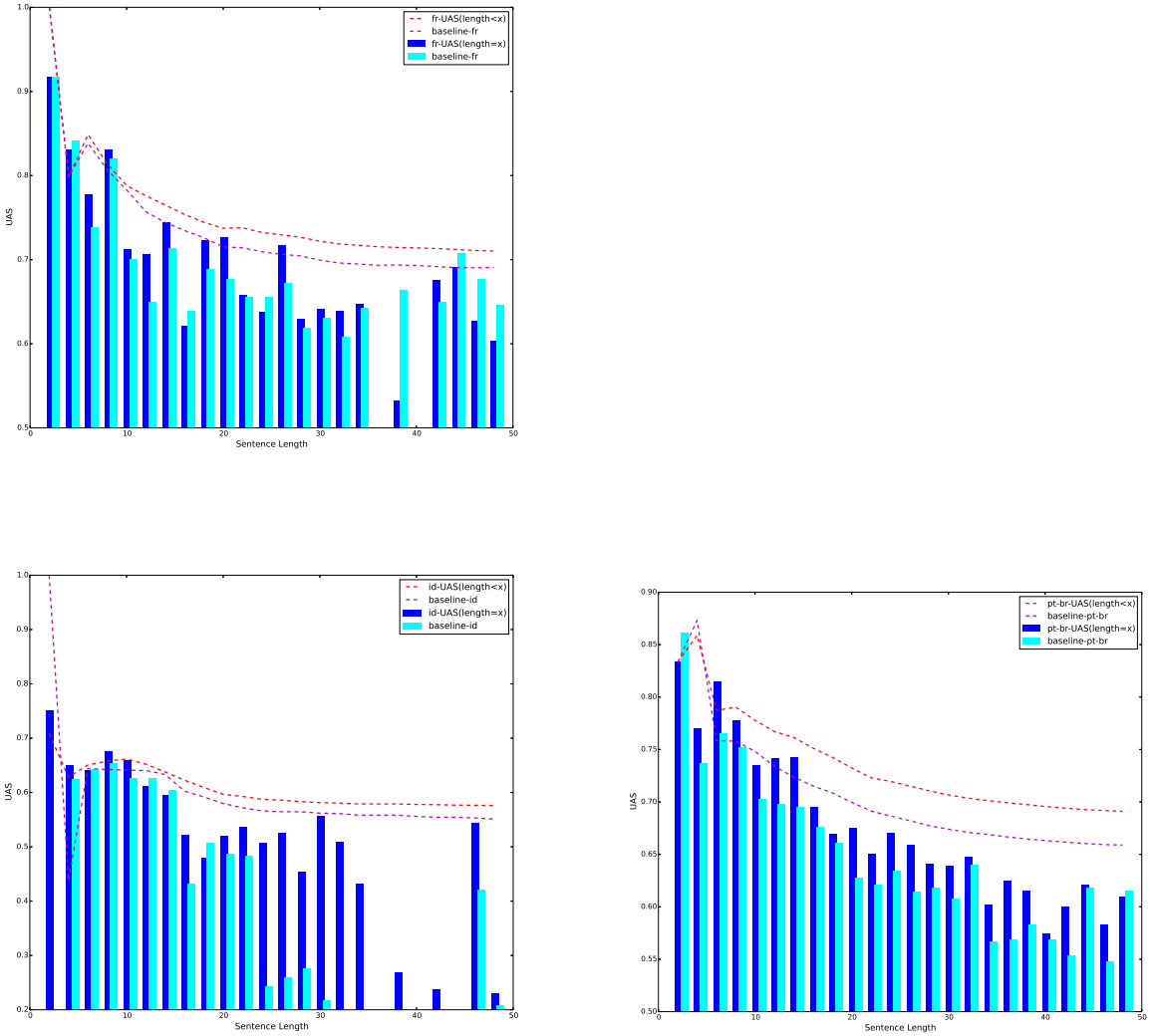
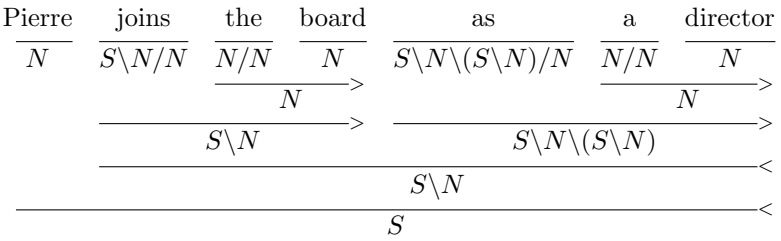


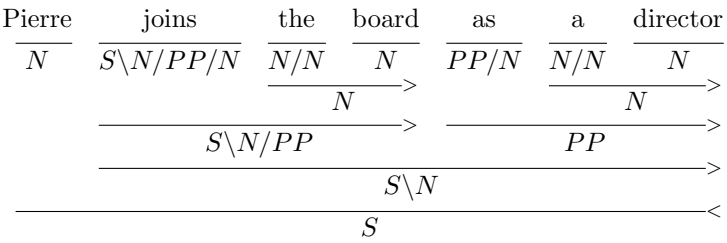
Figure 6.15: Performance with the basic model and the +special model for French, Indonesian, and Portuguese (left to right)

the lexical items in the viterbi parses with the annotated derivations. We investigate what types of linguistic constructions can be recovered by our induction algorithm given the universal dependencies. The following changes are made to resolve the systematic differences between the induced categories and the categories used in CCGBank. All subscripts that indicate the morpho-syntactic

features are removed from the lexical categories as our induced grammar does not distinguish the different types of verb phrases (eg. $S_{[to]}\backslash NP$ for to-infinitives). We accept both $S\backslash S$ and $S\backslash N\backslash(S\backslash N)$ as correct categories for verb phrase modifiers. CCGBank makes clear distinctions between complements (a obligatory argument of the predicate, see Figure 6.16b) and adjuncts (an optional argument that provides auxiliary modifications, see Figure 6.16a), but introduces inconsistencies on this issue from the automatic conversions from the Penn Treebank. Our induction adopt the adjunct treatment, and therefore excludes all sentences that are analyzed with the complement category PP in CCGBank. Overall, we are able to recover 70.2% annotated supertags in the sampled set. Control verbs, which are followed by to-infinitives and -ing forms, contribute the most to the error cases.



(a) CCG derivations illustrating adjunct analysis



(b) CCG derivations illustrating complement analysis

Figure 6.16: CCG derivations: adjunct vs. complement

Table 6.4 summarizes the top induced lexical categories for common part-of-speech in English. The learned categories mostly reflect the correct syntactic properties, for example, verbs learn different subcategorizations, adverbs and adjectives learn the appropriate modifier categories, prepositions take a noun as the argument and attach to nouns or verbs, and nouns acquire both the

nominal category N and the compound modifier category N/N.

Although our induction scheme does not aim to learn coindexation categories that recover long-range dependencies, we strive to match their syntactic signatures by allowing the model to automatically learn to correct local dependencies. Table 6.5 shows the top categories induced for the wh-words, where our algorithm originally suffered from structure mismatches. Most usages in wh-questions and relative clauses learn the special categories that fit appropriately in these syntactic contexts.

part-of-speech	**Description**	**Lexical Categories**
JJ	Adjective	N/N, N, $S\backslash N$
JJR	Adjective, comparative	N/N, N, $S\backslash N$
JJS	Adjective, superlative	N/N, N, S/N
RB	Adverb	$S\backslash N/(S\backslash N)$, S/S, $S\backslash S$
RBR	Adverb, comparative	N/N, N, $S\backslash S$
RBS	Adverb, superlative	$N/N/(N/N)$, N/N, N
NN	Noun, singular or mass	N, N/N, $N\backslash N$
NNS	Noun, Plural	N, N/N, $N\backslash(N/N)$
NNP	Proper noun, singular	N, N/N, $N\backslash N$
NNPS	Proper noun, plural	N, N/N, $N\backslash N$
VB	Verb, base form	$S\backslash N/N$, $S\backslash N$, $S\backslash S/N$
VBN	Verb, past participle	$S\backslash N$, $S\backslash N/N$, $N\backslash N$
VBP	Verb, non-3rd person singular present	$S\backslash N/(S\backslash N)$, $S\backslash N/N$, $S\backslash N$
VBZ	Verb, 3rd person singular present	$S\backslash N/(S\backslash N)$, $S\backslash N/N$, $S\backslash N$
VBG	Verb, gerund or present participle	$S\backslash N/N$, $S\backslash N$, N/N
VBD	Verb, past tense	$S\backslash N/N$, $S\backslash N$, $S\backslash N/(S\backslash N)$
DT	Determiner	N/N, N, $N\backslash N/(N\backslash N)$
RP	Particle	$S\backslash S$, $S\backslash S/N$, N
TO	to	$S\backslash S/(S\backslash S)$, $S\backslash N/(S\backslash N)$, S/S
PDT	Predeterminer	N/N, $S/S/(S/S)$, $N/N/(N/N)$
MD	Modal	$S\backslash N/(S\backslash N)$, $S/S\backslash N/(S/S\backslash N)$, S/S
IN	Preposition or subordinating conjunction	$N\backslash N/N$, $S\backslash S/N$, $S\backslash N\backslash(S\backslash N)/N$
PRP	Personal pronoun	N/N, $N\backslash N/(N\backslash N)$, $N/N/(N/N)$
PRP$	Possessive pronoun	N/N
CC	Coordinating conjunction	$N\backslash N/N$, $S\backslash S/S$, S/S
CD	Cardinal number	N/N, $N\backslash N$, N

Table 6.4: Top induced lexical categories for common part-of-speech

part-of-speech	**Example usage**	**Categories**
WP (what, who, whom)	Those **who** wrote oppose the changes	$N\backslash N/(S\backslash N)*$, N, $N/S*$
WDT (which, that)	drafts **that** he submitted	$N\backslash N/(S\backslash N)*$, N, $N\backslash N/S*$
WP$(who, whom, what)	Treasury bonds, **whose** value increases..	N/N, $N\backslash N/(N\backslash N)$
WRB (when, how, where)	**When** he finally arrived, I ...	S/S, $S/S/(S/S)$, $N\backslash N/S*$

Table 6.5: Top induced lexical categories for wh-words

6.7 Summary

We proposed a weakly supervised CCG grammar induction scheme that recovers the latent syntactic structures for multiple languages. In addition, we isolated a set of linguistic constructions that can not be modeled by local dependencies, and demonstrate that the grammar generalizes better with special categories that align with CCG's underlying semantic structures. All seed knowledge and assumptions during induction are universal language constraints, and therefore makes our induction system applicable to construct broad scale multilingual CCG treebanks.

Chapter 7

CONCLUSION

In this thesis, we focused on inducing CCG grammars for two types of parsing tasks: 1. mapping sentences to meaning representations in a domain-specific setting, and 2. analyzing the sentences′ underlying syntactic structures. To eliminate reliance on costly supervised training, various forms of universal or language specific constraints and supervising signals were explored in an effort to facilitate grammar induction from small amount of labeled training data.

In Chapter 4 and 5, we exploited CCG′s lexicalised nature, and presented a more restricted, linguistically motivated grammar induction scheme for learning a CCG semantic parser. Inspired by several linguistic theories, our grammar is engineered to align closely with the syntactic structures of English, by introducing detailed syntactic modeling of a wider range of constructions than considered in previous work, for example explicit treatments of relational nouns, Davidsonian events, and quantification, as well as relaxed grammar for parsing elliptical languages. Chapter 5 further addressed the sparsity problem caused by morphology, and introduced a *morpho-syntactic*, factored CCG lexicon that abstracts over systematic morphological, syntactic, and semantic alternations within word classes. The proposed lexical generalization model allows us to impose word usage restrictions and efficiently encode a general grammar for semantic parsing. Evaluations in two benchmark domains demonstrated state-of-the-art accuracies, highlighting the benefit of modeling systematic variations in languages and enforcing constraints during learning. The learned grammars were significantly more compact and generalized better than those produced by previous CCG learning algorithms.

In Chapter 6, we induced broad-coverage CCG syntactic structures for multiple languages, assuming supervision of bilexical dependencies, and showed how to leverage the knowledge into effectively pruning the amount of latent structure that must be explored during learning. Exper-

iments demonstrated that remediation of discrepancies between the two formalisms is essential to achieve robust grammar. The weakly supervised induction produced linguistically plausible lexicons, yielding substantial improvements over unsupervised approaches, especially for longer sentences with increasing syntactic complexity.

7.1 Future Work

There are potentially a few future directions that can be investigated following the methods discussed in this thesis, including expanding to more languages, improving grammar quality, and scaling semantic parsing.

Enhancing Grammar Induction We have demonstrated initial results on performing broad coverage syntactic grammar induction from widely available dependency treebanks. However, as shown in Chapter 6, structural discrepancies encoded in the two formalisms need to be addressed in order to improve the induced syntactic structures. Furthermore, the current induction scheme does not cover the full spectrum of CCG - categories that build long-range dependencies through coindexation are missing from the grammar space. Both problems could be resolved by provision of semantic supervision, such as the predicate-argument relations from the PropBank [84] or language specific rules. Despite comprising a small number of construction types, obtaining semantically appropriate analyses will largely prevent errors from propagating to contextual structures.

Constructing Multilingual CCG Treebanks Only a few languages currently have the luxury of an existing CCG treebank, which is essential for training supervised parsers. In this thesis, all seed knowledge and assumptions we incorporated are universal linguistic constraints, and therefore, without extensive tuning, the system can be easily transfered and deployed on another language. We have only applied the model to selected languages, but future work will aim to use the induction scheme as the first stage to generate broad scale multilingual CCG treebanks, which will potentially benefit many NLP applications. We would further look into using additional types of supervision

and language specific priors to better model CCG's semantic structures and boost accuracy for each individual language.

Improving Dependency Grammar Different grammar formalisms, equipped with their own strengths and plagued by their own weaknesses, can often improve the parsing performance of another formalism [1, 2, 38, 63, 89] when introduced as complementary information in the system. CCG is unique in encoding linguistically interpretable information like word order, subcategorization and semantic dependencies directly in the categories, making it a perfect fit to provide external assistance for dependency parsing, where direct access to these knowledge may be limited. We would hope to extend our multilingual grammar induction work, and study the effectiveness of utilizing such resources in the state-of-the-art dependency parsers.

Scaling Semantic Parsing We demonstrated that signicant performance gains can be achieved in CCG semantic parsing by introducing a more constrained, linguistically motivated grammar induction scheme. Because our methods are domain independent, they should also benefit other semantic parsing tasks, such as open domain applications, and other learning algorithms that use different types of supervision, as we hope to verify in future work. We would also like to study how to extend these gains to languages other than English, expecially morphologically rich languages, as they will potentially gain the most generalization from the *morpho-syntactic* factored lexical representation.

In addition, while the syntactically rich model helps induce grammars that generalize well, extensive amount of training in CCG and high proficiency in target languages are the key to curate templates that allow all types of constructions. Integrating the syntactic structures induced from dependencies into the engineering process will likely alleviate the required level of expertise and labor. Given the transparent mapping between CCG's syntax and semantics, it should be natural to enrich the syntactic grammar by developing strategies that automatically associate them with under-represented semantic meanings to complete the set of templates. Grounding the representations will then disambiguate the choice of syntax and possibly correct any previous mistakes. Such

a joint approach will enable a wider application of our grammar induction method to parsing in open domain, as well as across languages.

BIBLIOGRAPHY

[1] Bharat Ram Ambati, Tejaswini Deoskar, and Mark 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)*, pages 604–609, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

[2] Bharat Ram Ambati, Tejaswini Deoskar, and Mark Steedman. Improving dependency parsers using combinatory categorial grammar. In Gosse Bouma and Yannick Parmentier 0001, editors, *EACL*, pages 159–163. The Association for Computer Linguistics, 2014.

[3] Jacob Andreas, Andreas Vlachos, and Stephen Clark. Semantic parsing as machine translation. 2013.

[4] Y. Artzi and L.S. Zettlemoyer. Bootstrapping semantic parsers from conversations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2011.

[5] Yoav Artzi, Kenton Lee, and Luke Zettlemoyer. Broad-coverage ccg semantic parsing with amr. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1699–1710, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[6] Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics (TACL)*, 1:49–62, 2013.

[7] Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1(1):49–62, 2013.

[8] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2013.

[9] Taylor Berg-Kirkpatrick and Dan Klein. Phylogenetic grammar induction. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1288–1297. Association for Computational Linguistics, 2010.

[10] Alexandra Birch and Miles Osborne. Ccg supertags in factored statistical machine translation. In *In ACL Workshop on Statistical Machine Translation*, pages 9–16, 2007.

[11] Yonatan Bisk, Christos Christodoulopoulos, and Julia Hockenmaier. Labeled grammar induction with minimal supervision. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 870–876, Beijing,China, July 2015.

[12] Yonatan Bisk and Julia Hockenmaier. Simple robust grammar induction with combinatory categorial grammars. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*. Association for the Advancement of Artificial Intelligence, 2012.

[13] Yonatan Bisk and Julia Hockenmaier. An hdp model for inducing combinatory categorial grammars. *TACL*, 1:75–88, 2013.

[14] Yonatan Bisk and Julia Hockenmaier. Probing the linguistic strengths and limitations of unsupervised grammar induction. 2015.

[15] Phil Blunsom and Trevor Cohn. Unsupervised induction of tree substitution grammars for dependency parsing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1204–1213. Association for Computational Linguistics, 2010.

[16] Prachya Boonkwan and Mark Steedman. Grammar induction from text using small syntactic prototypes. In *In Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 438–446, 2011.

[17] J. Bos. Wide-coverage semantic analysis with boxer. In *Proceedings of the Conference on Semantics in Text Processing*, 2008.

[18] Johan Bos, Cristina Bosco, and Alessandro Mazzei. Converting a dependency treebank to a categorial grammar treebank for italian. In M. Passarotti, Adam Przepiórkowski, S. Raynaud, and Frank Van Eynde, editors, *Proceedings of the Eighth International Workshop on Treebanks and Linguistic Theories (TLT8)*, pages 27–38, Milan, Italy, 2009.

[19] Stephen A. Boxwell and Chris Brew. A pilot arabic ccgbank. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors, *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, may 2010. European Language Resources Association (ELRA).

[20] Stephen A. Boxwell, Dennis Mehay, and Chris Brew. Brutus: A semantic role labeling system incorporating ccg, cfg, and dependency features. In Keh-Yih Su, Jian Su, and Janyce Wiebe, editors, *ACL/IJCNLP*, pages 37–45. The Association for Computer Linguistics, 2009.

[21] S. R. K. Branavan, Luke S. Zettlemoyer, and Regina Barzilay. Reading between the lines: Learning to map high-level instructions to commands. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 1268–1277, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[22] Q. Cai and A. Yates. Large-scale semantic parsing via schema matching and lexicon extension. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2013.

[23] Q. Cai and A. Yates. Semantic parsing freebase: Towards open-domain semantic parsing. In *Proceedings of the Joint Conference on Lexical and Computational Semantics*, 2013.

[24] B. Carpenter. *Type-Logical Semantics*. The MITPress, 1997.

[25] Glenn Carroll, Glenn Carroll, Eugene Charniak, and Eugene Charniak. Two experiments on learning probabilistic dependency grammars from corpora. In *Working Notes of the Workshop Statistically-Based NLP Techniques*, pages 1–13. AAAI, 1992.

[26] Eugene Charniak. *Statistical Language Learning*. MIT Press, Cambridge, MA, USA, 1994.

[27] Danqi Chen and Christopher D Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, volume 1, pages 740–750, 2014.

[28] D.L. Chen and R.J. Mooney. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the National Conference on Artificial Intelligence*, 2011.

[29] Alexander Simon Clark. Unsupervised language acquisition: Theory and practice, 2001.

[30] Stephen Clark and James R. Curran. Wide-coverage efficient statistical parsing with ccg and log-linear models. *Comput. Linguist.*, 33(4):493–552, December 2007.

[31] Stephen Clark, Julia Hockenmaier, and Mark Steedman. Building deep dependency structures with a wide-coverage ccg parser. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 327–334, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[32] J. Clarke, D. Goldwasser, M. Chang, and D. Roth. Driving semantic parsing from the world's response. In *Proceedings of the Conference on Computational Natural Language Learning*, 2010.

[33] Shay B Cohen, David M Blei, and Noah A Smith. Variational inference for adaptor grammars. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 564–572. Association for Computational Linguistics, 2010.

[34] Shay B Cohen, Kevin Gimpel, and Noah A Smith. Logistic normal priors for unsupervised probabilistic grammar induction. In *Advances in Neural Information Processing Systems*, pages 321–328, 2008.

[35] Shay B Cohen and Noah 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*, pages 74–82. Association for Computational Linguistics, 2009.

[36] Trevor Cohn, Phil Blunsom, and Sharon Goldwater. Inducing tree-substitution grammars. *Journal of Machine Learning Research*, 2011.

[37] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011.

[38] Gregory F. Coppola and Mark Steedman. The effect of higher-order dependency features in discriminative phrase-structure parsing. In *ACL (2)*, pages 610–616. The Association for Computer Linguistics, 2013.

[39] Deborah A Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. Expanding the scope of the atis task: The atis-3 corpus. In *Proceedings of the workshop on Human Language Technology*, 1994.

[40] D. Davidson. The logical form of action sentences. *Essays on actions and events*, pages 105–148, 1967.

[41] Jos de Bruin and Remko Scha. The interpretation of relational nouns. In *Proceedings of the Conference of the Association of Computational Linguistics*, pages 25–32. ACL, 1988.

[42] Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 1370–1380, 2014.

[43] Gregory Druck, Gideon Mann, and Andrew 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: Volume 1-Volume 1*, pages 360–368. Association for Computational Linguistics, 2009.

[44] Jason Eisner. Efficient normal-form parsing for combinatory categorial grammar. In *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics*, ACL '96, pages 79–86, Stroudsburg, PA, USA, 1996. Association for Computational Linguistics.

[45] Dominic Espinosa, Michael White, and Dennis Mehay. Hypertagging: Supertagging for surface realization with ccg. In *Proceedings of ACL-08: HLT*, pages 183–191, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[46] Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete Appl. Math.*, 42(2-3):177–201, April 1993.

[47] Kuzman Ganchev, Jennifer Gillenwater, and Ben 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: Volume 1 - Volume 1*, ACL '09, pages 369–377, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[48] Dan Garrette, Chris Dyer, Jason Baldridge, and Noah A. Smith. Weakly-supervised grammar-informed bayesian CCG parser learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 2246–2252, 2015.

[49] Daniel Gildea and Julia Hockenmaier. Identifying semantic roles using combinatory categorial grammar. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, EMNLP '03, pages 57–64, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[50] Jennifer Gillenwater, Kuzman Ganchev, Joao Graça, Fernando Pereira, and Ben Taskar. Sparsity in dependency grammar induction. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 194–199. Association for Computational Linguistics, 2010.

[51] D. Goldwasser and D. Roth. Learning from natural instructions. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2011.

[52] Aria Haghighi and Dan Klein. Prototype-driven grammar induction. In *In Proceedings of ACL06*, 2006.

[53] William P. Headden III, Mark Johnson, and David McClosky. Improving unsupervised dependency parsing with richer contexts and smoothing. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, 6 2009.

[54] Larry Heck, Dilek Hakkani-Tür, and Gokhan Tur. Leveraging knowledge graphs for web-scale unsupervised semantic parsing. In *Proc. of the INTERSPEECH*, 2013.

[55] Jerry R Hobbs, Mark Stickel, Paul Martin, and Douglas Edwards. Interpretation as abduction. In *Proceedings of the Association for Computational Linguistics*, 1988.

[56] Julia Hockenmaier. Creating a ccgbank and a wide-coverage ccg lexicon for german. In *In Proc. of the 44th Annual Meeting of the ACL and 21st International Conference on Computational Linguistics (COLING/ACL-2006*, pages 505–512, 2006.

[57] Julia Hockenmaier and Yonatan Bisk. Normal-form parsing for combinatory categorial grammars with generalized composition and type-raising. In Chu-Ren Huang and Dan Jurafsky, editors, *COLING*, pages 465–473. Tsinghua University Press, 2010.

[58] Julia Hockenmaier and Mark Steedman. Generative models for statistical parsing with combinatory categorial grammar. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 335–342, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[59] Julia Hockenmaier and Mark Steedman. Ccgbank: A corpus of ccg derivations and dependency structures extracted from the penn treebank. *Comput. Linguist.*, 33(3):355–396, September 2007.

[60] Matthew Honnibal, Jonathan K Kummerfeld, and James R Curran. Morphological analysis can improve a ccg parser for english. In *Proceedings of the International Conference on Computational Linguistics*, 2010.

[61] Bevan K. Jones, Mark Johnson, and Sharon Goldwater. Semantic parsing with bayesian tree transducers. In *Proceedings of Association of Computational Linguistics*, 2012.

[62] R.J. Kate and R.J. Mooney. Using string-kernels for learning semantic parsers. In *Proceedings of the Conference of the Association for Computational Linguistics*, 2006.

[63] Sunghwan Mac Kim, Dominick Ng, Mark Johnson, and James Curran. Improving combinatory categorial grammar parse reranking with dependency grammar features. In *Proceedings of COLING*, pages 1441–1458, 2012.

[64] Dan Klein. The unsupervised learning of natural language structure. *Ph.D. Thesis*, 3 2005.

[65] Dan Klein and Christopher D. Manning. A generative constituent-context model for improved grammar induction. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 128–135. Association for Computational Linguistics, 2002.

[66] Dan Klein and Christopher D. Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

[67] J. Krishnamurthy and T. Mitchell. Weakly supervised training of semantic parsers. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2012.

[68] Jayant Krishnamurthy and Thomas Kollar. Jointly learning to parse and perceive: Connecting natural language to the physical world. *Transactions of the Association for Computational Linguistics*, 1(2), 2013.

[69] T. Kwiatkowski, S. Goldwater, L. Zettlemoyer, and M. Steedman. A probabilistic model of syntactic and semantic acquisition from child-directed utterances and their meanings. *Proceedings of the Conference of the European Chapter of the Association of Computational Linguistics*, 2012.

[70] Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. Scaling semantic parsers with on-the-fly ontology matching. 2013.

[71] Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2010.

[72] Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. Lexical generalization in CCG grammar induction for semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2011.

[73] Mike Lewis, Luheng He, and Luke Zettlemoyer. Joint a* ccg parsing and semantic role labelling. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Lan-*

guage Processing, pages 1444–1454, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[74] Mike Lewis and Mark Steedman. Combined distributional and logical semantics. *Transactions of the Association for Computational Linguistics*, 1:179–192, 2013.

[75] Mike Lewis and Mark Steedman. A* ccg parsing with a supertag-factored model. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 990–1000, Doha, Qatar, October 2014. Association for Computational Linguistics.

[76] Mike Lewis and Mark Steedman. Improved CCG parsing with semi-supervised supertagging. *TACL*, 2:327–338, 2014.

[77] P. Liang, M.I. Jordan, and D. Klein. Learning dependency-based compositional semantics. In *Proceedings of the Conference of the Association for Computational Linguistics*, 2011.

[78] David Mareček and Zdeněk 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*, EMNLP-CoNLL '12, pages 297–307, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[79] Cynthia Matuszek, Nicholas FitzGerald, Luke Zettlemoyer, Liefeng Bo, and Dieter Fox. A joint model of language and perception for grounded attribute learning. In *Proceedings of the International Conference on Machine Learning*, 2012.

[80] Igor Mel'čuk. *Dependency Syntax: Theory and Practice*. State University of New York Press, 1988.

[81] Scott Miller, David Stallard, Robert Bobrow, and Richard Schwartz. A fully statistical approach to natural language interfaces. In *Proceedings Association for Computational Linguistics*, 1996.

[82] S. Muresan. Learning for deep language understanding. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2011.

[83] Tahira Naseem, Harr Chen, Regina Barzilay, and Mark Johnson. Using universal linguistic knowledge to guide grammar induction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2010.

[84] Martha Palmer, Daniel Gildea, and Paul Kingsbury. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–105, 2005.

[85] Barbara H Partee and Vladimir Borschev. Integrating lexical and formal sematics: Genitives, relational nouns, and type-shifting. In *Proceedings of the Second Tbilisi Symposium on Language, Logic, and Computation*, 1998.

[86] Hoifung Poon. Grounded unsupervised semantic parsing. In *Association for Computational Linguistics (ACL)*, 2013.

[87] James Pustejovsky. The generative lexicon. volume 17, 1991.

[88] Yvonne Quirmbach-Brundage Yoav Goldberg Dipanjan Das Kuzman Ganchev Keith Hall Slav Petrov Hao Zhang Oscar Tckström Claudia Bedini Núria Bertomeu Castelló Jungmee Lee Ryan McDonald, Joakim Nivre. Universal dependency annotation for multilingual parsing. In *Proceedings of the ACL 2013*. Association for Computational Linguistics, August 2013.

[89] Kenji Sagae and Yusuke Miyao. Hpsg parsing with shallow dependency constraints. In *In Proc. ACL 2007*, 2007.

[90] Noah Ashton Smith and Jason Eisner. *Novel estimation methods for unsupervised discovery of latent structure in natural language text*, volume 67. 2007.

[91] Benjamin Snyder, Tahira Naseem, and Regina 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: Volume 1 - Volume 1, ACL '09, pages 73–81, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[92] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer, 2013.

[93] Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel 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*, HLT '10, pages 751–759, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[94] Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. Bootstrapping dependency grammar inducers from incomplete sentence fragments via austere models. In Jeffrey Heinz, Colin de la Higuera, and Tim Oates, editors, *ICGI*, volume 21 of *JMLR Proceedings*, pages 189–194. JMLR.org, 2012.

[95] Valentin I. Spitkovsky, Hiyan Alshawi, Daniel Jurafsky, and Christopher D. Manning. Viterbi training improves unsupervised dependency parsing. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, CoNLL '10, pages 9–17, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[96] M. Steedman. *Surface Structure and Interpretation*. The MIT Press, 1996.

[97] M. Steedman. *The Syntactic Process*. The MIT Press, 2000.

[98] M. Steedman. *Taking Scope*. The MIT Press, 2011.

[99] M. Steedman and J. Baldridge. Combinatory categorial grammar. *Non-Transformational Syntax*, pages 181–224, 2003.

[100] Emily Thomforde and Mark Steedman. Semi-supervised ccg lexicon extension. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1246–1256, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[101] Daniel Tse and James R. Curran. Chinese ccgbank: Extracting ccg derivations from the penn chinese treebank. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 1083–1091, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[102] Kewei Tu and Vasant Honavar. On the utility of curricula in unsupervised learning of probabilistic grammars. *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, 22(1):1523, 2011.

[103] Gokhan Tur, Anoop Deoras, and Dilek Hakkani-Tur. Semantic parsing using word confusion networks with conditional random fields. In *Proc. of the INTERSPEECH*, 2013.

[104] Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics, 2010.

[105] Adam Vogel and Dan Jurafsky. Learning to follow navigational directions. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 806–814, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[106] Y.W. Wong and R.J. Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the Conference of the Association for Computational Linguistics*, 2007.

[107] Wenduan Xu, Stephen Clark, and Yue Zhang. Shift-reduce ccg parsing with a dependency model. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 218–227. Association for Computational Linguistics, 2014.

[108] Xuchen Yao and Benjamin Van Durme. Information extraction over structured data: Question answering with freebase. In *Association for Computational Linguistics (ACL)*, 2014.

[109] J.M. Zelle and R.J. Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the National Conference on Artificial Intelligence*, 1996.

[110] L.S. Zettlemoyer and M. Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2005.

[111] L.S. Zettlemoyer and M. Collins. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2007.

[112] L.S. Zettlemoyer and M. Collins. Learning context-dependent mappings from sentences to logical form. In *Proceedings of the Joint Conference of the Association for Computational Linguistics and International Joint Conference on Natural Language Processing*, 2009.